

MOBILE COMPUTING

edited by
TOMASZ IMIELIŃSKI
HENRY F. KORTH

Kluwer Academic Publishers

MOBILE COMPUTING

**THE KLUWER INTERNATIONAL SERIES
IN ENGINEERING AND COMPUTER SCIENCE**

MOBILE COMPUTING

edited by

Tomasz Imielinski
Rutgers University

Henry F. Korth
AT&T Bell Laboratories



KLUWER ACADEMIC PUBLISHERS
Boston / Dordrecht / London

Distributors for North America:

Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA

Distributors for all other countries:

Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available
from the Library of Congress.

Copyright © 1996 by Kluwer Academic Publishers

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher, Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061

Printed on acid-free paper.

Printed in the United States of America

CONTENTS

PREFACE	xxi
1 INTRODUCTION TO MOBILE COMPUTING	
<i>Tomasz Imielinski and Henry F. Korth</i>	1
1 Introduction	1
2 Technology Overview	3
3 Research Issues	17
4 Book Content	33
REFERENCES	39
2 THE PARCTAB UBIQUITOUS COMPUTING EXPERIMENT	
<i>Roy Want, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, and Mark Weiser</i>	45
1 Introduction	45
2 Ubiquitous Computing	47
3 PARCTAB System Design	50
4 User-Interface Design for Palm-Sized Computers	57
5 PARCTAB System Architecture	63
6 Developing System and Application Components	71
7 A Classification of PARCTAB Applications	75
8 Experiences with the PARCTAB System	81
9 Conclusion	91
REFERENCES	97

**3 SCALABLE SUPPORT FOR TRANSPARENT
MOBILE INTERNETWORKING**

<i>David B. Johnson</i>	103
1 Introduction	103
2 Problem Analysis	105
3 The Basic Mobile IP Protocol	108
4 Route Optimization	115
5 Protocol Scalability	121
6 Conclusion	124
REFERENCES	125

**4 LOCATION MANAGEMENT FOR
NETWORKS WITH MOBILE USERS**

<i>B. R. Badrinath and Tomasz Imielinski</i>	129
1 Introduction	129
2 Location Management in the Internet	133
3 Location Management in Cellular Telephone Networks and in PCN	141
4 Performance Issues	145
5 Future: Adaptive Location management	147
6 Conclusions	150
REFERENCES	150

**5 DYNAMIC SOURCE ROUTING IN AD HOC
WIRELESS NETWORKS**

<i>David B. Johnson and David A. Maltz</i>	153
1 Introduction	153
2 Assumptions	157
3 Basic Operation	157
4 Optimizations	162
5 Performance Evaluation	169
6 Related Work	174
7 Conclusion	178
REFERENCES	179

6 ROUTING OVER MULTI-HOP WIRELESS NETWORK OF MOBILE COMPUTERS

<i>Charles E. Perkins and Pravin Bhagwat</i>	183
1 Introduction	184
2 Overview of Routing Methods	185
3 Destination-Sequenced Distance Vector (DSDV) Protocol	187
4 Examples of DSDV in Operation	193
5 Properties of the DSDV Protocol	199
6 Comparison with Other Methods	200
7 Future Work	202
8 Summary	203
REFERENCES	205

7 IMPROVING THE PERFORMANCE OF RELIABLE TRANSPORT PROTOCOLS IN MOBILE COMPUTING ENVIRONMENTS

<i>Ramón Cáceres and Liviu Iftode</i>	207
1 Introduction	208
2 Wireless Networking Testbed	209
3 The Effects of Motion	212
4 Improving Performance	218
5 Wireless Transmission Errors	225
6 Conclusions	227
REFERENCES	228

8 INDIRECT TRANSPORT LAYER PROTOCOLS FOR MOBILE WIRELESS ENVIRONMENT

<i>Ajay V. Bakre and B.R. Badrinath</i>	229
1 Introduction	229
2 System Model	230
3 Indirect Transport Layer	233
4 Implementation and Handoffs	238
5 Performance Results	240
6 Alternatives to Indirect Protocols	247
7 Conclusion and Future Work	249

REFERENCES	251
9 CONNECTING MOBILE WORKSTATIONS TO THE INTERNET OVER A DIGITAL CELLULAR TELEPHONE NETWORK	
<i>Markku Kojo, Kimmo Raatikainen and Timo Alanko</i>	253
1 Introduction	253
2 Mobile Nodes and TCP/IP Protocols	256
3 The Mowgli Communication Architecture	259
4 Enhanced Functionality for Mobility	265
5 Discussion	268
REFERENCES	269
10 ASYNCHRONOUS VIDEO: COORDINATED VIDEO CODING AND TRANSPORT FOR HETEROGENEOUS NETWORKS WITH WIRELESS ACCESS	
<i>Johnathan M. Reason, Louis C. Yun, Allen Y. Lao, and David G. Messerschmitt</i>	271
1 Introduction	272
2 MPEG and Mobile Channels	273
3 System Level Considerations	276
4 QOS and Traffic Capacity	281
5 Video Coding for a Substream Transport	286
6 Results	293
7 Conclusions and Future Work	296
REFERENCES	297
11 WIRELESS PUBLISHING: ISSUES AND SOLUTIONS	
<i>T. Imielinski and S. Viswanathan</i>	299
1 Introduction	299
2 Publishing Mode	301
3 Publishing Using Temporal Addresses	308
4 Publishing Using Multicast Addresses	318
5 Adaptive Publishing	321

6	Other Information Delivery Methods	325
7	Conclusions and Implementation Status	327
	REFERENCES	328
12	BROADCAST DISKS: DATA MANAGEMENT FOR ASYMMETRIC COMMUNICATION ENVIRONMENTS	
	<i>S. Acharya, R. Alonso, M. Franklin, and S. Zdonik</i>	331
1	Introduction	332
2	Structuring the Broadcast Disk	335
3	Client Cache Management	341
4	Modeling the Broadcast Environment	343
5	Experiments and Results	346
6	Previous Work	356
7	Summary and Future Work	358
	REFERENCES	360
13	APPLICATION DESIGN FOR WIRELESS COMPUTING	
	<i>Terri Watson</i>	363
1	Introduction	363
2	Application Design for a Wireless Environment	365
3	Wireless Platform	366
4	W* : A Wireless Application	367
5	Experiences	369
6	Conclusions	371
	REFERENCES	371
14	MOBISAIC: AN INFORMATION SYSTEM FOR A MOBILE WIRELESS COMPUTING ENVIRONMENT	
	<i>Geoffrey M. Voelker and Brian N. Bershad</i>	375
1	Introduction	375
2	System Overview	378
3	Using Dynamic URLs	380
4	Active Documents	382

5	Mobisaic on the Desktop	386
6	Implementation	386
7	Future Work	389
8	Conclusions	393
	REFERENCES	394
15	PROVIDING LOCATION INFORMATION IN A UBIQUITOUS COMPUTING ENVIRONMENT	
	<i>Mike Spreitzer and Marvin Theimer</i>	397
1	Introduction	398
2	Architecture	400
3	Design Considerations	407
4	Status and Experience	413
5	Conclusions	419
	REFERENCES	423
16	UNIX FOR NOMADS: MAKING UNIX SUPPORT MOBILE COMPUTING	
	<i>Michael Bender, Alexander Davidson, Clark Dong, Steven Drach, Anthony Glenning, Karl Jacob, Jack Jia, James Kempf, Nachiappan Periakaruppan, Gale Snow, and Becky Wong</i>	425
1	Introduction	426
2	The Power Management Framework	427
3	System State Checkpoint and Resume	432
4	PCMCIA on Unix	435
5	Serial Wide-Area Connectivity and Link Management	440
6	Nomadic Electronic Mail	443
7	Summary	446
	REFERENCES	447
17	SCHEDULING FOR REDUCED CPU ENERGY	
	<i>Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker</i>	449
1	Introduction	450
2	An Energy Metric for CPUs	450

3	Approach of This Paper	452
4	Trace Data	452
5	Assumptions of the Simulations	453
6	Scheduling Algorithms	455
7	Evaluating the Algorithms	457
8	Discussion and Future Work	462
9	Conclusions	467
	REFERENCES	469
	APPENDIX A	469
	A.1 Description of Trace Data	470
18	STORAGE ALTERNATIVES FOR MOBILE COMPUTERS	
	<i>Fred Douglis, Ramón Cáceres, M. Frans Kaashoek, P. Krishnan, Kai Li, Brian Marsh, Joshua Tauber</i>	
1	Introduction	474
2	Architectural Alternatives	476
3	Hardware Measurements	477
4	Trace-Driven Simulation	480
5	Results	486
6	Related Work	498
7	Conclusions	500
8	Differences from the Preceding Version	503
19	DISCONNECTED OPERATION IN THE CODA FILE SYSTEM	
	<i>James J. Kistler and M. Satyanarayanan</i>	
1	Introduction	507
2	Design Overview	508
3	Design Rationale	511
4	Detailed Design and Implementation	515
5	Status and Evaluation	525
6	Related Work	530
7	Future Work	531
8	Conclusions	532
	REFERENCES	533

20 EXPERIENCE WITH DISCONNECTED OPERATION IN A MOBILE COMPUTING ENVIRONMENT*M. Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria*

<i>R. Ebling, Puneet Kumar, and Qi Lu</i>	537
1 Introduction	537
2 Constraints of Mobile Computing	538
3 Overview of Coda File System	539
4 Implementation Status	541
5 Qualitative Evaluation	542
6 Quantitative Evaluation	554
7 Work in Progress	563
8 Conclusions	568
REFERENCES	569

21 MOBILITY SUPPORT FOR SALES AND INVENTORY APPLICATIONS

<i>Narayanan Krishnakumar and Ravi Jain</i>	571
1 Introduction	572
2 Application Scenario: Mobile Sales and Inventory	573
3 System Architecture	574
4 Database System Design	577
5 Mobile Sales Transactions	583
6 Maintaining Service Profiles	588
7 Conclusions	592
REFERENCES	592

22 STRATEGIES FOR QUERY PROCESSING IN MOBILE COMPUTING

<i>Masahiko Tsukamoto, Rieko Kadobayashi and Shojiro Nishio</i>	595
1 Introduction	595
2 Techniques Used in Mobile Communication Protocols	596
3 Query Processing for Location Sensitive Queries	600
4 Evaluation	608
5 Conclusions	617
REFERENCES	618

23 THE CASE FOR WIRELESS OVERLAY NETWORKS

<i>Randy H. Katz and Eric A. Brewer</i>	621
1 Introduction	621
2 Applications Enabled by Wireless Overlays	623
3 Applications Viewpoint	625
4 Gateway-Centric Network Management	628
5 Overlay Network Management	635
6 Applications Support Services	640
7 Related Work	644
8 Summary and Conclusions	648
REFERENCES	648

24 THE DIANA APPROACH TO MOBILE COMPUTING

<i>Arthur M. Keller, Tahir Ahmad, Mike Clary, Owen Densmore, Steve Gadol, Wei Huang, Behfar Razavi, and Robert Pang</i>	651
1 Introduction	651
2 DIANA—The Overall Architecture	656
3 User Interface and Display Independence	660
4 The DIANA Network Architecture	665
5 Application Development Methodology	670
6 Current Implementation	674
7 Future Issues	674
8 Concluding Remarks	677
REFERENCES	677

25 THE CMU MOBILE COMPUTERS AND THEIR APPLICATION FOR MAINTENANCE

<i>Asim Smailagic and Daniel P. Siewiorek</i>	681
1 Introduction	681
2 CMU Mobile Computers and Their Applications	683
3 VuMan as a Maintenance Assistant	684
4 Application Software	688
5 Conclusions	689

6	Figures	691
	REFERENCES	698
26	GENESIS AND ADVANCED TRAVELER INFORMATION SYSTEMS	
	<i>Shashi Shekhar and Duen-Ren Liu</i>	699
1	Introduction	699
2	Genesis System	704
3	Data Management in Genesis	708
4	Performance Issues in Genesis	715
5	Conclusions	720
	REFERENCES	721
	INDEX	725

CONTRIBUTORS

Norman Adams

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Tahir Ahmad

Stanford University
Computer Science Department
Stanford, California 94303

Timo Alanko

Department of Computer Science
University of Helsinki
P.O.Box 26 FIN
00014 University of Helsinki, Finland

Ajay Bakre

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

B.R.Badrinath

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

Michael Bender

Sun Microsystems, Inc.
Mountain View, California.

Brian N. Bershad

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Pravin Bhagwat

University of Maryland
College Park, MD

Eric Brewer

Computer Science Division,
Department of Electrical Engineering and
Computer Sciences
University of California
Berkeley, CA 94720-1776

Ramón Cáceres

AT&T Bell Laboratories
101 Crawfords Corner Road
Holmdel NJ 07733

Mike Clary

Sun Microsystem, Inc.
Mountain View, California

Alexander Davidson

Sun Microsystems, Inc.
Mountain View, California

Alan Demers

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Owen Densmore

Sun Microsystems, Inc.
Mountain View, California

Clark Dong

Sun Microsystems, Inc.
Mountain View, California.

Steven Drach

Sun Microsystems, Inc.
Mountain View, California.

Fred Douglis

AT&T Bell Laboratories
600 Mountain Ave., Room 2B-105
Murray Hill, NJ 07974

Steve Gadol

Sun Microsystems,
Mountain View, California

Anthony Glenning

Sun Microsystems, Inc.
Mountain View, California.

Rich Gold

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

David Goldberg

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Maria R. Ebling

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

John R. Ellis

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Wei Huang

Stanford University,
Computer Science Department
Stanford, California 94303

Tomasz Imielinski

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

Liviu Iftode

Department of Computer Science
Princeton University
Princeont, NJ 08554

Karl Jacob

Sun Microsystems, Inc.
Mountain View, California.

Jack Jia

Sun Microsystems, Inc.
Mountain View, California.

Ravi Jain

Bell Communication Research
445 South Street
Morristown, NJ 07960

David B. Johnson

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

M. Frans Kaashoek

Massachusetts Institute of Technology
Cambridge, MA

Rieko Kadobayashi

ATR Media Integration
Communications Research Laboratories
Seika-cho, Soraku-gun
Kyoto 619-02, Japan

Randy Katz

Computer Science Division
Department of Electrical Engineering and
Computer Sciences
University of California
Berkeley, CA 94720-1776

Arthur M. Keller

Stanford University,
Computer Science Department
Stanford, California 94303

James Kempf

Sun Microsystems, Inc.
Mountain View, California.

J.Kistler

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

Markku Kojo

Department of Computer Science
University of Helsinki
P.O.Box 26 FIN
00014 University of Helsinki, Finland

Hank Korth

AT&T Bell Laboratories
Murray Hill, NJ 07974

Narayanan Krishnakumar

Bell Communication Research
445 South Street
Morristown, NJ 07960

P. Krishnan

AT&T Bell Laboratories,
Holmdel, NJ 07733

Puneet Kumar

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

Allen Y. Lao

Computer Science Division
Department of Electrical Engineering and
Computer Sciences
University of California
Berkeley, CA 94720-1776

Kai Li

Computer Science Department,
Princeton University
Princeton, NJ 08544

Duen-Ren Liu

Institute of Information Management
National Chiao Tung University, Hsinchu
Taiwan, Republic of China

Qi Lu

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

David Maltz

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

Brian Marsh

D.E. Shaw & Co.
New York, NY

David G. Messerschmitt

Computer Science Division
Department of Electrical Engineering and
Computer Sciences
University of California
Berkeley, CA 94720-1776

Lilly B. Mumment

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

Shojiro Nishio

Dept. of Information Systems Engineering,
Faculty of Engineering, Osaka University
2-1 Yamadaoka, Suita, Osaka 565, Japan

Robert Pang

Stanford University
Computer Science Department
Stanford, California 94303

Nachiappan Periakaruppan

Sun Microsystems, Inc.
Mountain View, California.

Charles E.Perkins

IBM Watson Research Center
Yorktown Heights, NY

Karin Petersen

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Kimmo Raatikainen

Department of Computer Science
University of Helsinki
P.O.Box 26 FIN
00014 University of Helsinki, Finland

Behfar Razavi

Sun Microsystems,
Mountain View, California

Johnathan M.Reason

Computer Science Division, Department
of Electrical
Engineering and Computer Sciences, Uni-
versity of California
Berkeley, CA 94720-1776

M. Satyanarayanan

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

Bill N Schlit

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Shashi Shekhar

Department of Computer Science
University of Minnesota, Twin Cities,
Minneapolis, Minnesota

Scott Shenker

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Daniel P. Siewiorek

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

Asim Smailagic

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

Gale Snow

Sun Microsystems, Inc.
Mountain View, California.

Mike Spreitzer

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Joshua Tauber

Massachusetts Institute of Technology
Cambridge, MA

Marvin Theirmer

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Masahiko Tsukamoto

Dept. of Information Systems Engineering,
Faculty of Engineering, Osaka University
2-1 Yamadaoka, Suita, Osaka 565, Japan

Geoffrey M. Voelker

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

S. Viswanathan

Bell Communication Research,
445 South Street,
Morristown, NJ 07960

Roy Want

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Terri Watson

Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195

Mark Weiser

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Brent Welch

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California, 94301

Becky Wong

Sun Microsystems, Inc.
Mountain View, California.

Louis C. Yun

Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California
Berkeley, CA 94720-1776

PREFACE

The rapid development of wireless digital communication technology has created capabilities that software systems are only beginning to exploit. The falling cost of both communication and of mobile computing devices (laptop computers, hand-held computers, etc.) is making wireless computing affordable not only to business users but also to consumers.

Mobile computing is not a "scaled-down" version of the established and well-studied field of distributed computing. The nature of wireless communication media and the mobility of computers combine to create fundamentally new problems in networking, operating systems, and information systems. Furthermore, many of the applications envisioned for mobile computing place novel demands on software systems.

Although mobile computing is still in its infancy, some basic concepts have been identified and several seminal experimental systems developed. This book includes a set of contributed papers that describe these concepts and systems. Other papers describe applications that are currently being deployed and tested. The first chapter offers an introduction to the field of mobile computing, a survey of technical issues, and a summary of the papers that comprise subsequent chapters. We have chosen to reprint several key papers that appeared previously in conference proceedings. Many of the papers in this book are being published here for the first time. Of these new papers, some are expanded versions of papers first presented at the NSF-sponsored Mobidata Workshop on Mobile and Wireless Information Systems, held at Rutgers University on Oct 31 and Nov 1, 1994.

Many people and organizations assisted us in this project. Ron Ashany and Maria Zemankova of NSF helped provide support for the workshop that spawned this project. Bob Holland of Kluwer Academic Publishers has assisted in seeing the effort through to completion. We conducted a formal refereeing process for original papers in this book and would like to thank those referees, though they remain anonymous. S. Viswanathan assisted us extensively with text format-

ting. Above all, we thank the authors who have contributed their papers to this collection.

Hank would like to thank his wife, Joan, and children Abby and Joe for their understanding and patience. Tomasz would like to thank his wife Celina and sons, Marcin and Konrad, for their support and encouragement.

INTRODUCTION TO MOBILE COMPUTING

Tomasz Imielinski* and Henry F. Korth**

* Rutgers University, New Brunswick, NJ 08903

** AT&T Bell Laboratories, Murray Hill, NJ 07974-0636

1 INTRODUCTION

The rapidly expanding technology of cellular communication, wireless LANs, and satellite services will make information accessible anywhere and at any time. In the near future, tens of millions of people will carry a portable palmtop or laptop computer. Smaller units, often called personal digital assistants or personal communicators, will run on AA batteries and may have only a small memory; larger ones will be powerful laptop computers with large memories and powerful processors. Regardless of size, most mobile computers will be equipped with a wireless connection to the fixed part of the network, and, perhaps, to other mobile computers. The resulting computing environment, which is often referred to as *mobile* or *nomadic* computing, no longer requires users to maintain a fixed and universally known position in the network and enables almost unrestricted mobility. Mobility and portability will create an entire new class of applications and, possibly, new massive markets combining personal computing and consumer electronics.

Not only will information be easily accessible from virtually any place and time, but also, it will be stored in a highly decentralized, distributed information infrastructure often termed the “information superhighway.” A wide variety of information servers (both public and proprietary) will be accessible to mobile computers. We are already seeing the beginnings of this with the rapidly growing popularity of the World-Wide Web across a broad range of computer users. As the mobile infrastructure develops, it will become what is referred to as the “first wireless mile” or “wireless on-ramp” for the information superhighway. In some applications, mobile computers themselves may contain data, or data may be stored on flash-memory “smart cards.”

This book presents a snapshot of the new, exciting, and rapidly developing field of mobile computing. The focus of this book is on software issues of mobile computing rather than hardware. Although hardware in this area is advancing rapidly, and the capabilities of hardware spurred much of the initial excitement regarding mobility, it is increasingly clear that it is developments in the software domain that will bring the power of mobile hardware to a wide group of potential users. Emerging developments in software are leading to practical, easy-to-use applications.

The papers in this book address areas of interest both to software practitioners and researchers working in the fields of networking, operating systems, and databases. Topics include network and communication issues, location awareness (both at the system level and the application level), and application software. Since the infrastructure to support mobile computing includes not only the mobile computers themselves, but also the stationary computers (base stations) that support mobility, many of the papers address client/server, network control, and distributed information management issues. In other words, the book considers anything above and including the network level of the OSI hierarchy. Furthermore, the book includes several chapters in which interesting prototype mobile systems are reviewed.

The general, abstract view of a mobile system consists of mobile hosts (MHs) interacting with the fixed network via mobile support stations (MSSs). The connection between the MH and MSS is via a wireless link. Each MSS is in charge of a *cell*. Cells can have sizes ranging from picocells of approximately one hundred meters in diameter to macro cells and perhaps even global satellite cells. The capabilities of mobile hosts will vary from “dumb” terminals to complex “walkstations” which essentially have the capabilities of desk-top computers. We expect that users of mobile computers will want to run a variety of applications, depending on the physical size and power of their machine. These range from standard desk-top applications like word processors, spreadsheets, and electronic mail, to remote information access via database system applications or Web browsers. Additionally, there may be location-dependent applications specific to the particular location of the user at a particular time.

In this chapter, we review mobile computing technology, research issues, and prototypes. We explain the need for new primitives (in “middleware”) in order to help develop mobile computing software. The need for such primitives is a consequence of new features of the mobile computing environment, namely mobility itself, narrow and varying bandwidth, and limitations in battery power. As will be demonstrated later in this chapter and elsewhere in the book, mobility will not only affect the network layer (the physical address of a mobile

host is no longer fixed) but also higher layers, especially for location-dependent applications. Bandwidth and energy limitations will require solutions in which several sites on the network will cooperate with the mobile unit.

Section 2 introduces wireless technology with its two main resource restrictions: bandwidth and battery power. Next, Section 3 discusses the research challenges in mobile computing. Finally, Section 4 provides a short overview of the book's content.

2 TECHNOLOGY OVERVIEW

In this section, we discuss features of radio communication that are important from the standpoint of software development: both for the mobile unit for the network infrastructure.

2.1 What is Special about Radio-based Computer Networks?

Perhaps the most important distinction between the wired and wireless environment is that there is only one “ether” - one global bandwidth which is to be shared by all users. Consequently, the bandwidth limitations are much more severe and long-lived for wireless environments, as compared to those that are wired. Fortunately, the same channel (frequency, time slot) can be re-used if the points of re-use are physically sufficiently far apart for the signals not to interfere. As a result, the overall architecture of the wireless system is based on the concept of a massively distributed system divided into physical *cells*. These cells are, in effect, sub-networks. A mobile unit has to interact with many cells, cross their boundaries, etc. In so doing, the mobile unit moves to a new sub-network and changes its own physical network address. This concept of replication and re-use has a profound impact on software systems. Software should scale well with the inherently distributed infrastructure. It must allow interoperability and seamless transition among cells. It is the software system architecture which is affected most critically by the use of wireless communication. Technology trends point towards smaller cell sizes than in current cellular telephony systems. Goodman [Goo91] discusses the architectural consequences of frequency re-use, which leads to cells of smaller size, called *picocells*. Picocells use transmission power levels two orders of magnitude lower than current cells. This increased number of cells increases the number of times that a mobile unit

crosses cell boundaries, thus significantly increasing the demand on the network control functions. This trend toward smaller cell sizes further emphasizes the need for a distributed control model.

Fading, Noise, and Interference

Radio communication with a mobile computer is highly variable and hard to control. The phenomenon of *fading*, signal strength suddenly decreasing, is a significant problem (and one familiar to anyone who has listened to an automobile radio).

It is quite normal for a mobile radio to experience fades of 40–50 dB [Cal88] in fraction of a second. A fast moving automobile in an urban environment may be subjected to dozens of significant fades (20dB, or more) per second. For comparison, a representative signal-to-noise ratio (SNR) objective [Cal88] for an analog wireline link is about 46 dB. Signal fluctuations in a wireline link are no more than 1–2 dB for a short-haul circuit and short time-span, and perhaps 10–15 dB for gradual degradation due to wear and tear of the network.

During a fade, the signal may be so weak that an undesired, interfering signal from a neighboring cell may dominate and, consequently, the receiver may lock on that undesired signal. There are, in general, two types of fading:

1. **Short-term multipath fading, or Rayleigh fading**, which is due to the same signal taking different paths and arriving at the receiver shifted in phase.
2. **Long-term fading, or “radio shadows,”** which is caused by the topography of the terrain (like mountains) and can lead to signal dropouts.

The problem of short-term fading is addressed by exploiting antenna diversity. In this way, two or more inputs to the mobile terminal are provided so that fading effects are uncorrelated. Long-term fading is dealt with by deploying multiple antenna sites.

Noise and interference have a significant negative effect on the bit-error rate (BER). Typically, the BER in a mobile radio runs up to six orders of magnitude higher than in non-mobile point-to-point radio.

The high bit-error rate of the wireless environment eventually translates into a lower bit rate for network throughput.¹ Since mobility of the terminal contributes to increased fading and consequently, to increased error rate, we may expect that the bit rate between a mobile terminal and a base station will drop with speed of the terminal's movement.

Radio Transmission Power

In wireless environments, management of transmission power is critical. The importance of power arises from two factors:

1. Energy is a scarce resource in mobile units. Therefore, transmission with power as low as feasible is highly desirable. We shall discuss energy issues in more detail later in this chapter.
2. Signals from other terminals are considered by a given terminal as interference, which has a negative effect on the signal-to-noise ratio. Thus, it is important that each terminal use the "right" amount of power to transmit successfully without unnecessary degradation of others' transmissions.

Let us review the basic relationships among power, distance and signal-to-noise ratio. In the ideal free space, signal strength diminishes with the *inverse square* of the distance. For example, if the received signal is 100 watts at a distance of 1 mile, it will be 25 watts at 2 miles, 4 watts at 5 miles and 1 watt at 10 miles. In practice, since mobile terminals do not move in free space but rather have to deal with various obstacles, the loss of power is much more significant, ranging from inverse cube of distance to exponents as high as the 6th power. For example, with inverse of the 6th power law in effect, the 100 watt signal level at the first mile would degrade to only 1.5 watts at 2 miles.

The degree of attenuation also depends critically on frequency – the higher the frequency, the more attenuation and consequently, smaller range.² For example, for 3–5 GHz, omnidirectional transmission leads to impractically short distances. For these reasons, directional antennae are used so as to concentrate the energy along a fixed point-to-point path. Such systems are typically point-to-point between stations 10–50 miles apart.

¹Wired networks drop packets due to congestion, while wireless networks lose packets mainly due to the error rate. Thus, quoting Mark Weiser, in a wired network, the transmitter has to back off when facing packet loss, while in a wireless network it should "try harder."

²But the antenna size is smaller as well.

Parameters at the Application-Level

Let us consider the issue of the parameters of the mobile computing environment to which the application should have access. Although it is desirable to shield applications from the low-level details of the wireless network, the application-level system software may be able to use information about low-level conditions in its choice of strategy. Applications running on the top of radio communication channels have to be adaptive to the changing channel conditions.

A strong case can be made that BER and signal strength (SNR) are physical parameters which should be “accessible” to higher level protocols (such as the transport level and above). BER data would help the transport layer protocol to decide if the drop in the channel reliability is due to congestion or due to the error rate. SNR data can help an application to decide if the mobile unit should attempt to defer a transmission to a time when channel conditions are more favorable. Only with application-level knowledge can it be decided whether the user is best served by delaying transmission or by potentially costly attempts to transmit under adverse conditions.

There are other relevant parameters of the mobile computing environment to consider materializing at the application level: the *location* of the mobile unit and the *tariff*. Location of the mobile unit (either as a cell where the unit currently resides or, perhaps, exact longitude and latitude determined by a global positioning system (GPS)) is an important parameter for some *location-aware* applications such as local yellow pages, route information services, etc. Tariffs for wireless communication are still in a state of flux, but clearly the cost of application activity can be affected seriously by the type of tariff in effect in the current environment. For example, channel access can be charged on a per-packet basis, a connection-time basis, etc. The charges themselves may also depend on the channel quality, which in turn may depend on the mobile unit’s location or the time of day. The application, in order to minimize overall charges, may adjust its behavior to the tariff conditions. Additionally, if the mobile unit is equipped with multiple interfaces, the application can decide to switch to the one which is most suitable for the current situation. For example, a unit equipped with a cellular modem as well as a CDPD modem may use the CDPD modem for short and bursty electronic mail transmission and the regular cellular modem for long, bulk data transfers (for example, ftp).³

³CDPD is expected to charge per packet, while cellular charges are usually per minute of connection.

2.2 Network Technology

In this section, we review the existing wireless communication technology.

Analog Cellular Systems

The cellular system was pioneered at AT&T Bell Laboratories in early 1970s and was deployed in U. S. in the early 1980s. The coverage area is subdivided into cells with sizes varying from a few kilometers in diameter to between 50 and 100 km in earlier cellular systems. The first generation, called AMPS (Advanced Mobile Phone Service), which is still used for cellular telephony, uses analog frequency modulation for speech transmission. Individual calls use different frequencies, using a system referred to as frequency division / multiple access (FDMA).

In the AMPS system, 50 MHz of bandwidth in the bands of 824–849 MHz and 869–894 MHz are allocated to cellular mobile radio. In each geographic region, there are two carriers (“A carrier” and “B carrier”), which control 25 MHz each. This spectrum is subdivided into 832 channels, each of 30 KHz. To avoid interference, neighboring cells use different channels. Typically, a cell has 6 neighbors and, therefore, the cell may use 1/7 of the allotted 832 channels. This is called a *7-group re-use cluster*. Analogously, in the 12 group frequency plan each cell uses only 1/12 of the total bandwidth and its bandwidth allocation is disjoint with that of 11 neighboring cells.

The Cellular Digital Packet Data (CDPD) is designed to provide packet data services on the top of the existing AMPS. The CDPD system provides data services without interfering with the existing analog and digital telephone services that use the same AMPS channels. In CDPD, voice transmission has a higher priority than data transmission. Data can be transmitted on a channel only if there is no voice call which is using that channel. Voice transmission can preempt data transmission. In such cases, another channel must be found for the data transmission to continue. The base station, called MDBS, makes the determination about the next available channel either by a forward power monitor (sniffer), which measures the signal strength on the uplink channel (from the mobile terminal to the base station) or by using a deterministic algorithm for channel assignment (unplanned versus planned hop). A given data transmission may thus possibly spread over several physical channels, especially when the voice traffic is intense. The maximum supported bit rate is 19.2 Kb/s.

CDPD is currently being deployed in a number of regions in the country. It re-uses the existing infrastructure (base stations) and avoids the costly initial investment. This is a major advantage over the proposed Personal Communication System (PCS), which we discuss below. Although CDPD is viewed as mainly a transitional technology (before PCS is fully deployed), it may actually stay longer than originally expected.

Digital Cellular Systems

Digital cellular systems are sometimes referred to as *second-generation cellular*. Due to the development of the digital speech coding techniques, digital systems are gaining rapidly in significance. Among the advantages of digital cellular communications are the following [Cal88]:

- Robustness of digital networks: Resistance to noise and crosstalk. Efficient error correction by treating all errors uniformly
- Intelligence of the digital network
- Flexibility and integration with the wired digital networks
- Reduced RF transmission power (increasing battery life in handsets)
- Encryption for communication privacy
- Reduced system complexity
- Higher user capacity

Two basic techniques for managing shared access are competing in digital cellular radio: TDMA (Time Division Multiple Access) and CDMA (Code Division Multiple Access).

With TDMA, only a single user can use a given frequency at any time. With CDMA (spread spectrum), a frequency-based channel is used simultaneously by multiple mobile units in a given cell. Spread-spectrum techniques are very effective in dealing with two basic problems facing cellular communication: multipath fading and the interference from other users. This is due to the frequency diversity introduced by the wide bandwidth, and results in potentially higher cell capacity as compared to other, non-spread access methods [KMM95]. Another consideration for using CDMA in the cellular system is its attractive re-use factor. For non-spread techniques such as TDMA and FDMA,

the same frequency cannot typically be re-used in adjacent cells due to the possibility of interference. With spread-spectrum signaling, the possibility of frequency re-use in adjacent cells exists. As a result, CDMA is anticipated to have larger capacity in a multi-cell system than either FDMA or TDMA.⁴ Additionally, CDMA provides a natural way to explore the bursty nature of sources. For example, for a two-way telephone conversation, the voice activity of each participant is about 50% of the time. CDMA can take advantage of periods of inactivity, effectively doubling the number of simultaneous conversations in the system [KMM95]. Thus, spread spectrum promises both higher capacity and a simplified frequency management scheme.

We distinguish between Direct Sequence Cellular CDMA (DS/CDMA) and Frequency Hopping Cellular CDMA (FH/CDMA). In DS/CDMA, the spread-spectrum signals are generated by linear modulation with sequences that are assigned to individual users as their signature codes. The sequences are typically orthogonal and allow the receivers to demodulate the spread-spectrum signal in such a way that the signal with matching signature is recovered, while other signals are suppressed. The motivation behind FH/CDMA is the same as DS/CDMA – the signal is spread to achieve frequency diversity. The spreading of the signal is obtained by choosing a frequency-hopping sequence for transmission. If the frequency-hopping sequences are orthogonal, the users within a cell do not interfere with each other. The main difference between the two schemes is that in DS/CDMA, the signal requires a wide and contiguous frequency band while in FH/ CDMA, the spectrum does not need to be contiguous. In particular, it allows the implementation of FH/CDMA for private land mobile operations, where licenses are given on the basis of isolated narrowband channels.

There are a number of basic standards and deployed systems in Europe, the U. S. and Japan. Below we summarize some of them.

- **The Pan-European Global System for Mobile Communications (GSM).** GSM is based on TDMA with eight slots per radio channel. Each user transmits periodically in each of the eight slots with a duration of 0.57 seconds. In the present version, GSM supports full-rate 22.8 Kb/s transmission. In 1993 and 1994, GSM experienced tremendous growth in Europe with over 2 million subscribers.

⁴However, in an *isolated* cell, due to the orthogonality of the CDMA waveforms, the capacity of the cell is less than it would be with the non-spread techniques

- **The IS-54 standard in the U. S., based on TDMA.** IS-54 is a North American standard which is based on TDMA. IS-54 retains the 30 KHz spacing of AMPS to make the evolution from analog to digital easier. Each frequency channel provides a raw bit rate of 48.6 Kb/s. There are six time slots, two of which are assigned to each user. Thus, each 30 KHz channel can serve three users simultaneously with the same re-use patterns as AMPS. Consequently, IS-54 provides three-fold growth of the capacity compared to AMPS. The IS-54 standard is “dual mode” (analog and digital) and can operate in the same spectrum as AMPS. IS-54 is designed mainly for low-data-rate communication[Kat94]. A conventional analog cellular modem uses the whole channel while achieving 2.4 Kb/s.
- **The IS-95 standard in the U. S, based on CDMA.** The IS-95 standard is based on spread-spectrum CDMA. With IS-95, many users share the same channel for transmission - there is no longer a need for frequency re-use, since each cell can use the entire bandwidth. CDMA requires the use of sophisticated power-control schemes to allow the mobile units which are close to the base station to reduce the transmission power levels. Power control is necessary in order to avoid the so called “near-far” terminal problem, where a strong signal received from a nearby terminal may suppress a weaker signal from a terminal which is far away. The IS-95 CDMA based standard offers a number of benefits including better cell capacity, elimination of the need for planning frequency assignments to cells, better efficiency for voice due to voice activity detection (and channel use in periods when there is no speech activity). Additionally, CDMA power control techniques contribute positively to the reduction of the overall RF transmission power as well as allowing “soft handoff,” which allows a “gradual” transition. The “gradual” transition is due to the gradual deterioration of the signal strength as opposed to more rapid changes in other methods.

The U. S. Federal Communication Commission opened three bands for spread spectrum users: 902–928 MHz, 2,400–2,483.5 MHz and 5,725–5,850 MHz.

Cordless Telephony

There are currently an estimated 60 million cordless telephones in the U. S. with total sales reaching 15 million units per year [PGH95]. Cordless technology requires only very low power of transmission, but it is limited to very short ranges. The main digital cordless standards include CT2 and DECT (Digital European Cordless Telecommunications).

The CT2 spectrum allocation consists of 40 FDMA channels with 100 KHz spacing in the range of 864–868 MHz. The maximum transmitting power is 10mW, which is very small. CT2 supports the transmission of data from 2.4 Kb/s to 32 Kb/s.

The functionality of DECT system is closer to that of a cellular system than it is to that of a classical cordless telephone system. DECT uses TDMA with 12 slots per carrier in each direction. Furthermore, it can allocate several slots to a single call to increase the data rate. DECT systems are only beginning to be shipped and the unit sales reached 100,000 last year. It is a potential basis for a future, low cost, picocell-based system.

Wireless Local Area Networks

The work on radio-based computer networks goes back to the early seventies and the work on Aloha network at the University of Hawaii [Tan81]. The existing wireless LAN technology can be traced back to these early efforts.

Wireless LANs are providing data rates typically two orders of magnitude higher than outdoor radio. Thus, data rates exceeding 1 Mb/s are quite common. FreePort provides wireless Ethernet (IEEE 802.3) and operates in the 2400–2483.5 MHz (receiver) and 5725–5850 MHz (transmit) frequencies, using direct-sequence spread spectrum. WaveLAN provides peer-to-peer communication in 902–928 MHz (in the U. S.) and uses direct sequence spreading with the CSMA/CA (carrier sense, multiple access, with collision avoidance) protocol. Finally, Altair uses the Ethernet protocol (CSMA/CD, or carrier sense, multiple access, with collision detection) and operates in microwave spectrum near 18 GHz and requires a site-specific FCC license.⁵

Standards such as IEEE 802.11 (which we discuss in more detail in the context of power management features) are being developed in the U. S. and in Europe. In general, the goal is to provide data rates exceeding 1 Mb/s and also to support architectures with infrastructure (base stations) as well as “ad-hoc architectures,” where the terminals communicate directly with each other (peer to peer) without the mediation of a fixed base station [PGH95].

⁵FCC licenses are not required for WaveLAN and FreePort which operate in the ISM (Industrial, Scientific and Medical) part of the spectrum. These frequency bands were originally designated for operation of equipment which generates and uses RF energy locally for industrial, scientific and medical applications, excluding applications in the field of telecommunications. It was later suggested to use these frequencies for local telecommunication, such as on-site communication.

Karn [Kar91] points out deficiencies of the CSMA protocol in the context of the wireless medium. The *hidden terminal* problem arises when station Y can hear both stations X and Z but X and Z cannot hear one another. In such a case, X and Z will collide at Y since CSMA will allow both X and Z to transmit. The exposed terminal problem arises when station X can hear far away station Y even though X is too far from Y to interfere with its traffic to other nearby stations. In this case, CSMA is too conservative, it will stop X from transmitting, wasting an opportunity to re-use channel bandwidth locally. Karn proposes to eliminate carrier sensing, turning CSMA/CA into MACA (Multiple Access with Collision Avoidance). If station X wants to transmit to the station Y, it first sends a short request to send (RTS) packet and inhibits its transmitter long enough for the addressed station to respond with a CTS (Clear to send). Collisions are still possible for the RTS and CTS packets, but the hidden and exposed terminal problems are avoided. Different variants of this protocol have been implemented at Xerox and at Berkeley but there are still a number of interesting issues, such as power control which have to be resolved.

Private Wireless Data Networks and Metropolitan Area Wireless Networks

Several private wireless data networks provide service over wide areas using the licensed spectrum. For example ARDIS (Advanced Radio Data Information Service) offers service to over 400 metropolitan areas with the data rate ranging from 8 Kb/s to 19.2 Kb/s in some areas. RAM Mobile Data offers service over its MobiTex network providing coverage in 216 metropolitan areas with a data rate of 8 Kb/s. CDPD, which we described in more detail above, offers 19.2 Kb/s maximum data rate over analog cellular. Motorola's Embarc with its expanding set of devices (such as the Info receiver) provides satellite-based information services.

Recently formed Metricom provides an innovative metropolitan-area wireless network based on frequency-hopping spread-spectrum technology. This network, called Ricochet, is a mesh-like network consisting of shoebox-sized radios, that are mounted on existing pole-tops or buildings. It offers effective bit rates in the range of 10–40 Kb/s over 902–928 MHz frequency band.

Infrared Technology

Infrared technology offers an alternative to the standard radio frequency communication. In general, it is much cheaper to use, but it is restricted only to

very short distances and is subject to line-of-sight transmission limitations. An invisible infrared light beam, in the frequency spectrum of laser beams (in the terahertz range), is focused from a transmitter to the receiver over a very short distance.

Infrared transmission is subject to the following restrictions [Bat94]:

- Transmission distance of less than 2 miles
- Line of sight limitations
- Restricted to of 16 Mb/s throughput
- Proneness to environmental disturbances, such as fog, dust, and heavy rain

The advantages of the infrared technology are as follows:

- Reasonably high bandwidth
- No license required
- Cost effective
- Capable of traversing multiple paths without interference
- More secure than radio
- Immune to radio frequency interference and electromagnetic interference

Satellite Networks

Mobile satellite services allow complete global coverage. In these systems, satellites play the role of mobile base stations. Satellite-based systems may be categorized according to the orbital altitude:

- **Geostationary satellites (GEOS)**, at an altitude of 35,786 km
- **Low earth orbit satellites (LEOS)**, at the altitudes on the order of 1000 km
- **Medium earth orbit satellites**, with widely varying altitudes between those of GEOS and LEOS.

The major advantage of GEOS systems is that contiguous global coverage up to 75 degrees latitude can be provided with just 3 satellites. On the other hand, the main drawback is a large 240–270 ms round trip propagation delay and the higher RF power required. On the other hand, LEOS systems require less power but require frequent handoffs.⁶

The bit rates for the satellite communication systems are relatively low. For example, for Qualcomm's OmniTracks, which provides two-way communication as well as location positioning, the downlink data rate is between 5 Kb/s and 15 Kb/s and the uplink data rate is between 55 b/s and 165 b/s.⁷ [Kat94] Experimental satellite systems provide higher data rates, for example NASA's ACT satellite offers T1 data rates at 1.5 Mb/s.

The Hughes Direct Broadcast Satellite System provides coverage to most of North America. It is *highly* asymmetric, with a shared 12 MB/s satellite downlink, a wire-line Internet-gateway uplink, and latencies greater than 500 ms. Individual users may achieve up to 400 Kb/s on the downlink. The system is wireless, but not mobile; its 24-inch satellite dish, though rapid to deploy, will not support communication for mobile units.

Future PCS

Recently, the U. S. Federal Communication Commission (FCC) allocated bandwidth in the 2 GHz range for "personal communication services" (PCS). This bandwidth was allocated via auction, and earned the U. S. government almost 7 billion dollars. Winning bidders in markets projected to be lucrative (such as New York and Los Angeles) bid approximately 20 dollars *per person*. This amount is just for the right to use the spectrum – still higher costs must be incurred to build the requisite infrastructure.

Two basic categories: *high tier*, which supports macrocells, and *low tier*, which is optimized for low power and low complexity, will be supported. These two tiers roughly correspond to the digital cellular and digital cordless categories. At this point, the initial list of potential standards has been narrowed down from 16 to 7 but it is still unclear as to what the standards will be and when the initial infrastructure will be deployed.

⁶Note that handoffs occur here even when the terminal is not moving at all (since low-earth orbit satellites move relative to fixed terrestrial positions).

⁷Note the asymmetry in uplink and downlink data rates.

One of the main challenges is how to mix continuous, real-time data such as voice (and perhaps, in the future, video) with the alphanumeric data (such as electronic mail, which is less time-critical). Several protocols such as PRMA (Packet Reservation Multiple Access) [WCW91] have been proposed. These protocols are based on the concept of reservation of slots within frames. In PRMA, the channel bit stream is first organized into slots, such that each slot can carry one packet from the mobile terminal to the base station. The time slots are grouped into frames. Each slot within a frame is either available or reserved, on the basis of a feedback packet broadcast in the previous frame from the base station to all of the terminals. Terminals with new information to transmit contend for access to available slots. At the end of each slot, the base station broadcasts the feedback packets that report the results of transmission. A terminal that succeeds in sending a packet to the base station obtains the reservation for this slot in the subsequent frames. The base station, on failing to receive a packet in the reserved slot, informs all the terminals that the slot is once again available. PRMA has proved to be suitable for data and voice communication and demonstrated efficient use of spectrum for voice.

2.3 Palmtop and Laptop Technology

The palmtop and laptop technology is expanding so rapidly that there is little point in discussing specific products. We refer the reader to the web home pages of major players on the market such as Apple, Motorola, Hewlett-Packard, NEC, Toshiba, IBM, etc.

Instead, we concentrate on the major issues brought about by the miniaturization of computer terminals. These are: extension of battery life, user interface, and display issues.

Battery Life

Energy supply is the major bottleneck for mobile wireless computers. In a recent *USA Today* article, longer battery life was mentioned as the feature most desired by mobile users.⁸ Unfortunately, expected growth of battery life is very slow (only 20% in the next 10 years [CB92]). Thus, energy efficiency

⁸The issue of energy management is nevertheless controversial. Many sceptics claim that periodic recharging will be sufficient to make the battery limitations go away. Others point out that, for example, the mobile terminals used in cars will use car's energy sources, not their own batteries. We still believe that longer battery life will be an important feature driving the market.

is a necessary feature both at the level of hardware, and software. Hardware providers are offering energy-efficient systems that switch off the background light of the screen, power down the disk (or eliminate the disk completely in favor of flash memory) and offer CPUs with an energy-efficient *doze mode*. For example, the Hobbit chip consumes 5,000 times less energy while in doze mode than in the active mode (250 mW in the active mode as opposed to 50 μ W in doze mode). There is a growing pressure on software vendors to incorporate energy management features.

To illustrate the constraint of limited available power, consider a laptop computer with a CD-ROM and a display. The constant power dissipation in a CD-ROM (for disk spinning) is approximately 1.0 W and the power dissipation for the display is approximately 2.5 W. Assume that AA batteries are to be used as the power source. A typical AA cell is rated to give 800 mA-Hr at 1.2 V (0.96 W-Hr). Thus, the assumed power source will last for only 2.7 hours. To increase the longevity of the batteries, the CD-ROM and the display may have to be powered off most of the time.

Transmitting and receiving consumes power as well. In practice, the power required for transmitting grows as a fourth power of distance between the transmitter and the receiver. Powering the receiver can also drain batteries. For example, a WaveLAN card consumes 1.7 W with the receiver on and 3.4 W with the transmitter on. An active cellular phone consumes even more, 5.4 W [FZ94], while consuming only 0.3 W if in standby mode.

We believe that design of energy efficient software will be one of the main research challenges in mobile computing. Below, we summarize some of these efforts including CPU scheduling for low power, new communication protocols, and energy-aware application design.

Storage Technology

Flash memories constitute new, more energy efficient, storage alternatives to disks. Flash memories consume relatively little energy and provide low latency and high throughput for read accesses. However, their high cost, \$30–50/Mbyte, is still a limiting factor. Furthermore, flash memories require additional system support for erasure management since writing to flash memory requires prior erasing. Additionally, the number of times the same memory block can be rewritten is usually limited (on the order of 100,000 times). Chapter 18 provides a detailed discussion of storage alternatives for mobile computers.

User Interface and Display Issues

The two key issues in human-computer interaction are the desirability of replacing the keyboard with a pen-based interface and the challenge of dealing with a small-sized display.

Pen-based interfaces have been introduced in recent years with mixed acceptance. The mixed results so far are due in large part to problems with hand-writing recognition (shown by early experience with the Apple Newton MessagePad). As recognition-related technology improves, it is expected that pen-based interfaces will gain in acceptance.

Small display size is a serious problem, especially for users who would like to access remote information services, such as those provided on the World-Wide Web. The need to deal with varying display sizes suggests that applications be structured in a *display independent* manner. Some of these issues are discussed in Chapter 24.

3 RESEARCH ISSUES

In this section, we review the main research issues in mobile computing and summarize current work in the field.

As we pointed out in the previous section, the main research challenges of mobile computing are due to mobility, variable communication conditions, and energy limitations. We shall discuss how these challenges affect different layers of OSI hierarchy starting from the networking layer and including some of the discussion of the data link layer as well. Here, we make an important distinction between the mobile *communication* which involve physical and MAC layers, and *mobile computing*, which we broadly define as including mobile networking (data link, network, transport layers) as well as software applications such as wireless access to information resources, client-server interaction, etc. In this book, we do not discuss the layers belonging to mobile communications, but rather, restrict our attention to those belonging to mobile computing.

3.1 Mobile Networking

In this section, we review how the network, transport, and data-link layer protocols are affected in mobile and wireless environments.

Mobility Management - Network Layer

The physical location of a mobile unit no longer determines its network address. Thus, mobility poses a major challenge from the point of view of the network level of OSI. How does the network know where a given user is? How does the network route messages to mobile users?

These questions are currently being addressed in two different communities: the Internet community and the cellular-communication community. The work in the Internet community involves *mobile IP* [JMP95, PB94] aiming towards a standard which would extend IP in order to deal with mobile hosts. The work in the cellular-communication community is the effort on *location management* of cellular phone users [BI96]. The latter deals mainly with connection-oriented communication, since it is motivated by issues of call set-up in telephony. On the other hand, the mobile IP work assumes a connectionless, packet-switching scenario.

The main problem of mobility management is how to find an adequate tradeoff between *searching* and *informing*. Searching is performed by the system in case the address of the message recipient is not known or, at least, it is not known precisely. Informing is the activity of the mobile unit, which informs the system about its position. There are two extreme solutions: one in which mobile units never inform the system about their positions and another, where units always inform the system about their movements. The second scenario works very well for those units that receive messages frequently. In such cases, the overhead of searching large portions of the network (possibly, even the entire network) is avoided at the expense of the mobile unit informing the system upon each move. If, additionally, the unit does not move between cells often, the second strategy is a clearly a good choice. On the other hand, if the unit moves very often and does not receive many messages, it is probably better not to inform the system but, rather, to have the system perform a search when necessary. There is clearly a need for a tradeoff: when mobile unit occasionally informs the system (or specially designed *home agent*). The exact nature of the tradeoffs and compromises is described in [IB92].

Since the users are mobile, there is a possibility of the user receiving the same message twice (though in different cells) or not receiving a message at all while in transition between two cells. Multicasting to mobile clients presents its own challenges [AB94b]. The main issue in multicasting is how to guarantee “exactly once” or “at least once” delivery in an efficient manner. The MCAST protocol provided in [AB94b] offers a preliminary solution. An interesting problem in multicasting is how to maintain efficiently a *group view* – the set of MSSs which have, in their cells, at least one member of a given multicasting group.

Ad-hoc networking is the ultimate challenge for mobile networking. In ad-hoc networking, mobile terminals can form networks without participation of the fixed infrastructure. Such networks arise in rapid-deployment situations, like emergency services at a disaster site, or military operations in a remote area. They can also be employed in business situations such as meetings held in venues without network infrastructure. The structure of an ad-hoc network is highly dynamic. Routing tables may change frequently due to changing communication conditions and power levels. A given terminal can serve as a router between two other terminals, due to its current intermediate position at one instant of time and no longer be in that position a short time later.

Transport Layer - Effects of Mobility and Wireless Connection

The task of efficiently maintaining end-to-end flow and congestion control over a mix of wired and wireless links is a difficult task due to the different characteristics of the wired and wireless networks. Congestion is the main source of packet loss on wired links, because error rates are very low. The situation is reversed on the wireless link, where packet loss is caused mainly by high error rates. Consequently, wireless and wired links require different techniques to achieve reliability and flow control.

Cáceres and Iftode in [CI95] (included as Chapter 7) was the first to observe that the congestion control in TCP may cause incorrect behavior of TCP when dealing with mobile hosts in a wireless network. The lack of acknowledgments from moving hosts who are in the middle of handoff between cells will cause the transmitting host to back off (slow down) exponentially, and it may take some time for the sender to come back to the original transmission rate. This leads to an unnecessary drop in throughput resulting from TCP’s misinterpretation of the receiver’s move as network congestion. Similarly, higher error rates on the wireless link will be interpreted by TCP as congestion, again leading to an

unnecessary drop in the data rate. TCP was designed for wired networks with low error rates and not for unreliable wireless networks with hosts which may occasionally disconnect.

One possible solution to this problem involves extending TCP (or another transport protocol) to handle wireless as well as wired links. Such a solution would require eventual replacement of the old version of TCP by a new protocol. This may not be practical. In order to avoid TCP modification, Badrinath et al. [BB95a] (also discussed in Chapter 8) propose to split the TCP protocol into two parts: one between the sender and the MSS local to the receiver and another, which manages the connection between the mobile host and the MSS. The TCP which runs between the sender and the MSS need not be modified. Only the link between the MSS and the mobile host requires a modified protocol. The proposed protocol, called *indirect* TCP (I-TCP), has been implemented [BB95b] and demonstrated to achieve much better throughput than the standard TCP, especially for mobile hosts which move across the non-overlapping cells, and when the sender is located far away from the receiver.

The idea of *indirect protocols* can be extended further to other layers of OSI and to include such application protocols as *ftp*, *http* (used in the World-Wide Web), and remote procedure calls. To illustrate this further, let us consider the *http* protocol, where the client has to make a separate TCP connection in order to reach each single Web page. While this is a reasonable solution for the fixed host, it is unacceptable for the mobile host communicating on a slow wireless link. Indeed, since each TCP connection requires a 3-way connection set up and a 4-way connection drop,⁹ the resulting overhead will be unacceptable on a wireless link.

A better solution is to let the MSS *represent* a wireless client by opening and closing such TCP connections for it. The mobile client would then open just one *http* connection with its MSS. Such a solution would require writing a separate protocol to handle the wireless link between the client and the MSS and would leave the link between the MSS and the server unchanged. Thus, only the part of the protocol dealing directly with the wireless and mobile host would require modification. Making the MSS a local agent or *proxy* of the wireless client is

⁹To establish the TCP connection, the sender has to send a special packet with the initial sequence number to which the receiver responds with its own sequence number. The third packet is an acknowledgment from the sender. To close the TCP connection, four packets have to be exchanged: the sender sends the connection closing packet, which is acknowledged by the receiver, then the receiver also closes the connection on its side and the sender sends an acknowledgement.

natural choice since the MSS is “close” to the wireless link and can monitor it better than some more remote intermediary.

The concept of indirection can, in general, lead to the modification of standard client-server protocols to a client-proxy-server model, where clients are represented (to certain extent) by proxies (their local MSS) when dealing with remote servers. Such proxies can do most of the work for the clients and contact them only when it is absolutely necessary. Among the issues to be resolved are handoffs which result from mobile client moving between different cells *during* a session with the server, and authorization and security issues dealing with clients using proxies outside their home environment, etc.[BBMI93]

3.2 Information Services

As we go up in the OSI hierarchy, we have to address the issue of interactions of mobile wireless clients with the information resources on the fixed network. Network information resources can be accessed literally from anywhere via wireless links of varying quality and possibly varying tariffs.

Performance Metrics

Information content and methods of the information delivery depend on the location of the user. Hence, user location is an important and dynamically changing parameter. Cost of information will vary depending on the location of the user, who may now face the choice of getting desired the information now, but for higher cost, or later, for less cost. Finally, an additional precious resource, battery power, has to be minimized when interacting with the fixed network. We speculate that in the client’s interactions with information services (such as the World-Wide Web), measures such as *queries per Hz* and *queries per watt* will be important. The analogous performance criteria in cellular telephony is cell capacity: the number of telephone calls which may be handled in a cell per unit of time. In the future, a query, a request for a page, will be the analog of a telephone call in terms of performance metrics. Consequently, we believe that in the future wireless services, the cell capacity will be measured in terms of the number of queries which the local MSS can handle per unit of time as well as the number of queries which a mobile client can issue per unit of power.

Issues in Dissemination of Information to Mobile Clients

Wireless Information services will be characterized by their geographic *scope*. *Wide-area* services such as stock market information, will be offered on a national scale. *Macroservices*,¹⁰ such as weather, would be provided on a regional basis, with regions extending to tens or hundreds of miles. The geographic scope of *microservices* such as traffic conditions will extend to the size of the current cells: a few miles in diameter. Finally, one could imagine *picoservices* in an area corresponding to future *picocells*, perhaps hundred meters in diameter. For example, parking availability could be provided within such a scope.

Designing support for information services for mobile users will require addressing the following key new issues:

- Interoperability and adaptability to networking environments ranging from high-bandwidth wireless LANs to low-bandwidth cellular and infrared communication links.
- Energy-efficient data access for terminals running on self-contained power supplies
- Support for mobility and disconnection
- Support for active services; e.g., triggers, periodic data delivery, etc.

Below, we describe the impact of the each of the above issues on the design of wireless information services of the future.

There are two basic modes of information dissemination to be considered: *publish* and *provide on demand* [IV94]. The latter is the traditional client-server approach (the client submits a request and the server responds). The former (“publish”) is different [IVB94b]. The server either periodically broadcasts information or sends it on a specific multicast channel [OPSS93, IV96]. Such published data will eventually be filtered by the client. The server assists the client in this filtering process by providing a directory of published information. There are many advantages of the publishing mode: broadcasting the most frequently accessed data items (hot spots) saves bandwidth since it cuts down the

¹⁰The granularity of services does not directly correspond to the cell granularity (macrocells, microcells) and should not be confused with it.

number of separate but identical provide-on-demand requests. Also, it saves client energy by avoiding energy-consuming uplink transmissions from client to server. Providing the directory helps the client to tune selectively only to relevant information and remain in *doze mode* for the rest of the time. This strategy saves considerable amount of energy as demonstrated in [IVB94b, IVB94a]. Examples of periodically published data include stock quotes, traffic, and weather conditions.

Each cell's bandwidth can be divided into three channels:

- uplink channel
- on-demand downlink channel
- broadcast downlink channel

How much bandwidth is allocated to each channel depends on the cell, and is a function of the cell population and the pattern of requests. For example, one cell may carry stock market information on its broadcasting channel, another cell's server may decide, due to a limited interest in stocks, to provide stock information only on demand.

In general, cells will differ in the type of radio infrastructure and consequently by the amount of available bandwidth. Thus, not only the content, but also structure and the mode of information presentation will be location-dependent. There is an obvious need for a protocol suite which would make this dependency hidden from the user who relocates between such cells. Some of the services will have to be continued upon crossing cell boundaries with minimal service disruption. Thus, there is a need to handle *service handoff* in a manner analogous to the way in which cellular handoff is handled now. To ensure rapid response to client requests, a “hot list” of items which the client accesses most frequently will have to be re-bound to a new set of addresses when the client enters a new cell. Therefore, each cell must provide a “directory of services,” mapping services to their mode of their delivery (broadcast, multicast, or on-demand) and channel. This directory must include local services as well as global, wide-area services, and must be provided on a standard channel. The exact mapping between services and their addresses is not visible to the user. The system seamlessly rebinds the services according to the allocation used in the new cell.

Inadequacy of the current concept of information services for wireless and mobile environments

There are already several efforts to modify network information services, such as Web browsers, to deal with wireless and mobile clients. This work is currently performed by groups at MIT [K+94], Rutgers [ABIN95], the University of Washington [VB94], and Xerox [WSA+96]. Before we summarize these efforts, we point out the main inadequacies of the current World-Wide Web protocols for mobile clients:

- Page addressing based on the location of servers.
- Lack of presentation autonomy
- Lack of display independence and presentation adaptability
- Stateless servers

We address these issues, in turn, below.

Location-Based Addressing

In the http protocol, each page has a fixed IP address. Each client has to make a connection (ftp connection) to that address in order to access this page. Thus, there is only one address and one access mode (on-demand) which is currently supported. This is inadequate for location-dependent information services for wireless and mobile users, where the content of the presented page changes with the user's location, and where the method of page delivery varies from cell to cell. For example, the *traffic conditions* page will have different content in New Brunswick than in Princeton, and its method of delivery may vary as well; for instance it may be published every minute in Princeton, while provided on demand in New Brunswick.

Thus, pages should be identified independent of the location of the server which offers them. The best solution would be to use the same address for each location-based page in every location, and create a binding to the local address analogously to the way "800" telephone numbers in North America are bound to actual telephone numbers [Che92, OPSS93, ABIN95].

Presentation Autonomy

The local MSS should have the autonomy to decide how a particular remote service is going to be presented to users registered in its cell. Consider again the example of the stock information service: the MSS may decide to publish (broadcast) stock quotes for the S&P100 stocks every minute, broadcast stock quotes for the S&P500 stocks every 5 minutes and provide other quotes on demand with a cache invalidation service[BI94], which broadcasts periodically reports only about significant changes of the stock values. MSS may also decide that special events such as a stock reaching a new high require a special broadcast message which should be sent to subscribers. The remote stock server will not and should not be aware of this autonomous decision of the MSS and different MSSs may decide to “carry” the stock information services differently in order to best utilize their resources and perhaps also maximize the profit.¹¹

Presentation Adaptability

By display independence, we mean the ability to present a page on a range of devices including the telephone (through a voice processing card and prompted menu interface enabling the user to enter selections from the telephone keypad), palmtops with small limited resolution, and powerful laptops equipped with full color monitors. Presentation adaptability allows different presentation of a page depending on the current load on the network, available bandwidth, and energy resources on the client. For example each link should have *resource prerequisites* associated with it and only links which are “eligible” for the current environment should be displayed on the client’s machine. In this way, different clients may see different *views* of the same service. In fact, the same client may see different views of the same service at different times in the same cell due to the varying load on the network.

Stateless Servers

Web servers are *stateless*, that is, they do not keep any information related to the state of their clients. Although this is a reasonable design for current Internet users, this approach is less reasonable for low-power clients connected by low-bandwidth links. By retaining information about a client, a server can deal more effectively with such issues as location-dependent data and presentation adaptability.

¹¹No one knows what the tariff structure will be for the future wireless services but one may assume that the MSS will have to pay the remote server for syndicating its service and will generate a profit by reselling this service to its local residents.

Efforts to Modify Web Services for Mobile and Wireless Clients

The DataMan group at Rutgers has developed an authoring interface [ABIN95] to design location-aware information services. Pages are categorized as location-dependent and location-independent. For example, the *printer* page is location-dependent and provides directions to the nearest printer from the user's current location. Such pages are stored on multiple MSSs with possibly different contents, depending on the MSS's location. Location-dependent pages are accessed via a new mechanism, called *nearcast*, by associating a unique multicast address with each such page. The mobile client accesses a location-dependent page by multicasting a request for that page. The *nearest* MSS which has a copy of the page and is a member of its multicasting group responds to the client's request.¹² This is similar to subject-based naming in the spirit of [Che92]. The mobile client always uses the same address despite the possibility that the underlying actual address of the requested page varies.

The *scope* of a page is another concept which is associated with a location-dependent page. The scope is the set of locations in the immediate neighborhood of the current location in which the page remains invariant. Thus, the client needs to refresh the page content only when moving out of the page's scope, rather than upon each crossing of a cell. The current system provides support for multiple granularities of location information. In this way, the scope of the page can be characterized, as, say "Rutgers Campus" without the need to specify all the MSSs on campus.

The DataMan prototype provides features for autonomous presentation of a service by an MSS. Thus, a given page can be provided either on demand or published periodically, depending on the structure and volume of requests in each cell.

Finally, DataMan provides page invalidation through multicasting. If a content of a given page changes in time, a page invalidation message is multicasted by the server on a specific, predefined multicast address.

In the Mobisaic project [VB94] (discussed in Chapter 14) at the University of Washington, the location-dependent context of mobile hosts is abstracted as <dynamic-variable, value> pairs. Changes to the values of such variables

¹²In this way, the nearcast address plays a role analogous to "800" numbers (in North America) which are bound to possibly different numbers depending on the area where it is used.

are propagated to interested parties using a publisher-subscriber mechanism. Based on this availability of “mobile computing contexts” of users, the authors present two interesting concepts (and their implementations):

- **Dynamic URL:** A URL that references dynamic variables and whose values are resolved when the URL is accessed. This allows the same URL to specify different document contents.
- **Active Documents:** Documents whose contents change or become invalid when a specified set of dynamic variables changes.

The paper also presents the necessary extensions to HTML for authoring active documents.

Other work at University of Washington [Wat94] (presented in Chapter 13) involves principles for partitioning functionality at the application-level between a mobile, wireless client and a proxy on the static network. As an example of a wireless application, the paper presents W*, a Web browser. This browser is explicitly designed to optimize wireless communication and allow for a proxy to cache and/or parse documents on behalf of the mobile client.

Work at MIT [K+94] deals with optimization of wireless link between the mobile unit and the MSS. In order to avoid expensive page transfers, an alternative solution is proposed in which pages are generated locally by the mobile client. Thus, the mobile client, upon requesting a particular Web page, receives the HTML code which it runs to locally generate the page.

3.3 Power Management

Power management is already being addressed in hardware. Features like switching off backlighting, providing the CPU with a power-efficient doze mode, replacing disks with more energy-efficient semiconductor memories, are just a few examples of steps towards lowering power consumption. There is now a growing pressure to improve energy management further at the software level.

Energy-efficient software is possible only if energy-management features already exist on the hardware level. Thus, the CPU should have a low-energy mode, a disk that can be spun down, a radio receiver that can be switched off when idle, etc. Energy-aware software algorithms will take advantage of these features by

maximizing the time the CPU spends in the doze mode, keeping the receiver off most of the time, minimizing energy consuming transmissions, spinning down the disk, etc.

There is substantial room for power saving features on the application layer. We have already discussed the publishing mode of information dissemination in which the server periodically broadcasts “hot spots” of information without explicit requests from the clients.

There is research underway at a variety of levels:

- **Operating Systems:**

- The work of [WWDS94] to schedule CPU operations for low energy (discussed in Chapter 17)
 - The work at Sun [B+93] on energy-efficient features in operating systems (discussed in Chapter 16)

- **Communication Protocols:**

- MAC Layer: such as in IEEE 802.11 and CDPD
 - Application Layer: by using publishing mode, avoiding uplink transmissions, etc.

- **I/O and File systems:**

The work of [DCK+94] dealing with spinning down the disk.

The main objective of the low-energy features of 802.11 is to let the mobile unit keep the receiver off most of the time. This is accomplished by careful synchronization between the MSS and the mobile unit. When the MSS receives a message addressed to one of the mobile units which reside in its cell, it does not forward the message directly, but buffers it. Periodically, the MSS broadcasts a “preview” which is a list of identifiers of all the mobile units which have an outstanding message to be delivered by the MSS. The mobile unit turns its receiver on synchronously to listen to the periodic preview. If the preview contains the the mobile unit’s identifier, then the mobile unit either keeps its receiver on until the message is received, or it informs the MSS as to when it wants to receive its messages. In either case, if the mobile unit receives only a few messages a day (even the most heavy e-mail users typically do not receive more than 50-100 messages a day) it can keep the receiver off most of the time,

saving considerable energy. For example in [IGP95], the energy savings for the WaveLAN card are provided for a protocol which is a variation of 802.11. With the average load of 30 messages a day, the receiver will be “on” for approximately 10 seconds to a minute, rather than hours, leading to significant energy savings. CDPD offers a similar low-energy feature. In [IGP95] some of the effects of varying preview size are studied. It would be useful to put some of these features on the application layer, so that applications can influence the size and the frequency with which the preview is broadcasted, based on their tolerance of delay. It is also argued there that avoiding uplink transmissions from the mobile unit to the MSS, which are very energy consuming (especially in outdoor environments) is highly desirable. Therefore, solutions in which the mobile unit stays active immediately following the preview message, which contains unit’s identifier, are preferable.

Further reduction in transmissions can be achieved by means of transport layer modifications which cut down on the number of acknowledgements from the mobile recipient. Perhaps, analogously to congestion control, we need power control by controlling the number of acknowledgments depending on the power resources of the client.

The energy management issues dealing with I/O operations are discussed in Chapter 18, where it is demonstrated that replacing the disk with flash memory can lead to very substantial energy savings.

3.4 Systems Issues in Mobile Computing

The key issue which will affect the basic design of file systems and, to a large degree, database systems as well, is the issue of disconnection.

Operating System and File System Issues

Disconnection support is an important aspect of mobility support. Mobile users will be often disconnected from the shared file systems and should be allowed to work on their local file system until they reconnect again. On reconnection, the changes to the file are propagated to the file server. The Coda file system which is based on the Andrew file system allows for disconnected operation[KS92] (see Chapter 19). A disconnected Coda client can continue to work by using any data in the cache. When the client reconnects, the updates are reintegrated. Update conflicts are handled similarly to reconnection process of a partitioned

replicated database. It is possible that updates of a disconnected client will be discarded if some other connected client has modified the files.

The Ficus replicated file system developed at UCLA [HPGP92] also supports a form of disconnected access. In Ficus, each file is replicated to enhance read availability. Further, it adopts a primary copy approach to replication control. First, the update is propagated to the primary copy and later messages are sent to the secondary copies. The exclusive-write/multiple-readers scheme of synchronization is implemented by the use of write tokens. Only a site with the write token can modify the file. Thus a disconnected client can be considered as a server with write tokens for the files for which it has the primary copy. However, on reconnection, communication to the secondary sites needs to be accomplished. The file system proposal at Columbia [TD92] attempts to alleviate the problem of expensive global communication of a replicated file system by using a lazy update scheme. Further, to reduce update propagation, two different read semantics (loose read, a best-effort value, and a strict read, giving the most recent version) are provided.

Disconnected operation has been added to the AFS as part of the Little Work project at the University of Michigan [HHRB92, HH93]. Before disconnection, most recently accessed data is cached at the client. Upon reconnection, file system updates that are recorded in a log are propagated to the server. While in disconnected mode, the mobile unit performs local operations but network operations are logged and deferred. When the network connection is re-established, the cache manager iterates through the log and updates are transferred to the server. If a conflict is detected, then the replay agent notifies the user that manual resolution is needed.

Work in this area [KS92, HPGP92, TD92] has focused on environments in which the users are connected either over a fast network or totally disconnected from the network. Weak connectivity (wireless connection) and energy restrictions have not been considered.

Caching in mobile environments is addressed in [BI94]. There, the MSS does not keep the state information about clients but rather periodically broadcasts invalidation reports so that, upon reconnection, the mobile unit can determine which cached items are valid and which items are invalid without performing expensive uplink requests.

Database Systems Issues

It is clear that mobile clients will not run a complete database system locally. Most likely, mobile users will use very lightweight client software on their machines and use remote database servers to process most queries and transactions.

Clients will, however, perform simple queries and transactions on local data, especially when they are disconnected. Such transactions will generally use operations of three types:

- Operations against closely-held data, in which the chance of data conflict is low.
- Operations that commute, such as increment and decrement.
- Operations on a *partition* of the actual data. An example of this is a set of tickets where a mobile unit may be allocated a subset of the set of available tickets prior to disconnection [SS90].

If transactions follow the above restrictions, updating the shared database at reconnect time is straightforward. Special applications may require more complex reconnection protocols and, possibly, human intervention. The impact of mobility on transaction processing is described in [KJ96].

During periods when the mobile unit is connected, queries should be processed in an energy-aware manner. This involves making tradeoffs between energy used in communication and energy used locally. It may also mean choosing a query processing plan that is suboptimal in terms of time, but optimal in terms of energy (for example, by reducing the time the disk is spinning). This issue is explored in detail in [AG92].

Many applications of mobile computing, especially those relating to travel and traffic information, require the processing of location-dependent queries in which the user's location is a parameter of the query. Processing such queries is discussed in [IB92].

Recovery is an interesting issue since the mobile unit is subject to catastrophic failures like theft, physical destruction, etc. Thus, there is no such thing as truly stable storage on the mobile unit. As a result, the local log may have to be uploaded periodically to the fixed network as described in [AB94a]. Furthermore, since much of the mobile user's interaction is of a non-database nature,

it is desirable to apply database-style recoverability to a broad set of mobile applications.

Disconnection management is even more important in the database context than it is in the context of file systems. Because disconnection is part of routine mobile computing, rather than a failure mode, the database buffer manager must be able to deal with periods of disconnection without the imposition of high-overhead at the time of reconnection. [BI94] shows new techniques for cache invalidation suitable for mobile and wireless environments.

For many, perhaps most, applications, the user interface will be application specific. However, general-purpose interfaces require an alternative to SQL-based languages. Graphical user interfaces designed for workstations tend not to map well to the small screen of mobile computers. For casual database users, and users of PDAs, a pen-based interface is required. Preliminary work in this area is described in [AHK92].

A more detailed discussion showing which database issues will be most affected in mobile and wireless environments is presented in [1] as well as in [IB94].

3.5 Research Prototypes

There are several research prototypes currently being developed, many of which are described in this book. The Ubiquitous Computing project at Xerox, the Infopad project at Berkeley, and Coda at CMU were among the most significant early efforts. Xerox's work is described in this book in Chapters 2, 15, and 17. The CMU work is described in Chapter 25. The Infopad work [SLBR94] represents one extreme in the model of computation for a mobile system. It gives the smallest role to the mobile station. While this imposes the disadvantage that the mobile unit is useless without a network connection, it offers significant advantages in power (100 milliWatts for the entire mobile unit), and in the ability to devote the entire device to user interaction. In this work, it is assumed that the mobile devices ("Pads") are always connected. Computation and storage is handled by stationary machines on the backbone (wired) network, and the mobile device (a multimedia terminal) need handle only bitmaps sent over the wireless network (an RF line).

4 BOOK CONTENT

In this section, we review the content of the book by briefly summarizing each chapter.

After the first two chapters, the ordering of chapters is generally consistent with the bottom up view of the OSI hierarchy. Thus, we begin at the network layer by reviewing mobility management issues and ad-hoc networking (in Chapters 3 through 6), and then continue with the transport layer issues in Chapters 7 through 10. Chapters 11 through 14 discuss higher level protocols for information services, and Chapters 15 through 21 general systems issues in mobile computing. Finally, Chapters 22 through 25 review some research prototypes and applications.

Chapter 2 presents a comprehensive summary of the most prominent and extensive research prototype: the Ubiquitous Computing project at Xerox. In the Xerox model, there is no fixed attachment between the user and the mobile terminal. Xerox's vision predicts that small computers will become as universal as paper pads and that mobile users will possibly use different terminals at different locations. The PARCTab system integrates a palm-sized mobile computer into an office network with small-cell wireless communication. Three different types of devices, PARCTab (a palm sized computer), PARCPad (an electronic notepad), and Liveboard were built to illustrate three different scales of devices. The paper presents the basic philosophy of ubiquitous computing and describes the system design and application components of this pioneering work. Thus, it serves as a good overall guide to the broad issues facing the designer of a mobile system.

Mobility management is addressed in Chapters 3 and 4. Chapter 3, "Scalable Support for Transparent Mobile Host Internetworking," by David Johnson, describes the current status of work on Mobile IP in the form of a review of the latest IETF proposal. This chapter reviews the main concepts introduced by the IETF proposal such as registration, tunneling, foreign agents, etc. It also discusses the location update mechanisms and location caching. Chapter 4, "Mobility Management in Internet and Cellular Environments," by B. R. Badrinath and T. Imielinski, compares work on mobility management on the Internet with the work on location management in cellular telephony. Two distinct research communities are working on this problem. The paper attempts to compare different approaches and points out similarities and differences in network and system assumptions.

Work on ad-hoc networking protocols is presented in Chapters 5 and 6. Chapter 5, “Dynamic Source Routing in Ad Hoc Wireless Networks,” by David B. Johnson and David A. Maltz, presents a routing protocol for ad-hoc networks which is based on dynamic source routing. An interesting feature of this protocol is that it adapts quickly to routing changes when hosts movement is frequent, yet it requires little overhead when hosts move less often. The overall overhead of the protocol is measured through simulations and is determined to be quite low, falling to just 1% of the total data packets transmitted. Chapter 6, “Routing over Multi-Hop Wireless Network of Mobile Computers,” by Charles Perkins and Pravin Bhagawat, offers an alternative approach to routing in ad-hoc networks. The mobile hosts are viewed as routers which periodically advertise their view of the interconnection topology to other mobile hosts. The key challenge is to avoid advertising unstable routes. To this end, a route updating mechanism, which is based on the past history, is designed. The routing algorithm is an extension of the today’s distance vector approach (Bellman - Ford) with additional features which reduce the impact of fluctuations in unstable routes.

The transport-layer issues in wireless and mobile environments are discussed in Chapters 7 through 10. Chapter 7, “Improving the Performance of reliable Transport Protocols in Mobile Computing Environments,” by Ramón Cáceres and Liviu Iftode, introduces the basic challenges of designing transport-layer protocols for wireless and mobile environments. As we have pointed out earlier, reliable transport protocols such as TCP interpret delays and losses resulting from handoffs and disconnections as signs of network congestion. This degradation is quantified in this chapter through the measurement of protocol behavior in a wireless networking testbed. It is demonstrated that the current TCP implementation produces unacceptably long pauses in communication during cellular handoffs (800 milliseconds or longer). The need for differentiation between motion-related and congestion-related packet losses is postulated and preliminary solutions are sketched. Chapter 8, “I-TCP, Indirect TCP for Mobile Hosts,” by A. Bakre and B. R. Badrinath, is an implementation of a solution to the problem reported in Chapter 7. I-TCP splits the end-to-end transport-layer interaction into two parts: the fixed network part and the wireless part. The fixed network part remains unchanged (which is beneficial since it is impractical to assume that the entire TCP will have to be changed to accommodate mobile hosts), while the wireless part, between the MSR and the mobile host is new. The wireless part of the I-TCP “understands” that the handoff should not cause exponential backoff. The experimental results present significant performance gains (throughput) especially for wide-area communication and with moves occurring between nonoverlapped cells. Chapter 9 provides an extension of TCP for wireless environment which is also based on the concept of indirec-

tion (as in Chapter 8). The proposed “Mowgli” communication architecture is designed to split the channel with end-to-end control into two parts with a store-and-forward type interceptor. This interceptor allows implementation of two separate subsystems, one wireline-oriented and the other, wireless-oriented. Thus, similarly to the work in Chapter 8, the existing TCP/IP communication infrastructure can be retained, while the special purpose protocol has to be implemented only for the wireless link.

Chapter 10, “Asynchronous Video: Coordinated Video Coding and Transport for Heterogeneous Networks with Wireless Access,” by J. Reason, L. C. Yun, Allen Lao and David Messerschmidt, addresses transport-layer problems dealing with the transmission of asynchronous video over wireless networks. The main goal of this work is to minimize the perceptual delay, that is, the delay perceived by the end user. Two related techniques are introduced in order to reduce this perceptual delay: coding the video in such a way that there is no need for transcoders, and asynchronous reconstruction of the video presentation at the receiver. For instance, the low motion areas are more tolerant of delay than the high motion ones. This leads to the additional compression through discarding fine-resolution information in the areas of high motion and transporting information in low-motion areas less often. Analytical and simulation results are presented that demonstrate substantial gains in traffic capacity. Allowing the worst-case transport-layer delay to increase may actually increase the overall network traffic capacity. This is particularly advantageous on wireless access links, though it is incompatible with the approach taken by existing MPEG compression standards for video.

Chapters 11 through 14 discuss the higher level issues (above transport layer) dealing with the wireless access to information. Chapter 11, “Wireless Publishing: Issues and Solutions,” by T. Imielinski and S. Vishwanathan introduces the concept of *publishing* into client-server interactions in wireless environments. Publishing is a periodic broadcasting of information by the MSS over a wireless medium. Publishing mode is intended for information which is likely to be requested frequently and its advantages include reduction in uplink traffic, as well as power savings on the client platform. Chapter 11 provides an overview of different addressing methods for publishing mode, which are based either on temporal or on multicast addressing. Addressing allows clients to tune selectively to relevant pieces of published data and switch to a low-power mode of operation (CPU in the doze mode, the receiver switched off, etc.), while no relevant information is being published. Algorithms for adaptive scheduling of publishing mode in conjunction with the standard “on-demand” client-server interactions are presented as well.

Chapter 12, “Broadcast Disks: Data Management for Asymmetric Communication Environments,” by S. Acharya, R. Alonso, M. Franklin and S. Zdonik views repetitive broadcast (publishing mode) as a “disk on air”. The paper describes different scheduling policies with frequency of broadcast proportional to the access probability. Similarly to [IVB94a], periodic broadcast is viewed as a form of a cache on air and another level of memory hierarchy.

Chapters 13 and 14 present first attempts (along with the DataMan project described earlier) to study the design and development of location-dependent information services.

Chapter 13, “Application Design for Wireless Computing,” by Terri Watson, presents principles for partitioning functionality at the application-level between a mobile, wireless client and a proxy on the static network. As an example of a wireless application, the paper presents W*, a Web client browser. This browser is explicitly designed to optimize wireless communication and allow for a proxy to cache and/or parse documents on behalf of the mobile client.

Chapter 14, “Mobisaic: An Information System for a Mobile and Wireless Computing Environment,” by G. Voelker and B. Bershad, deals with primitives necessary to build location-dependent applications. The location-dependent context of mobile hosts is abstracted there as <dynamic variable, value> pairs. Changes to the values of such variables are propagated to interested parties using a publisher-subscriber mechanism. Based on this availability of “mobile computing contexts” of users, the authors present two interesting concepts (and their implementations): dynamic URLs and active documents. The paper also presents the necessary extensions to HTML for authoring active documents.

Chapter 15, “Providing Location Information in a Ubiquitous Computing Environment,” by Mike Spreitzer and Marvin Theimer, reviews security issues dealing with the control information about location. A new architecture is presented in which users have primary control over location information.

Chapters 16 through 21 deal with broad systems issues of mobile computing, ranging from the operating and file system issues to database applications for mobile computers.

Chapter 16, “Unix for Nomads: Making Unix Support Mobile Computing,” by Michael Bender, et al., addresses the design issues for extensions of Unix¹³ (and, specifically Solaris) to support nomadic computing. It is argued there

¹³Unix is a trademark of X/Open

that substantial changes ranging from the kernel level through the user command set are necessary to support mobile computing. The paper discusses the kernel changes to support power management, the PCMCIA standard, serial line networking and new electronic mail applications designed to specifically deal with slow, serial line connections.

Energy-management issues are discussed in Chapters 17 and 18 (along with Chapter 11).

Chapter 17, "Scheduling for Reduced CPU Energy," by Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker, discusses modifications in the CPU scheduling policies to reduce energy consumption. The metric of interest is how many instructions are executed for a given amount of energy, or MIPJ (Instructions per Joule). Trace driven simulations are provided to compare different classes of schedules with varying computing speeds allowing the CPU to use lower voltage and consequently, reduce energy consumption.

Chapter 18, "Storage Alternatives for Mobile Computers," by F. Douglis, et al., investigates the impact of new storage devices such as small hard disks, flash-memory disk emulators, and flash memory cards on file system design. As in Chapter 17, one of the main issues is power consumption. It is concluded that using magnetic disks for file storage on mobile computers is basically unacceptable from an energy point of view. To extend battery life, it is possible to spin down the disk when it is idle. But even then, the power consumption is an order of magnitude higher than for a file system using flash memory. In general, a flash-memory file system has the most attractive quality with respect to energy and performance. However, the price and capacity limitations are still major drawbacks.

Chapters 19 and 20 deal with the impact of disconnection on file system design. Disconnection is one of the central issues in mobile computing and Coda was the first implementation of the file system explicitly supporting disconnection. Chapter 19, "Disconnected Operation in the Coda File System," by James Kistler and M. Satyanarayanan, describes the design and implementation of the Coda file system at CMU. Clients view Coda as a single, location-transparent Unix file system. While a client is disconnected, its file system requests are serviced solely based on the contents of local caches. File replication is used in order to improve availability. While Chapter 19 describes design and implementation issues, Chapter 20, "Experience with Disconnected Operation in a Mobile Computing Environment," by M. Satyanarayanan et al., presents qualitative and quantitative data on file access using Coda file system over a period of about 2 years. This chapter describes the results of quantitative evalua-

tions including disk space requirements, reintegration latency, and the value of log optimization. Hoarding and hoard profiles, which describe the path names identifying objects of interest to the user, lead to cache-content optimization. The impact of more advanced concepts such as multi-level hoard priorities and “meta expansions” to minimize the size of hoard profiles is described as well.

Chapters 21 and 22 deal with some of the database issues in mobile computing. Chapter 21, “Mobility Support for Sales and Inventory Operations,” by N. Krishnakumar and Ravi Jain, describes the use of an escrow mechanism to allow a distributed server architecture in which a mobile transaction can execute using the data held by the server to which it is directly connected. The mobile sales and inventory applications are examples of a well-defined market segment, but the solutions described in the paper are more general, and can apply to any set of mobile users who share a common database. Chapter 22, “On Query Processing Strategies for Mobile Computing,” by Masahiko Tsukamoto, Rieko Tanaka, and Shojiro Nishio, considers queries that request data from mobile hosts or request data about mobile hosts (such as location). This work considers several different query processing strategies based on either broadcast or notification. The optimal strategy depends upon parameters of the system (computers and network), the degree of mobility of the mobile hosts, and rate of query submission.

Chapters 23 and 24 describe some of the ongoing research prototype work. Chapter 23, “The Case for Wireless Overlay Networks,” by Randy Katz and Eric Brewer describes a heterogeneous architecture for future mobile information services that is based on a wireless overlay network. In an overlay network several possibly different wireless networks may coexist. For example, a high bandwidth infrared network can be overlaid with a lower bandwidth radio network to provide connectivity between the areas of IR coverage. The resulting architecture is discussed in detail along with the new issues of overlay network management and application support services. A new testbed project “Bay Area Research Wireless Access Network (BARWAN)” is described. Chapter 24, “The DIANA Approach to Mobile Computing,” by A. Keller et al., provides an approach to deal with heterogeneity in the mobile environment. Not only must applications run on the wide variety of mobile platforms available (and likely to become available), but a given *instance* of an application may have to move from one platform to another – and possibly to a different kind of platform. The DIANA system attempts to separate user interface and display management from the underlying system and communication support. The DIANA environment also addresses the problem of application development by allowing applications to be tested in a simple environment before being interfaced into DIANA.

Chapters 25 and 26 focus on two important applications of mobile computing: maintenance and repair systems and travel information systems. Chapter 25, "The CMU Mobile Computers as Maintenance Assistants," by Daniel P. Siewiorek and Asim Smailagic, describes a "wearable computer" called "Vu-Man." An earlier version of VuMan was used as a campus navigational assistant. The current version is targeted at maintenance operations by the U. S. Marines. It provides a means for the user to access maintenance manuals while performing repairs in locations where it is not practical to access a printed manual. (Current procedures require two people – one to do the work and the other to read the manual.) In addition to providing manual access, this system allows input of maintenance data via a simple user interface. Chapter 26, "Genesis and Advanced Traveler Information Systems (ATIS)," by Shashi Shekhar and Duen-Ren Liu, pertains to the application domain of transportation systems. Unlike many of the well-publicized services that provide routing information based on a static database of maps and real-time positional data, the Genesis project at Minnesota is including real-time traffic data as a parameter in routing decisions. This creates significant demands on the system, since traffic data must be processed promptly and triggers must be used to inform vehicles whose routes may be affected by accidents or reported congestion.

REFERENCES

- [AB94a] A. Acharya and B. Badrinath. Checkpointing distributed applications on mobile computers. In *Proc. Third International Conference on Parallel and Distributed Information Systems, Austin, Texas*, pages 73–80, September 1994.
- [AB94b] A. Acharya and B. R. Badrinath. Delivering multicast messages in networks with mobile hosts. In *Fourteenth International Conference on Distributed Computing Systems*, June 1994.
- [ABIN95] A. Acharya, B. R. Badrinath, T. Imielinski, and J. Navas. Towards Mosaic like location dependent services. Technical report, Rutgers University, May 1995.
- [AG92] R. Alonso and S. Ganguly. Energy-efficient query optimization. MITL-TR-33-92, Matsushita Information Technology Laboratory, Princeton, NJ, 1992.

- [AHK92] R. Alonso, E. Haber, and H. F. Korth. A database interface for mobile computers. In *Proceedings of the 1992 Globecom Workshop on Networking of Personal Communication Applications*, December 1992.
- [AK93] R. Alonso and H. F. Korth. Database system issues in nomadic computing. In *Proceedings of ACM-SIGMOD 1993 International Conference on Management of Data, Washington D.C.*, 1993.
- [B+93] M. Bender et al. Unix for nomads - making Unix support mobile computing. In *First Usenix Symposium on Location Dependent Computing*, August 1993. Also included in this volume.
- [Bat94] Bud Bates. *Wireless Networked Communications*. McGraw-Hill, 1994.
- [BB95a] Ajay Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *15th International Conference on Distributed Computing Systems*, May 1995.
- [BB95b] Ajay Bakre and B. R. Badrinath. M-RPC: A remote procedure call service for mobile clients. In *First ACM Conference on Mobile Computing and Internetworking*, May 1995.
- [BBMI93] B. R. Badrinath, Ajay Bakre, Roy Marantz, and T. Imielinski. Handling mobile hosts: A case for indirect interaction. In *Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, pages 91–97, October 1993.
- [BI94] D. Barbara and T. Imielinski. Sleepers and workoholics: caching strategies in mobile environments. In *Proceedings of ACM-SIGMOD 1994 International Conference on Management of Data, Minneapolis, Minnesota*, pages 1–13, May 1994.
- [BI96] B. R. Badrinath and T. Imielinski. Mobility management in Interenet and cellular telephony. In *this volume*, 1996.
- [Cal88] George Calhoun. *Digital Cellular Radio*. Artech House, 1988.
- [CB92] Ananth Chandrasekaran and R. W. Broderson. A portable multimedia terminal for personal communications. *IEEE Communications Magazine*, pages 64–75, December 1992.
- [Che92] D. Cheriton. Dissemination-oriented communication systems. Technical report, Department of Computer Science, Stanford University, 1992.
- [CI95] Ramón Cáceres and Liviu Iftode. Improving the performance of reliable transport protocols in mobile computing environments. In *IEEE Journal on Selected Areas in Communications*, pages 850–857, June 1995. Also included in this volume.

- [DCK+94] F. Douglis, Ramón Cáceres, M. Frans Kaashoek, P. Krishnan, Kai Li, Brian Marsh, and Joshua Tauber. Storage alternatives for mobile computers. In *SOSDI Usenix Symposium*, November 1994. Also included in this volume.
- [FZ94] G. Forman and J Zahorjan. Mobile wireless computing. *IEEE Spectrum*, 1994.
- [Goo91] David J. Goodman. Trends in cellular and cordless communications. *IEEE Communications Magazine*, June 1991.
- [HH93] L. B. Huston and Peter Honeyman. Disconnected operation for AFS. In *USENIX Conference on Location Dependent Computing*, August 1993.
- [HHRB92] Peter Honeyman, L. Huston, J. Rees, and D. Bachmann. The Little Work Project. In *Proceedings of the Third IEEE Workshop on Workstation Operating Systems*, April 1992.
- [HPGP92] J. S. Heidemann, T. W. Page, R. G. Guy, and G. J. Popek. Primarily disconnected operation: Experiences with Ficus. In *Proceedings of the Second Workshop on the Management of Replicated Data*, November 1992.
- [IB92] T. Imielinski and B. R. Badrinath. Querying in highly mobile and distributed environments. In *Proceedings of the Eighteenth International Conference on Very Large Databases, Vancouver*, 1992.
- [IB94] T. Imielinski and B. R. Badrinath. Mobile computing - solutions and challenges in data management. *Communications of ACM*, October 1994.
- [IGP95] T. Imielinski, M. Gupta, and S. Peyyetti. Energy efficient data communication and filtering. In *Proceedings of Usenix Symposium on Location Dependent Services*, April 1995.
- [IV94] T. Imielinski and S. Vishwanathan. Adaptive wireless information services. In *Proceedings of SIGDB*, October 1994.
- [IV96] T. Imielinski and S. Vishwanathan. Wireless publishing: Issues and solutions. In *this volume*, 1996.
- [IVB94a] T. Imielinski, S. , and B. R. Badrinath. Energy efficient indexing on air. In *Proceedings of ACM-SIGMOD 1994 International Conference on Management of Data, Minneapolis, Minnesota*, pages 25–37, May 1994.
- [IVB94b] T. Imielinski, S. Vishwanathan, and B. R. Badrinath. Power efficient filtering of data on the air. In *Proc. of EDBT*, volume 779, pages 245–258, March 1994.

- [JMP95] D. Johnson, A. Myles, and C. Perkins. Overview of MobileIP. In *RFC*, April 1995.
- [K+94] Frans Kaashoek et al. Dynamic documents: Mobile wireless access to WWW. In *Workshop on Mobile Computing Systems and Applications*, December 1994.
- [Kar91] P. Karn. MACA - a new channel access method for packet radio. Technical report, Qualcomm, 1991.
- [Kat94] Randy Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, pages 6–17, 1994.
- [KJ96] Narayanan Krishnakumar and Ravi Jain. Mobility support for sales and inventory applications. In *this volume*, 1996.
- [KMM95] R. Kohno, R. Meidan, and L. Milstein. Spread spectrum access methods for wireless communications. *IEEE Communications Magazine*, 33(1):58–68, January 1995.
- [KS92] J. Kistler and M. Satyanarayanan. Disconnected operation in the CODA file system. *ACM Transactions on Computer Systems*, 10(1), February 1992. Also included in *this volume*.
- [OPSS93] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus – an architecture for extensible distributed systems. In *Proc. of the 14th ACM Symposium on Operating Systems Principles*, December 1993.
- [PB94] Charles Perkins and Pravin Bhagawat. Routing over multi hop wireless network of mobile computers. In *Proc. ACM SIGCOMM'94*, 1994. Also included in *this volume*.
- [PGH95] Jay E. Padgett, Christoph G. Gunter, and Takeshi Hatori. Overview of wireless personal communications. *IEEE Communications Magazine*, pages 28–41, January 1995.
- [SLBR94] S. Seshan, M. T. Le, F. Burghard, and J. Rabaey. Software architecture of the InfoPad System. Mobidata Workshop, November 1994.
- [SS90] N.R. Soparkar and A. Silberschatz. Data-value partitioning and virtual messages. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Nashville, pages 357–367, April 1990.
- [Tan81] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, New Jersey, 1981.

- [TD92] C. Tait and D. Duchamp. An efficient variable consistency replicated file service. In *Usenix File System Workshop*, pages 111–126, May 1992.
- [VB94] G. Voelker and B. Bershad. Mobisaic: An information system for mobile wireless computing environment. In *Workshop on Mobile Computing Systems and Applications*, December 1994. Also included in this volume.
- [Wat94] Terri Watson. Application design for wireless computing. In *Workshop on Mobile Computing Systems and Applications*, December 1994. Also included in this volume.
- [WCW91] David J. Goodman W. C. Wong. A packet reservation multiple access protocol for integrated speech and data transmission. Technical report, Rutgers University, September 1991.
- [WSA+96] Roy Want, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldbert, John R. Ellis, and Mark Weiser. The PARCTab ubiquitous computing experiment. In *this volume*, 1996.
- [WWDS94] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *First Symposium on Operating Systems and Design (OSDI)*, November 1994. Also included in this volume.

THE PARCTAB UBIQUITOUS COMPUTING EXPERIMENT

Roy Want, Bill N. Schilit,
Norman I. Adams, Rich Gold, Karin Petersen,
David Goldberg, John R. Ellis, Mark Weiser

Xerox Palo Alto Research Center, Palo Alto, California, USA

ABSTRACT

The PARCTAB system integrates a palm-sized mobile computer into an office network. This project serves as a preliminary testbed for Ubiquitous Computing, a philosophy originating at Xerox PARC that aims to enrich our computing environment by emphasizing context sensitivity, casual interaction and the spatial arrangement of computers. This paper describes the Ubiquitous Computing philosophy, the PARCTAB system, user-interface issues for small devices, and our experience developing and testing a variety of mobile applications.

1 INTRODUCTION

For the past 30 years the operating speed and component density of digital electronics has steadily increased, while the price of components has steadily decreased. Today, designers of consumer goods are incorporating digital electronics into more and more of their products. If these trends continue, as we expect they will, many everyday items will soon include some form of computer.

Although computers are becoming ever more common in appliances such as VCRs, microwave ovens, and personal digital assistants, they remain largely isolated from one another and from more powerful desktop and laptop machines. We believe that in the future many computers will provide more valuable services in combination than they can in isolation. Ideally, many kinds

¹This work was supported by Xerox and ARPA under contract DABT63-91-C-0027. Portions of systems described here may be patented or patent pending.

of specialized machines will work together via networks to let users access and control information, computation and their physical and electronic environments.

In the Computer Science Laboratory (CSL) at Xerox PARC we have established a number of research projects to explore this vision, which we call Ubiquitous Computing. This paper presents the results of the PARCTAB project, an experiment intended to clarify the design and application issues involved in constructing a mobile computing system within an office building. The PARCTAB system provides a useful testbed for some of the ideas of the Ubiquitous Computing philosophy, which is described briefly in the next section. The system is based on palm-sized wireless PARCTAB computers (known generically as "tabs") and an infrared communication system that links them to each other and to desktop computers through a local area network (LAN). Although technological and funding limitations forced us to make numerous compromises in designing the PARCTAB hardware, nevertheless the system, as described in Section 3, meets most of our design goals. Likewise the small size and low resolution of the PARCTAB displays requires an innovative user interface design to allow efficient text entry and option selection. Our solutions are presented in Section 4.

A community of about 41 people at Xerox PARC take part in the system's operation and in PARCTAB application development, which are covered in some detail in Sections 5 and 6. To date, we have developed and tested more than two dozen PARCTAB applications that allow users to access information on the network, to communicate through paging and e-mail, to collaborate on shared drawings and texts, and even to monitor and control office appliances. Descriptions of the various PARCTAB applications as well as data on users' experiences with them are given in Sections 7 and 8, respectively.

By designing, constructing, and evaluating a fully operational mobile computing system and developing applications that exploit its unique capabilities, we have gained some insight into the practical benefits and real-world problems of such systems. In the paper's final section, we collect these lessons and present some of the many intriguing ideas that the PARCTAB project has spawned for future work in Ubiquitous Computing.

2 UBIQUITOUS COMPUTING

As inexpensive computers add limited intelligence to a wider variety of everyday products, a new model of computing becomes possible.

2.1 The Ubiquitous Computing Philosophy

This new technology aims for the flexibility of a far simpler and more ubiquitous technology: printed text. Depending on the need, print can be large or small, trivial or profound, verbose or concise. But though print surrounds us in myriad forms, it does not dominate our thoughts the way computers do today. We do not need to log on to road signs to use them or turn away from our colleagues to jot notes on a pad of paper. Similarly, ubiquitous computers would demand less of our concentration than present commercial computer interfaces that require users to sit still and focus their attention. Yet through casual interaction they would provide us with more information and all the advantages of an intelligently orchestrated and highly connected computer system.

Creating such an intuitive and distributed system requires two key ingredients: communication and context. Communication allows system components to share information about their status, the user and the environment—that is, the context in which they are operating. Specifically, context information might include such elements as:

- The name of the user's current location;
- The identities of the user and of other people nearby;
- The identities and status of the nearby printers, workstations, Liveboards, coffee machines, etc.;
- Physical parameters such as time, temperature, light level and weather conditions.

The combination of mobile computing and context communications can be a powerful one [40, 29, 27, 31, 32, 30]. Consider, for example, an employee who wants to show a set of figures to his manager. As he approaches her office, a quick glance at his tab confirms that the boss is in and alone. In the midst of their conversation, the employee uses the tab to locate the data file on the network server and to request a printout. The system sends his request by

default to the closest printer and notifies him when the job is finished. Many more examples of the Ubiquitous Computing philosophy are presented in Mark Weiser's article "The Computer of the 21st Century" [39].

2.2 A Ubiquitous Computing Infrastructure

Attaining the goals of Ubiquitous Computing will require a highly sophisticated infrastructure. In the ideal system, a real-time tracking mechanism will derive the locations and operational status of many system components and will use that context to deliver messages more intelligently. Users will be able to choose from among a variety of devices to gain mobile, high-bandwidth access to data and computational resources anywhere on the network. These devices will be intuitive, attractive and responsive. They will automatically adapt their behavior to suit the current user and context.

Although one can speculate about the design of a future system, unfortunately the components needed to build such an infrastructure have yet to be invented. Current processors and microcontrollers are slow and power-hungry compared to their likely descendants 10 years from now. We reasoned that we could bridge this technology gap by constructing an operational system that resembles an optimal design. Despite the inevitable compromise of some engineering characteristics, we could then use the system to assess the advantages and disadvantages of Ubiquitous Computing as if we had glimpsed into the future.

It is impossible to predict the range of device forms and capabilities that will be available a decade from now. We therefore based our device research on size, a factor that is likely to continue to divide computers into functional categories. A useful metaphor that highlights our approach is to consider the traditional English units of length: the inch, foot and yard. These units evolved because they represent three significantly different scales of use from a human perspective.

- Devices on the inch scale, in general, can be easily attached to clothing or carried in a pocket or hand.
- Foot-sized devices can also be carried, though probably not all the time. We expect that office workers will use foot-sized computers similar to the way that they use notebooks today. Some notebooks are personal and are carried to a particular place for a particular purpose. Other notepads are

scattered throughout the work environment and can be used by anyone for any purpose.

- In the future office there will be computers with yard-sized screens. These will probably be stationary devices analogous to whiteboards today.

2.3 Ubiquitous Computing Experiments at PARC

Researchers at PARC have built computer systems at the three scales described above [41]:

<i>inch</i>	PARCTAB, a palm-sized computer;
<i>foot</i>	PARCPAD, an electronic notepad;
<i>yard</i>	Liveboard, an electronic whiteboard.

These experimental devices use different mechanisms for communication and computation within the building's distributed system. The Liveboard is not mobile and connects directly to an Ethernet. Our mobile devices extend battery life by using low-power communication technologies: infrared (IR) signalling for the PARCTAB and near-field radio [6] for the PARCPAD. We have also investigated how operating system design can reduce power consumption [43] and this is well suited to mobile computers. The PARCPAD and Liveboard are described elsewhere by Kantarjiev [16, 11] and Elrod [8].

Our goals for the PARCTAB project were:

- To design a mobile hardware device, the PARCTAB, that enables personal communication;
- To design an architecture that supports mobile computing;
- To construct context-sensitive applications that exploit this architecture;
- To test the entire system in an office community of about 41 people acting as both users and developers of mobile applications.

3 PARCTAB SYSTEM DESIGN

We set several design goals for the PARCTAB hardware. It had to be physically attractive to users, compatible with the network, and capable of modifying its behavior in response to the current context. We believed that in order to fulfill these goals the PARCTAB had to be small, light and aesthetically pleasing enough that users would accept it as an everyday accessory. It needed reliable wireless connectivity with our existing networks and a tracking mechanism capable of detecting its location down to the resolution of a room. It had to run on batteries for at least one day without recharging.

We also believed that the PARCTAB's user interface had to let people make casual use of the device, even if they had only one free hand. The screen had to be able to display graphics as well as text. We wanted users to be able to make marks and selections using electronic ink, so the screen needed touch sensitivity with a resolution at least equal that of the display. Furthermore, the cost of the hardware and the network infrastructure had to be within reasonable limits so that we could deploy the system for lab-wide use.

Cost was not the only limitation on our design options. Some factors were also limited by available technology, such as the device's communication bandwidth, display resolution, processor performance and battery capacity.

3.1 PARCTAB Mobile Hardware

We carefully weighed the limitations and requirements above when making the engineering decisions that shaped the final appearance (Figure 1) and functionality of the PARCTAB hardware. One primary trade-off balanced weight, processor performance, and communications bandwidth against battery life. We also had to strike a compromise between screen resolution and the device's size, cost and processor speed.

Packaging

We believed an ergonomic package would be essential if people were to carry and use the tab regularly. We thus enclosed the PARCTAB in a production-quality custom plastic case with a removable belt clip. The tab is about half the size of current commercial personal digital assistants (PDAs), at 10.5cm x 7.8cm x 2.4cm (4.1in x 3.0in x 0.95in). It weighs 215g (7.5oz). We designed the tab so that users could choose either one-handed use with buttons or two-handed

use with a stylus. Because the package is symmetric, the tab can be used in either hand—an important feature for left-handers who wish to use the stylus. To convert from right- to left-handed use, the user executes a setup command that rotates the display and touch-screen coordinates by 180 degrees.

Display and Control Characteristics

We found that commercially available touch-sensitive displays provided adequate resolution for our needs. We chose a 6.2cm x 4.5cm (2.4in x 1.8in) LCD display with a resolution of 128 x 64 monochrome pixels.

The PARCTAB is most easily operated with two hands: one to hold the tab, the other to use a passive stylus or a finger to touch the screen. But since office workers often seem to have their hands full, we designed the tab so that three mechanical buttons fall beneath the fingers of the same hand that holds the tab (see Figure 1), allowing one-handed use. The device also includes a piezo-electric speaker so that applications can generate audio feedback.

Power Management

Power is the overriding concern that drives most of the design decisions of most small electronic devices, and the PARCTAB is no exception. With more power, there could be faster communication over longer distances, higher-resolution displays, and faster processors. But existing battery technology places stringent limits on the power available for such small components.

We found the prismatic (rectangular) Nicad cell to be the most suitable battery technology given our size, weight and performance goals. Four cells were sufficient to provide a rechargeable power source for the tab while meeting all the packaging requirements. We designed the core of the device around a 12MHz, 8-bit microcontroller (87C524), an Intel 8051 derivative, for two reasons. First, its on-board EPROM, RAM and I/O ports ensured a compact design. But equally important, this processor can be programmed to enter a low-power mode. The PARCTAB takes advantage of this mode when idle in order to extend battery life. The display, touch screen, additional RAM and the communication electronics can also be powered down by the microcontroller.

During normal operation a tab consumes 27mA at 5V. In low-power mode it consumes less than 30 μ A. We considered nominal use to be 10 minutes per hour, eight hours per working day. In operation, however, we found that the one-



Figure 1 The PARCTAB mobile hardware

day use requirement was easily met. In fact, using a battery storage-capacity of 360mAh, the typical tab need only be charged once per week. A smaller battery may suffice, in which case we estimate that the PARCTAB could be reduced to one-third of its current weight and volume if it were produced today by a commercial electronics company instead of a research lab. We anticipate that within a few years the functions of the PARCTAB probably could be put into a watch.

3.2 PARCTAB Communication

Limited space and power constrained our choice of a wireless communication technology to just two options: radio and infrared (IR). We chose 880nm IR to exploit the small, inexpensive IR components that were commercially available. These offered low power consumption at the modest communication speeds of 9600 and 19200 baud. Because IR signals are contained by the walls of a room, this technology also made it easier to design a cellular system. Moreover, IR

communication is unregulated. A radio link would have required more space, higher power equipment and potentially government operating licenses.

We decided that a cellular system [5] would best handle the competition for bandwidth that inevitably would arise in a building-wide system supporting many users. By creating small, room-sized communication cells (nanocells), we could minimize the communication distance from the hub to the mobile user, reducing power needs concomitantly. Since the radiated signal would be blocked by walls, messages would be more secure than if they were broadcast widely. Users are also less likely to interfere with one another's signals in a cellular system, although some situations—such as heavy tab use during a break in a large meeting—can still place large loads on the IR transceivers. Finally, small cells enable the system to pin down a user's location to the resolution of a room.

The tab infrared network [1, 26] thus consists of nanocells defined by the walls of a room surrounding an IR transceiver. Large open rooms and hallways may also support nanocells if transceivers are carefully placed out of communication range of each other. Transceivers connect to a LAN through the RS-232 ports of nearby workstations.

Transceiver Design

A transceiver serves as a communication hub for any PARCTABS located in its particular cell. Typically its communication radius is about 20 feet—less if limited by the walls of an office. The transceiver hardware performs numerous functions in addition to transmission and reception, including:

- Coding and decoding infrared packets;
- Buffering data;
- Executing link-level protocol checks (e.g., format or checksum);
- Providing a serial interface to a workstation's RS-232 port;
- Indicating visually its communication status.

We designed the transceiver conservatively to ensure reliable communication. For transmission, two dozen IR emitters are placed at 15 degree intervals on a circular printed circuit board. For reception, two detectors provide a total

viewing angle of 360 degrees (Figure 2). The transceiver is designed to be attached to a ceiling, preferably in the middle of a room as this usually gives an unobscured communication path over the required area. But since transceivers and PARCTABS can sense infrared light reflected from surfaces, it is not necessary that there be a line of sight between the two for them to communicate. Thus a single transceiver usually covers a room completely.

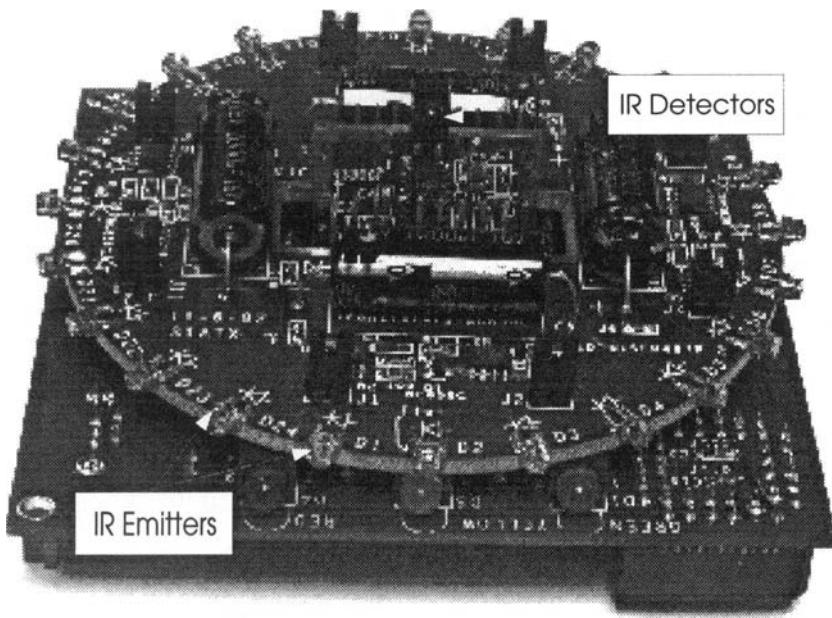


Figure 2 The PARCTAB transceiver

Local Area Network Interface

We found the approach of extending an existing LAN to provide wireless nanocellular communication very attractive for a number of reasons. The additional cost is small because the LAN wiring already exists. Most offices in our building are equipped with at least one workstation that has a spare RS-232 port. We thus had to string only a small amount of additional phone cable to connect ceiling-mounted transceivers to our UNIX workstations and, through them, the ethernet. And since well established communication mechanisms already exist between workstations in commercial distributed systems, we did

not have to reinvent that infrastructure. Transceivers could be attached to networks of other platforms, such as the PC or Macintosh, in much the same way.

Transmission Control

The decision to use infrared communication prompts a further design issue: how to enable many PARCTABS to share the medium? Conventional IR detectors have difficulty tuning narrow frequency ranges, ruling out the possibility of using frequency-division multiplexing to divide the bandwidth into several subchannels. We thus chose a simple digital packet-contention scheme that shares the medium using time-division multiplexing.

In this scheme, all data is bundled into packets formed by the baseband modulation of an IR carrier into a sequence of pulses. The pulses are uniform—all have a duration of $4\mu s$ —but the gaps between them are not. The variable duration of the silence between pulses encodes the data bits. The durations of the gap encoding a logic 1, logic 0, packet-start synchronization, and data-byte synchronization are all unique and may be decoded using a simple algorithm. By defining data as the absence of a signal, this technique minimizes power consumption, since the infrared carrier is switched off for most of a transmission.

The link-layer packets are divided into several fields, as shown in Figure 3 below. The packet type field is always sent at 9600 baud, and a subfield of the packet type defines the speed at which the rest of the packet will be transmitted. This permits variable speed transmission and allows future high-speed systems to remain backward-compatible. The present system transmits packets at 9600 and 19200 baud.

PKT TYPE	LENGTH (0-255)	DESTINATION	SOURCE	DATA PAYLOAD	CS
1	1	4	4	3-247	2

Figure 3 Format of the data fields for a link-layer IR packet (lengths in bytes).

The second field contains the length of the packet. Packets vary in length from 14 bytes for most uplink packets to a maximum of 256 bytes for a downlink

packet. Next follow unique 4-byte addresses of the destination and source devices, up to 247 bytes of payload data and finally a 2-byte checksum.

We assumed that communications traffic inside a cell would normally be low since applications are driven by user-generated events, such as button clicks. We thus expected a screen update to be followed by a relatively long silence while the user made the next selection. Because we also assumed that small packets generated under lightly loaded conditions would be delivered promptly, we chose to use a symmetric non-persistent carrier-sense multiple-access (CSMA) protocol to provide access to the IR channel. This protocol simply uses carrier sense and a random-exponential backoff whenever the channel is busy. It does not wait for a packet currently occupying the channel to complete before entering a new backoff period [33].

Reliability and Interference

The PARCTAB system cannot detect packet collisions because any IR transmission creates such a powerful signal that it saturates the local receiver, making it impossible to detect a packet sent simultaneously by another device. Mobile hardware can avoid losing link-layer packets by setting a bit in the packet type field that requests an acknowledgment. When a transceiver sees the request bit set, it immediately transmits a reply back to the sender. In a multiple-access network this type of acknowledgment is quite reliable, since the fact that the request was received implies that there was no contention and therefore the acknowledgment should also not encounter contention [36]. A PARCTAB sets the request bit for some types of tab packets—user events, for example—and then, if no acknowledgment arrives, resends the packet a fixed number of times until finally generating an audible alarm to the user. In principle, downlink packets sent from a transceiver to a PARCTAB could also use this mechanism. Instead, as described in Section 7, we ensure downlink reliability at a higher level of protocol.

When a PARCTAB is in view of two rooms—when in a hallway, for instance, with doors opening into two cells—both cell transceivers might acknowledge event packets simultaneously, corrupting the acknowledgment signal at the PARCTAB. To avoid this problem transceivers that are close enough to interfere with each other are given different network addresses and only acknowledge packets addressed to them, although they still transfer all the packets that they receive to the LAN. Whenever a PARCTAB enters a new cell the system notices events that it produces (e.g., beacons or button clicks) and instructs the tab to use a new transceiver address.

4 USER-INTERFACE DESIGN FOR PALM-SIZED COMPUTERS

As we developed applications for the PARCTAB, it became clear that a traditional user interface designed for the 640 x 480-pixel color display of a typical PC or workstation would not work well on the PARCTAB's 128 x 64-pixel monochrome display [25, 42]. Indeed, the PARCTAB's tiny screen, offering less than half the area of most PDA displays, forced us to devise innovative ways to select, display and enter information in a very limited space. As advancing technology produces higher resolution displays that can pack more information onto a small screen, some of the problems we faced will undoubtedly disappear. But text and symbols can shrink only so much before they become too small to read. Also, as displays increase in resolution, new devices will probably get commensurately smaller. Many of the user-interface solutions we describe below will thus remain relevant.

4.1 Buttons vs. Touch Screen

Since the PARCTAB is well suited for casual, spur-of-the-moment use, we did not want to compel users to free both hands to operate the device. The user interface thus had to allow users to control applications with the device's three buttons, its touch screen or a combination of both. This requirement complicated the interface design because a user selecting an item on the screen with the buttons alone must then be presented with an intermediate screen allowing her to invoke an operation on that item. Consequently, application developers must decide whether to require two-handed use (of both stylus and buttons) or whether to increase the number of screens in their program so that all the functions can be accessed via the buttons.

We found one convention that seems to solve this problem best, and developers incorporated it into several tab applications. It works as follows: on clicking the middle push-button, a menu of commands pops-up. The top and bottom buttons then move the cursor up and down, while a second click of the middle button selects the command on which the cursor currently rests. On screens that display scrolls or lists of text, the top and bottom buttons scroll the list up or down, respectively. If menus are designed intelligently, then users must usually just click the middle button twice to execute the most common action. Two-handed users can press an on-screen button to pop up the menu and can then point with the stylus to select an item directly.

We have also settled on a preferred interface style for using the push-buttons and the stylus to navigate a tree data-structure. The operator uses the stylus to navigate down through the hierarchy one screen at a time and clicks the middle button to navigate upward. This method works efficiently because users descending the hierarchy must at each level make a choice, a task performed simplest with the stylus. Ascending the tree, on the other hand, requires a user to repeat the same operation over and over, a task well suited to repeated push-button action.

4.2 Spurious Event Prevention

Because the PARCTAB applications often run elsewhere on the network there can be modest delays between a button click or screen touch and the update of the screen 5. The delay between event and response can occasionally cause errant behavior in the user interface. Consider the case in which a menu contains a button icon that selects another screen with its own button icon in a similar position. A user tapping the first button with the stylus might create multiple pen-events, either by unintentionally bouncing the pen on the touch surface or by impatiently tapping the button twice. The initial event will trigger a transition to the next screen, but the latter events could then cause additional, unwanted selections. We solved this problem by adding a field called an epoch to the event packet structure. Every time an application transmits a screen change, it also increments the epoch number in the PARCTAB. Any events that were in the application input queue with a previous epoch number can now be discarded, thus preventing any spurious transitions.

4.3 Text Display

We anticipated that it might be difficult to read text on the PARCTAB because its small display can show only eight lines of 21 (6 x 8-pixel) characters. In practice, this proved not to be a problem, as our popular e-mail application exemplifies. Word-wrap and hyphenation algorithms can often fit three or four words across the screen. The 8-line display is also small enough to update quickly despite the limited communication bandwidth.

Users scroll through text either by clicking the top or bottom push-buttons or by touching the upper or lower half of the display. The experience is similar to reading a newspaper column through a small window that can be moved up or down by the flick of a pen. Although this is relatively efficient, it is

nevertheless often useful to filter text information before it is displayed. For example, the PARCTAB e-mail application replaces lengthy message headers with a compressed form that includes only the sender and an abbreviated form of the subject field.

4.4 Text Entry

We experimented with two methods of text entry: graphic, onscreen keyboards and Unistrokes, a novel approach to handwriting recognition. Unistrokes [13] is similar to Graffiti, a system marketed subsequently by Palm Computing.

Keyboard Entry

An onscreen keyboard requires both an array of graphic keys arranged in typewriter format and an area to display text as it is entered. We have experimented with several layouts. The first presents key icons across lines 2 through 8 of the screen and displays the characters that have been “typed” on line 1, which scrolls left and right as necessary to accommodate messages longer than 21 characters. A delete-last-character function bound to the PARCTAB’s top push-button allows easy correction of mistakes. One of the other push-buttons serves as a carriage return that terminates an entry. We found that users could enter about two characters per second using this keyboard layout. Experiments with smaller keyboards show that they lower typing accuracy.

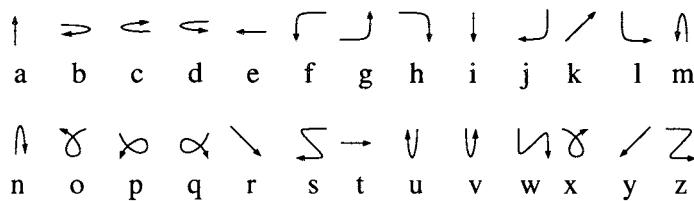


Figure 4 The Unistroke alphabet

Unistrokes

Techniques for handwriting recognition have improved in recent years, and are used on some PDAs for text entry. But they are still far from ideal since they respond differently to the unique writing characteristics of each operator. We

have experimented on the PARCTAB with Unistrokes, which depart from the traditional approach in that they require the user to learn a new alphabet—one designed specifically to make handwriting easier to recognize.

For each letter in the English alphabet there is a corresponding Unistroke character which can be drawn in a single pen stroke. The direction of the stroke is significant (Figure 4). To minimize the effort required to learn to write in Unistrokes, all Unistroke characters are either identical to English letters (e.g., L, S and Z) or are based on a characteristic feature of the corresponding English letter (e.g., the cross of T). We found that most people can learn the Unistroke alphabet in under an hour.

Because Unistroke characters are directional and better differentiated than English letters, they require less processing to recognize reliably. Because the characters are single strokes, users can draw each Unistroke character right on top of the previous one, using the entire screen. Thus the strokes themselves need not appear on the writing surface, but instead the PARCTAB neatly displays the corresponding English characters. Practiced Unistrokers found the simplicity and speed of text entry very attractive.

4.5 Option Selection

The PARCTAB's small screen makes it difficult to present users with a long list of options. We tried a number of different methods.

Text and Icon menus

The PARCTAB screen size places rather severe limits on the number of text or icon options in a menu. Vertically, eight lines of text fit onscreen. Spreading three text buttons per line across the display increases the number of selections to 24. Arranging 16 x 16-pixel icons in an uncluttered format yields about 15 options per screen (see Figure 1). Larger lists require alternative approaches.

Scrolling Lists

Some applications, including Tabmail and Arbitron (Figure 5), present choices in a scrolling list with each option represented by a single line of text. The item on which the cursor rests is highlighted; options that are unavailable because they do not make sense in the current context are crossed out. As users press

the top and bottom push-buttons to move the cursor up and down, the list scrolls as necessary to expose more options.

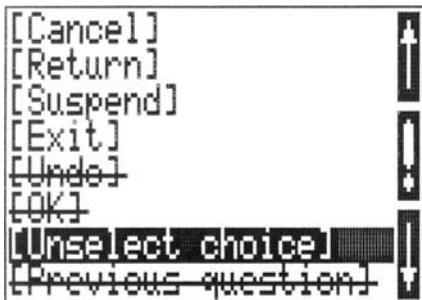


Figure 5 A screen from the PARCTAB Arbitron application

We considered using the “proportional” scroll bars common in PC user interfaces to allow fast touch-screen navigation of large ordered lists on the PARCTAB. This scheme takes the horizontal or vertical position of a screen touch as a percentage of the total screen dimension, then moves the cursor to a similar position in the long list of options. Unfortunately, we found that the resolution of the touch-screen restricts accurate selection to lists smaller than the maximum number of pen positions that can be resolved. The PARCTAB can resolve 128 horizontal positions per line.

We also chose not to use this type of interface element because it demands continuous display feedback for each movement of the pen. Typically the feedback would be generated remotely, rather than by the tab itself, because the application generating the feedback is running elsewhere on the network. The contention that would result between pen events and continuous display updates would make poor use of the communication bandwidth.

Elision and Incremental Searches

We used the PARCTAB to evaluate the efficiency of two somewhat more sophisticated methods for selecting one item (such as a name or word) from a large ordered list (such as a directory or dictionary): elision and incremental searching. Elision is based on k-ary search techniques. The system divides the list into 15 portions of roughly equal size and displays the first item in each

section, followed by an ellipsis (Figure 6). The display ends with the last item in the list.

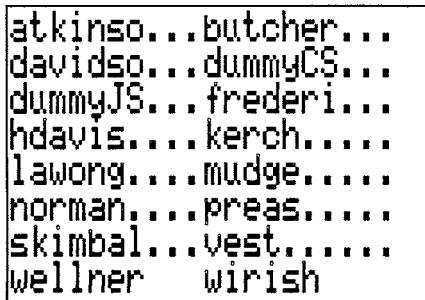


Figure 6 A screen from the PARCTAB locator application

The user selects the target item if it is displayed. Otherwise, selecting any ellipsis redraws the screen to show an expansion of the selected region of the list into 13 smaller portions as before. (The very first and last items in the complete list are always displayed so that users can navigate back to other regions.) The user continues “zooming in” on a particular region until the target item appears.

Elision is reasonably efficient. Because the PARCTAB screen can display 16 abbreviated words with ellipses between them, users need make at most $\log_{16} N$ selections to reach any item, where N is the size of the list. To select one item among one million, for example, requires no more than six selections. The mean word length in the American Heritage online dictionary, containing 84433 words, is 8.9 characters. A user typing a word from this dictionary on a graphic keyboard must thus make 8.9 selections, on average. Elision, by comparison, can bring up any word in this dictionary with just four selections.

Incremental search techniques, implemented in the PARCTAB dictionary application, can do nearly as well. Here the user types the first few letters of the item. With each letter entered, the application narrows the list of possible matches and displays the closest eight. We found that this method identified the desired word after 4.3 characters on average—thus 5.3 selections, since one more tap is needed to choose the correct match from the eight choices.

PARCTAB applications have made successful use of both elision and incremental searches. We observed advantages and disadvantages for each. Elision is the

more general method, since it performs well even when the ordered list has no special properties. It also usually requires fewer selections—especially if it is refined so that the system adjusts the size of the subsections to fall between guide words that have been frequently selected. Many PARCTAB users object to elision, however, because it demands hard thinking to pick the appropriate ellipsis.

5 PARCTAB SYSTEM ARCHITECTURE

A multilayer system architecture integrates the PARCTAB hardware into the PARC office network so that network applications can easily control and respond to mobile devices based on the devices' current context. Although the PARCTABS themselves behave more like terminals than independent computers, they do execute local functions in response to remote procedure calls. PARCTABS also generate events that are then forwarded by transceivers and the infrared gateways that manage them to processes called tab agents, which run on network machines. The agents keep track of the mobile tabs and link them to workstation-based applications. PARCTAB applications are generally event-driven, much like X11 or Macintosh programs. Figure 7 illustrates relationships among PARCTABS, transceivers, gateway and agent processes, and applications.

Developers can link into their applications a code library that hides the details of PARCTAB tracking, message routing, and error recovery. Of course, any application can obtain a tab's current location as needed so that the program can modify its behavior appropriately. We developed the PARCTAB system in the Unix programming environment (SunOS 4.3.1) running on SparcStation 2 connected by an ethernet. Communication between Unix processes is achieved using Sun RPC.

5.1 PARCTAB Processing Capabilities

One perennial issue in distributed systems design is the question of partitioning: how much computation should be performed by the mobile devices, and how much by larger computers fixed to the network. One alternative is to execute much of an application's interface locally in the mobile client, similar to the way America Online, Compuserve and Prodigy put most of their user interface

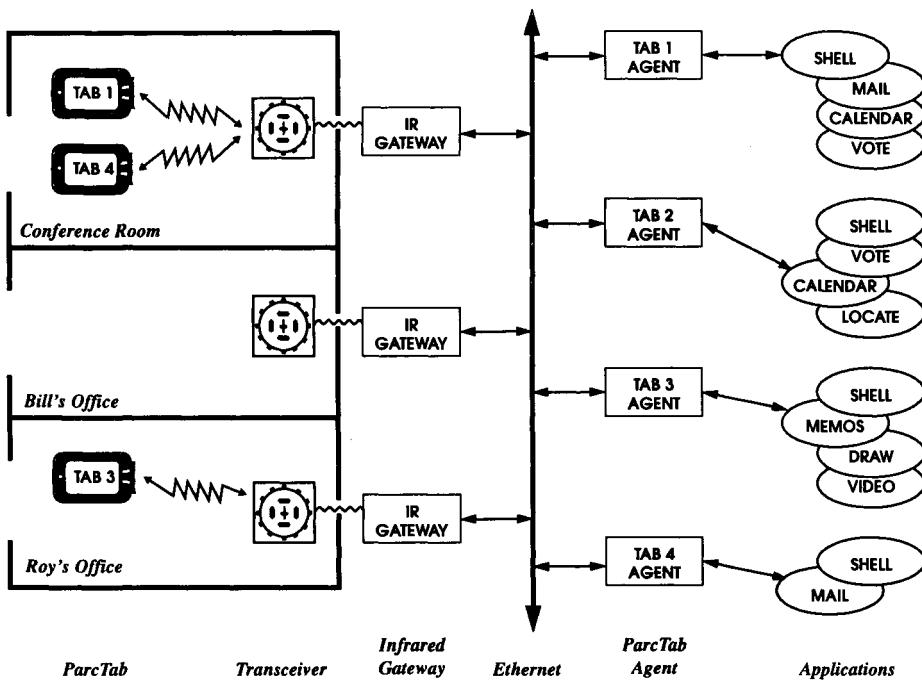


Figure 7 The PARCTAB system architecture

onto users' PCs. At the extreme defined by PDAs, a tab might even run whole applications and communicate only occasionally with the network.

Although this approach might reduce the load on the IR channel, it requires a fast processor and much memory. But using today's technology, the power requirements of state-of-the-art CPUs cannot be satisfied by conventional batteries of reasonable size and weight. With a 12MHz processor and 128Kb of memory, the PARCTAB is roughly equivalent in computational power to a PC from the early 1980s. To date, we have thus used tabs primarily as input/output devices that rely on workstation-based applications for most computation. In this model the mobile computer becomes a display device similar to a more conventional graphics terminal. Recently, however, we have experimented with a few applications that execute solely in the tab: taking notes using Unistrokes, for example, and browsing files downloaded from the network.

Tab Remote Procedure Call Mechanism

A simple communication mechanism called a tab remote procedure call (T-RPC) allows applications to control various PARCTAB resources, such as the display, touch screen, local memory and tone generator, while remaining oblivious to a tab's location and any underlying communication errors. This mechanism has been incorporated into a library of procedures available to application designers. When an application makes a call into the library, the library assembles a request packet in a format defined by a request/reply protocol.

PAYLOAD TYPE	SEQUENCE NUMBER	FUNCTION			MORE FUNCTIONS	END
		CODE	LENGTH	PARAMETERS		
1	1	1	1	0 - 242		1

Figure 8 Format of IR packet data payload as used by the request/reply protocol (lengths in bytes)

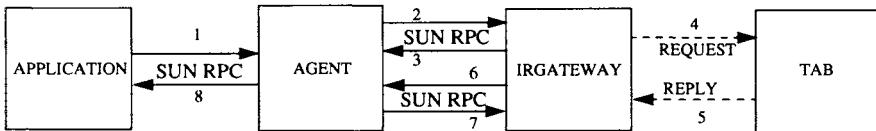
The request/reply protocol is contained in the data payload of the link-layer packet (Figure 8). The tab supports a set of about 30 function codes, several of which can be combined into a single packet. For efficiency multiple function-requests can be batched into a single packet under program control. A few examples of PARCTAB functions are: `display_text`, `display_bits`, `generate_tones`, `set_epoch` and `wake_up`.

An application delivers the request packet to a tab's agent process, which forwards it in turn to the tab. The application then waits for a reply. When the PARCTAB finishes executing the request, it returns a reply packet to the application containing an indication of its success and any appropriate results.

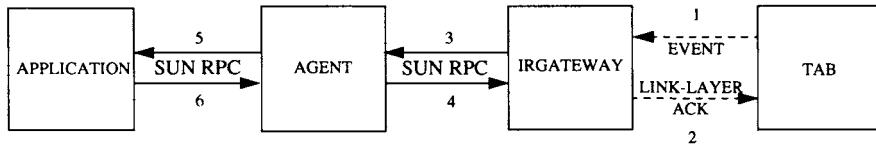
Sometimes a request or reply packet will be lost, or the system will be temporarily unable to determine the location of a tab. In that case, the agent will automatically time-out the reply and will retry the request at intervals defined by an exponential back-off algorithm. The back-off algorithm takes into account whether the tab is detected by the network or not, and whether the tab is free or busy executing another T-RPC request.

Only when a request is matched up with a corresponding reply will the application continue. The agent increments the sequence number for each new request to ensure that retried packets do not inadvertently execute a request twice. The agent likewise discards duplicate replies that result from retries or

detection by multiple transceivers. Figure 9 shows the complete path taken by a T-RPC call made from an application to a tab and back again.



T-RPC (Application to Tab Communication)



Event Notification (Tab to Application Communication)

Figure 9 The path taken by a T-RPC call made from an application to a tab.

PARCTAB Events

When a PARCTAB user presses a button or touches the screen, the device transmits an event signal. The PARCTAB may also generate certain events autonomously, such as a low-battery alert and a beacon. The beacon is a signal transmitted every 30 seconds, even when the device is idling in low-power mode, that allows the system to continue to monitor a PARCTAB's location when it is not active. A similar system has been used to locate people using Active Badges [38, 10, 14]. The power cost of waking up a tab every 30 seconds to emit one packet is not high and, in fact, we also designed the tab to listen for a moment after sending a beacon. If a wake-up request is received in this period the PARCTAB will power-up completely. The system can thus deliver priority messages to the device even when it is not in use.

The packet format used to signal PARCTAB events is similar to that used in the request/reply mechanism. The payload type field distinguishes events, requests and replies. In event packets, the function code is replaced by the appropriate event code.

5.2 Infrared Gateway

The IR-gateway process controls one or more infrared transceivers connected to the serial ports of a workstation. The gateway receives IR packets forwarded by transceivers and delivers them to tab agents. In the reverse direction, the IR-gateway receives packets from an agent over a local-area network, encodes them for IR transmission and delivers them to the appropriate serial port. The transceiver then broadcasts the packets over the IR medium to any tabs within its cell. These packets are coded according to the request/reply protocol described in Section 5.1.

The IR-gateway uses a name service to determine which agent should receive each packet. The gateway looks up the packet's source addresses (i.e., the tab's unique address) in the name-service directory to obtain the network address of the corresponding agent. Each gateway process maintains a long-lived cache of agent network address so that it rarely needs to use the name service.

The gateway also appends a return address and a location identifier to every packet it sends to an agent. The location identifier is a short textual description (e.g., "35-2232") of the location of the transceiver that received the packet. Context-sensitive applications can use the identifier in combination with centralized location databases and services to customize their behavior.

In addition to its main functions, the IR-gateway performs configuration, error-reporting, and error-recovery functions. Gateway processes also handle the flow control that matches low-speed infrared communications with the high-speed local area network.

5.3 Tab Agent

For each PARCTAB there is exactly one agent process, which acts like a switchboard to connect applications with tabs via IR-gateways. An agent performs four functions:

- It receives requests from applications to deliver packets to the mobile PARCTAB that it serves;
- In the reverse direction, it forwards messages (along with location identifiers) from its tab to the current application;

- It provides an authoritative source of tab location information for context-savvy applications;
- Finally, it manages application communication channels.

Since the agent is an intermediary on all messages, it has the most complete information on the location of its tab. Even if the PARCTAB moves to a new cell, its agent will soon receive a beacon signal and update the tab's location accordingly. Whenever the tab's location or status changes, the agent notifies a centralized location service [28] of the tab's last known location and its status: "interactive" if it is being used, "idle" if it is transmitting beacons but no other events, and "missing" if the tab is out of sight.

An agent also manages which application is allowed access to its tab at a particular moment. Because the PARCTAB screen is so small, each application takes over the entire display. Although the tab may run many network applications over time, only one "current application" can receive events from the tab and send it messages at a given moment. In our system, a tab's agent interacts with a special application called the "shell" (see Section 5.4) to decide which application is current.

PARCTAB users can currently choose between two shells: the standard shell described in the next section and an alternative described in Section 6.3.

5.4 Shell and Application Control

The shell is a distinguished application that provides a user interface for launching or resuming other tab applications.

A tab agent launches a shell when the agent is initialized, and if the shell exits, the agent automatically restarts it. When it is current, the shell displays an application menu like that shown in Figure 10 and waits for the user to select an application. If the user chooses to launch a program, then the shell creates a new Unix process, registers it with the tab's agent, and finally instructs the agent to switch to the new application. Whenever a user suspends or exits a PARCTAB application, the agent makes the shell the current application.

The shell and other applications communicate with an agent through the ApControl interface. This interface offers four procedures: register, suspend, resume, and quit. When an application invokes the 'suspend' or 'quit' com-

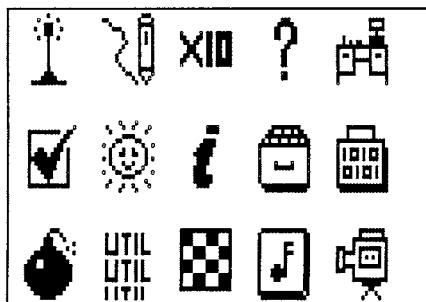


Figure 10 The top-level screen presented by the default Shell

mand, the agent switches control back to the shell. When a user chooses to resume a suspended application or to switch to a newly registered process, the shell calls the ‘resume’ procedure. If an application locks up in some way, a PARCTAB user can transmit a special “agent escape” event that forces the agent to suspend the current application and switch back to the shell.

The shell interface is based on user-customized screens. A screen contains active regions (called buttons) and graphic labels, both of which may be represented by text and bitmaps. Buttons invoke built-in actions: jumping to another screen, starting or resuming an application, playing a tune over the PARCTAB speaker, etc.

When the shell is started it loads a user’s tabrc initialization file, or a standard configuration file if that is not present. The contents of the tabrc file define the buttons, bitmaps, text and active areas that the shell draws on the PARCTAB’s top-level screen. The shell also looks for a user’s tabrc-personal file and uses that to extend the menus described by the tabrc file.

The grammar for the files consists of two parts, as shown below. The first section defines the screen structure displayed on the tab. The second section contains a list of actions, such as running a certain program, that the shell performs when it starts up. In this format, the star (“*”) indicates items that can occur zero or more times; unstarred items occur exactly once.

```

Tabrc   → Part*
Part    → (Initialize Action*)
        → (Screens Screen*)
Screen  → (label: Widget*)
Widget  → (Text text x y invert)
        (TextButton text x y Action)
        (Bitmap bitmap-file x y)
        (BitmapButton bitmap-file x y Action)
Action   → (Screen label)
        (Beep duration octave note ...)
        (Program program-args)
        (Load tabrc-file)

```

5.5 Example of System Operation

To explain how the PARCTAB system operates in practice, consider the following example. A user holding a PARCTAB in Roy's office presses a button. The tab transmits a button event packet and requests an acknowledgment.

A transceiver nearby picks up the signal, transmits an acknowledgment back to the tab, and then forwards the event packet over the serial connection. The IR-gateway process listening to the serial line receives the packet, extracts its source address and looks up the network address for the agent associated with the tab that sent the packet. The gateway stamps the packet with the transceiver's location identifier and its own network address, then sends it off to the agent.

When the agent receives the message, it first verifies that this is not a duplicate of a previous packet. It then forwards the data to whichever application is current. The application decodes the event and triggers a procedure call defined by the application developer.

If, for example, the application wants to update the PARCTAB display, then it calls a tab library function and the transmission process reverses. First, the library procedure packs the application's display data into a T-RPC request packet and sends the request to the appropriate agent. The procedure also blocks the application until the call is completed. Next the agent forwards the packet to whichever IR-gateway sent it a message last.

The IR-gateway encodes the request packet for transmission and sends it over the serial link to a transceiver, which broadcasts the data over the IR medium. When the PARCTAB to which the request is addressed receives the packet, it

decodes and executes the functions and then transmits a reply back to the IR-gateway indicating its success. The gateway duly forwards the reply to the correct agent as described above.

6 DEVELOPING SYSTEM AND APPLICATION COMPONENTS

Members of the experimental community have built PARCTAB applications using three different approaches: Modula-3 libraries, Tcl/Tk and the MacTabbit system. Each offered different levels of access to the PARCTAB and its capabilities.

6.1 Modula-3

Modula-3 was a natural choice to build the first PARCTAB applications because it is also the language for the PARCTAB's system software [21]. It had many characteristics that recommend it for both tasks, along with a number of shortcomings.

Modula-3 and System Development

Modula-3 is a relatively new language; it has a number of features that we believe are valuable in building large systems. These include garbage collection, light-weight threads, type safety, and support for modules and object-oriented programming. PARC's earlier successes using Cedar (an ancestor of Modula-3) for systems work influenced our decision. In addition, we hoped that the combination of type safety and object-orientation would result in higher quality, more reusable code.

Modula-3's threads were important for our design because they simplified the architecture of the IR-gateway and agent. Both are long running servers that interact with many clients at the same time. Each client has its own dedicated thread: if one client doesn't return promptly from a remote procedure call, others are not adversely affected. Building a non-blocking server without threads would require either changing the remote procedure call (RPC) mechanism to make it asynchronous or abandoning RPC in favor of some lower-level communication mechanism.

Modula-3 and Application Writers

Modula-3 also facilitated the development of reusable libraries for tab application writers. For example, we developed an object-based widget library to handle the user interface. The object-oriented approach meant that each addition could build on previous work.

To simplify development work, we also built a PARCTAB simulator in Modula-3. This program uses an X-window to mirror the PARCTAB display and mouse events to simulate the PARCTAB pen and buttons. In many cases developers prefer the simulator to the mobile hardware for program testing.

Although Modula-3 as a language met our needs well, the implementation we used had a number of deficiencies. Modula-3 is still a young language, and so the programming environment lacked certain tools, especially for debugging. In particular, there was no support for debugging multiple threads: tracking down the deadlocks and race conditions that come with multi-threaded programs was particularly challenging. Modula-3 also produced very large runtime images which occasionally taxed even our 64MB workstations.

To compensate for this shortcoming we built support mechanisms into the tab system software. Each process can write selected information to a log file, and system components have network-accessible interfaces for debugging and control. Programmers can use these interfaces to examine and set parameters, and to restart components. The IR-gateway, for example, has extensive commands for checking the status of the transceiver hardware.

6.2 Code Libraries

We implemented a class-based hierarchy of composable widgets, loosely modeled on the Trestle window toolkit [21], to provide routine components such as iconic and text buttons, scrollbars, bitmaps, text labels, scrollable text areas, and dialog boxes. The PARCTAB's very small screen generally precludes overlapping of widgets, so our widgets do not need to do the clipping required by a conventional window system. This greatly simplified the implementation.

We also built the TabGroup programming interface to support concurrent use of multiple tabs by a single application. A group of tabs could act as a shared whiteboard or notepad, for example, displaying what was drawn on one tab to all the others in the group. With TabGroup, a program can wait for all pending

output to be delivered to all its tabs, synchronize on input or other events, and detect tabs that have stopped responding. Using a single process to control a group of tabs with standard interfaces provided by the tab programming library is often easier than running a separate process for each tab and having the processes communicate by application-specific RPC.

6.3 The Tshell and Tcl

Originally the only software available to support developers was the widget library. Developers used it much as they might use a language specific windowing toolkit like Xt[15] to write X-windows applications. As a result, they had to focus on low-level properties of the window system rather than on what they wanted to accomplish. Furthermore, for designers implementing simple user interfaces the turn around time of writing an application in a language like Modula-3 or C was too long. It became apparent that we had to provide fast prototyping capabilities and support the implementation of simple user interfaces at a higher level.

We created the Tshell [24], a PARCTAB-shell extended with a Tcl interpreter and a subset of Tk [23], thus providing both a scripting language that supports remote communication and a windowing toolkit. The choice of Tcl/Tk over other extension languages was based mostly on three reasons:

1. Tcl/Tk is widely used.
2. Tk provides a complete set of building blocks for creating graphical user interfaces. We could quickly select and implement a subset of widgets useful for the PARCTAB's small display size.
3. Tcl/Tk can be embedded into applications so that Tcl interpreters in different applications can exchange commands.

The design of Tab-Tk, the port of Tk to the Tab, focused on maintaining the natural look and feel of the Tk widgets while exploiting the small area of the Tab display as much as possible. We made several key observations and decisions during the port:

- The PARCTAB screen is too small to display multiple windows at the same time. Screen management therefore employs the same “one window at a time” philosophy as other tab applications.

- Because the Tab's screen area is limited, it makes extensive use of menus. They must be intuitive to use and have good response times.
- PARCTAB size and limited processing capabilities call for simplicity. The current implementation of the Tk toolkit for the Tab therefore provides a core widget set of buttons, labels, menus, text, entries, frames and toplevel-windows. We left out such features as the packer and canvas, a full-fledged drawing widget.

Tcl/Tk provides a high level language to rapidly prototype the graphical user interfaces for PARCTAB applications and a communication platform that allows programs to exchange commands with Tcl interpreters in other applications. In a matter of three months, members of our community created a wide range of new applications, including a context-based reminder system, a remote controller for a presentation manager, a pan/tilt camera controller, a remote editor for leaving notes on a workstation.

6.4 The MacTabbit system

Our colleagues at the Rank Xerox Research Centre (RXRC - formally called "EuroPARC") used a different approach to develop applications for the PARCTAB. The Apple Macintosh is the computer of choice at RXRC, and tab users there wanted to access Macintosh applications. MacTabbit does this by arranging for the PARCTAB to control a small portion of the Mac screen. It echoes updates in this region to the tab and sends pen and button events on the tab to the Macintosh.

Using graphical application builders on the Mac such a Hypercard, users can quickly prototype specialized Tab interfaces on the Mac Screen. When the interface works correctly, it takes but a few seconds to move it on to a tab. Furthermore, once the connection has been made to Hypercard, a user may select from a variety of Hypercard-based applications.

MacTabbit has provided an excellent prototyping environment for people unfamiliar with the conventional tab programming environment, and it has drawn in developers who would not normally have become involved. System performance was also good given the small tab screen. An extension of the MacTabbit mechanism caches commonly used image fragments in the tab, thus reducing bandwidth requirements and further improving performance. RXRC has used the MacTabbit mechanism to prototype many tab applications such as

Forget-me-not (see Section 7.1), an automatic diary and reminder system, and a media-space controller (see Section 7.2).

7 A CLASSIFICATION OF PARCTAB APPLICATIONS

Mobile Application Categories
Information Access
Communication
Computer Supported Collaboration
Remote Control
Local data/applications

Table 1 Mobile Application Categories

Three characteristics differentiate a tab and the kinds of applications that it supports from traditional personal computers:

1. Portability: very small form factor, low-weight
2. Communication: low-latency interaction between users and system
3. Context-sensitive operation

Our system represents context by a combination of factors: location, the presence of other mobile devices, and the presence of people. Context also includes time, nearby non-mobile machines and the state of the network file system. Traditional computer systems have had access to much of this information, but they have typically not made much use of it. Context can be used to adapt the user interface, criteria for extracting and presenting data, system configuration, and even the effects of commands. Although context may be used to present the options most likely to be chosen, a well-designed system would also allow a user access to the full range of choices on request.

Some of the applications we describe are available on small commercial PDAs whose size is comparable to that of a tab, but no PDA has the network infrastructure to support the full range of applications supported by the PARCTAB.

The combination of a wireless network and the use of context make this system unique. A summary of the application categories we have experimented with is given in Table 1 and described in some detail in the following sections.

7.1 Information Access

Access to information stored in our computer networks has become central to the way we conduct our work. The PARCTAB IR network has provided a mechanism to make information access independent of location. (Note that although all stored information is accessible from any networked workstation, people tend not to use someone else's machine.)

Each PARCTAB is linked to our local area network and so can retrieve any information available through it or through remote networks connected to it. For example, the commonly used weather program displays the current weather forecast (obtained from the Internet) and the local temperature and wind-speed (obtained from a weather station on the local network). PARCTAB users also have at their fingertips a dictionary, a thesaurus, a Unix file browser and a connection to the World Wide Web. The WWW protocol is a popular way to access information stored all over the Internet. Some care must be taken, however, to adapt the information retrieved to the small PARCTAB screen.

PARCTAB applications have also been integrated with existing desk-top applications. The PARCTAB calendar manager, for example, works with Sun's calendar manager ("cm"), already in use. An update to a user's calendar either on a workstation or on a PARCTAB will enable the data to be viewed on both systems.

The tab location-based file browser shows how context can be used to filter information. Instead of presenting the complete file system hierarchy, it shows only files whose information is relevant to the particular room it is in. Such a mechanism can be used to provide a guided tour for a visitor or to provide information that is relevant to a location, such as the booking procedure associated with a conference room.

More complex uses of context can be seen in tools built at RXRC such as Forget-me-not [19, 22, 20, 18, 17]. This application provides a tab user with an automatic biography of their life by remembering for each day details such as: where the person went in the office, whom they met, the documents they edited or printed, and any phone calls that were made or received. The motivation

behind this work is to provide an aid to our fallible human memories, a so called memory-prosthesis. The application operates by providing an iconic interface that allows a user to search and filter the biography for a particular event. For example, suppose a forgetful user were trying to find the name of a document that she was editing when Mike came into the room a short while after the seminar last week. The filter would be set up to show documents in use when Mike was around, on the day of the seminar. As we seem to waste a great deal of our lives searching for things we have either misplaced or information we have forgotten, Forget-me-not has the potential to help us work more effectively.

7.2 Communication

Electronic mail has long been a popular communication tool for computer users. Mobile access further enhances e-mail by increasing its availability.

Group meetings often account for a large amount of our work time, and so electronic mail has been an important application for the PARCTAB. Access to e-mail during meetings seems to have satisfied a genuine need.

The PARCTAB e-mail application could be extended to use context to generate filters for displaying messages or notifying users of incoming mail. For example, all messages might be delivered while a user is alone, but only urgent ones would be delivered during a conference. In related work [12] a query language has been used to filter incoming mail.

Locator and Pager Operation

The PARCTAB system inherently provides a locator system, assuming that the person who needs to be found is carrying a PARCTAB. In an office, people can use context to decide whether to disturb a colleague, once they have been located [37]. For example, a person is more likely to welcome interruptions alone in their office than while in a meeting. With the PARCTAB system, a person may be paged unconditionally, or the importance of the page can be assessed in association with the recipient's context, so that the message will be either delivered or delayed until the context is more favorable.

Media Applications

Another RXRC application is the “Communicator”, a context-sensitive media-space controller. A description of the original media-space concept is given by Buxton [4] — a video-conferencing mechanism based on an analog-switch controlled by workstations, allowing users to establish video connections to various places in an appropriately wired building. The tab has been used to enhance this facility through an application that will suggest the easiest way to communicate with the person you wish to contact, and then help establish the connection. Knowledge of where the recipient is situated is known to the system because they are carrying a tab, the calling party only needs to know their name. If a media-space terminal is not available, the application might suggest the best alternative: a phone number, let you know they are actually next door, or offer to send an e-mail note from the tab screen. More recent work at the University of Toronto has taken this work further and combined Ubiquitous Computing with video in a reactive environment [3].

An application that pushes the PARCTAB’s communication abilities to their limits is media windowing. An otherwise unused IR channel can transmit one low-resolution frame of slow-scan video in about 1.5 seconds. These images are very grainy because of the coarse resolution of the PARCTAB screen and the limited bandwidth of the link. Nevertheless people are remarkably good at recognizing faces and scenes, and the images are still useful. Future systems with improved screens and higher bandwidth links could provide applications for remote monitoring and mobile communication using sound and video.

7.3 Computer Supported Collaboration

People often gather with a common goal or interest, perhaps at a lecture, or else to arrive at a common decision. Because the PARCTAB is small, it can easily be used in these collaborative situations.

Group Pointing and Annotation

A PARCTAB used as a pointing device operates much like a mouse. However, a PARCTAB can connect to different computers depending on its location.

Many PARCTABS can also connect to the same computer. Consider, for example, the case in which a lecture is presented using a large electronic display such as a Liveboard (see 2.3). Each tab in the audience can control a different

pointer on the display. We have built a remote display pointer using the PARCTAB screen as both a relative and absolute positioning tool: the user controls the location and motion of the pointer by moving a finger over the PARCTAB's touch surface¹.

Voting

The PARCTAB can also be used when members of a group wish to arrive at a consensus, perhaps anonymously. Even if anonymity is not important, simultaneous voting can collect data that is unbiased by the voting process. If people vote in sequence, earlier viewpoints inevitably bias later ones.

We have built a voting application called Arbitron for the PARCTAB system. It has proved particularly interesting in the context of presentations. Audience members with PARCTABS vote on the quality and pace of the material being covered by a presenter. The votes are collected anonymously and displayed on the Liveboard. The board is visible to both the audience and the presenter; thus everyone knows whether their colleagues are as bored or entranced as they are. Without the PARCTAB listeners would have to interrupt the presentation to ask the speaker to speed up, slow down, or move to another point.

Multi-tab Virtual Paper

Tabdraw is a multi-tab application that allows the tab screen to be used as if it were a piece of scrap paper. Each PARCTAB participating in the application owns a piece of virtual paper and can draw on it. The participants also have the option of seeing the drawings of their colleagues by superimposing them on their own work. This scheme ensures that users "own" the line segments they draw; no one else can erase them. As a result, many users can work together in a coordinated fashion without impairing fair participation.

The shared drawing is generally defined by the room that people are in. A group in one room will automatically obtain a separate drawing surface from that in another room. Alternatively, a group might arrange to share a drawing regardless of location.

¹A tab-based remote pointing and annotation tool was demonstrated as part of the Xerox exhibit at Expo '92 in Seville

7.4 Remote Control

Television and stereo system remote-controls have popularized the notion of control at a distance. In fact so many pieces of consumer electronics have such controllers that one can now buy universal remote controls that control many devices at the same time. A PARCTAB can also act as a universal controller. Furthermore, it can command applications that traditionally take their input from a keyboard or a mouse.

Since a tab can display arbitrary data, the controls available to a user can be changed depending on context. (Commercial universal remote controllers, in contrast, tend to need a large array of buttons.) Enabling the remote control application in an office may trigger a tab to provide a control panel that adjusts lighting and temperature, whereas in a conference room the interface might be biased toward presentation tools.

Program Controllers

During our experiments with group drawing and pointing tools it became clear that a PARCTAB has some interesting control possibilities as a drawing interface for a drawing program. It can make additional commands available without cluttering the main screen, and it can also provide a more powerful set of commands than was available in the original program by providing a single button that controls a sequence of low-level drawing primitives. If a program is already intended for remote use and has a network interface, controlling it with a PARCTAB is very easy.

X10 Remote Control

Another Ubiquitous Computing project at Xerox PARC, the Responsive Environment Project [9], has been exploring how environmental control can save energy during the day-to-day operation of a building. The project had created servers that control power outlets through a commercial system called X10 [2].

Because the servers controlling appliances in part of the building being studied by the Responsive Environment project were already connected to the local area network, it was a simple matter to build PARCTAB applications to control them.

7.5 Local Operation

The PARCTAB is near one extreme of a spectrum of possible devices ranging from the remote terminal (devoid of function without its connection to the network) to the standalone computer (capable of many operations without any communication links). The latest revision of the tab hardware has 128K of on-board memory, so that data and programs can be downloaded through the IR link and executed in a stand-alone mode. Operating the tab in this way frees a user from the IR network, but of course severely limits the tab's functionality.

The storage capacity of a mobile device will probably always be small compared to the expectations of its user. Consequently applications must take care to download only the most relevant information. For example, if a user has unread electronic mail at the end of a work day, the system might transfer the messages to the PARCTAB so that they could be read in transit or at home. (Currently, all downloading of information and programs occurs under the user's control.)

8 EXPERIENCES WITH THE PARCTAB SYSTEM

The PARCTAB system has been in use since March 1993 and now serves a small community of users. We have made a number of useful observations during this period and have begun to understand its successes and failures.

8.1 The Experimental Network at PARC

PARC was a convenient test site for the PARCTAB system because installation was very easy. Before the project began every office already contained a workstation connected by an ethernet. The hallways and common areas also had access to nearby workstations. It was easy to install a communication cell in an office by using velcro to attach a transceiver to the ceiling and then to run phone cable down a wall into a junction box. The junction box usually rests on the floor under a desk and has a power cable, and connects to the RS232 port of the workstation. Typically, the installation takes about 15 minutes.

Some users also installed cells in their homes. They already had ISDN lines, which connect a home ethernet to the office network, and so a transceiver

connected to a workstation at home was effectively tied to the PARCTAB infrastructure.

The first PARCTAB system released in March '93 consisted of 20 users and 25 cells. The experience gained in this time enabled a second release in April '94. The latter system was somewhat larger with a community of about 41 users and 50 cells. It included many improvements that enhanced the performance of the communication channel and the tabs' perceived reliability.

For example, the original system relied on a central name-and-maintain service (see Section 5.2) to route packets to tabs; when the service was unavailable the PARCTAB system could not function. The new release has a distributed name service that uses a network multicast mechanism to determine the address of system components.

We discovered in the first release there were problems caused by high utilization of the infrared network. High loads cause three problems: infrared packets are more likely to be corrupted; transmit buffers in the transceiver overflow, causing packets to be dropped; and the corrupted and dropped packets caused more retransmissions, increasing the load. The high load exposed bugs in the system design and implementation such as race conditions and badly-tuned retransmission policies.

To improve user's confidence in the system, we had to increase its reliability and availability. This involved not only fixing bugs but also mundane improvements such as a low-battery indicator for tabs. System components also needed mechanisms for self monitoring. All the PARCTAB system processes now have control panels designed to provide information in the event of a failure. We have also put new mechanisms in place to monitor and maintain the IR-gateway and the agent processes.

8.2 Infrared Interference

The PARCTAB could not be used effectively in several rooms in our building because of IR noise due to fluorescent lamps controlled by electronic ballasts. This is a waste of a unique form of communication bandwidth. Unfortunately, electronic ballasts are slowly replacing the older magnetic ballasts because they are more energy efficient. We found a considerable variation in interference levels from lamps made by different manufacturers. Some produce acceptable

levels of IR, and it would be useful if lamp manufacturers were required to adhere to a maximum limit for IR emissions.

Positioning of a room transceiver is also important. Installers should avoid direct sunlight, that can change position throughout a day (and during the year), and proximity to fluorescent lamps and to obstructions on the ceiling. Transceivers in adjacent cells should be positioned carefully so that their signals do not pass through doorway or interior windows and cause interference.

8.3 Usage Data Measured from the PARCTAB System

Part of the benefit of building a real system has been the opportunity to study how a versatile personal information-terminal might be used in advance of a commercial system. We studied the 1994 release of the tab system for three months to determine its use characteristics. The participants all consented to automatic logging of system events.

We began recording two weeks after system deployment so that users could familiarize themselves with the PARCTAB. To limit the data to a manageable quantity, we logged only the following events: Interactive, Switch, Idle, and Missing². Interactive occurs when a user powers up a tab, Switch occurs when a user switches to a new application, Idle is generated when a tab has not been used for 4 minutes, and Missing is a timeout event generated by the system when the infrared network cannot detect a particular tab. Each event was recorded along with a timestamp and cell location. In addition, there were two questionnaires given out to our users, one at the outset of the tab use study and one at the close. This provided contextual information, and information to interpret the logging data.

Which Applications were Popular?

The switch events can be used to determine the relative popularity of the various PARCTAB applications. Figure 11 shows the percentage of invocations accounted for by each application. Four were distinctly more popular than the

²During the 3 month study some system processes died and were restarted causing some events not to be logged. This results in minor, but conservative, inaccuracies in the reported statistics.

rest: e-mail, weather, file browser, and the loader. Possible implications of these results are discussed in Section 9.

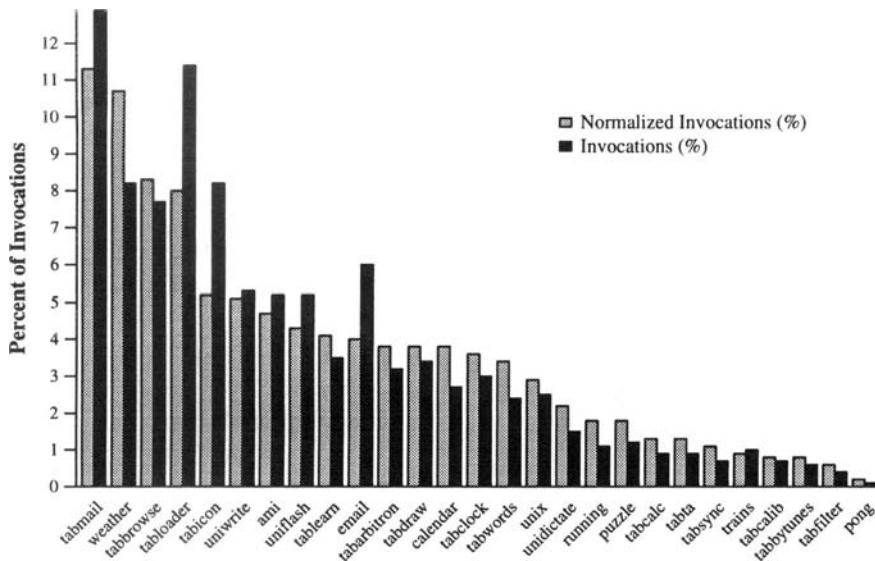


Figure 11 Histogram showing the number of invocations for each application (not including the shell or tshell) expressed as a percentage of the total invocations of these applications during the test period. Normalized results only count one invocation per day per user to remove distortions that might arise when users experiment with an application several times during a brief period. Applications that might normally be invoked several times a day suffer under this measure.

How Long were Applications in Use?

Another way of looking at application popularity is to consider how long each application was in use (see Figure 12). It should be noted that the total application interaction time is 4871 minutes over 3 months (13 weeks) for 41 users. This amounts to only 119 minutes/user or about 1.8 minutes/user/day (65 days, excluding weekends). From our logs the total number of application switches for all tabs throughout the study was 2996 and therefore the average interaction time was about 97 seconds.

The application popularity ranking is somewhat different from Figure 11. The e-mailer, unistroke test and learn program, unistroke notetaker, file browser, and the loader are the most long-lived applications. The weather program falls to 8th place (perhaps because it only imparts a small amount of information at any one time). Meanwhile the note-taker moves up to 3rd place – not surprising, as taking notes is by its nature a time-consuming activity. It is interesting to observe that reading e-mail, browsing system files, and loading data turn out to be the most used in both measurements.

This use pattern differed from the participants own expectations of use. Although they expected to read e-mail, (four of the participants did not use e-mail on the tab at all, due to incompatible mail systems), over half commented that they expected to use the tab primarily as a calendar. It is also worth noting that according to user reports the e-mail program was used to read e-mail much more than to send e-mail using Unistrokes. The Unistroke test and learn programs appear in the ranking even though they are typically not activated very often; users may spend a block of time running them when first acquiring the skill.

Graph 13 shows the percentage of application interactions that last less than a given time. We have removed interactions of less than 10 seconds because users often turn a tab on and then off immediately to confirm that it is working normally. From this graph we can see that 50% of interactions last less than 100 seconds (1.7 mins), 75% less than 230 seconds (3.8 mins) and 90% less than 500 seconds (8.3 mins). This supports our notion of the tab as a device for “casual” interactions.

Figure 14 shows what fraction of users had their tabs turned on for various total periods of time. The study group can be roughly divided into three user types. 7% (3 people) used the tab for 360-480 minutes during the test (6.4 minutes/day). 15% (6 people) used it for 144-360 minutes (3.9 minutes/day) and 78% (32 people) used it for less than 144 minutes in total (1.1 minutes/day). The average use time for the majority was very small, implying their interactions were generally very brief.

Who Used the PARCTAB, How Long and Where?

Figure 15 shows interaction time for each user, subdivided according to location: in their own office (black); in a common area such as a conference room, tea area or seminar room (grey); or in a hall or another person’s office (white). Only 3 people used a tab primarily (for more than 50% of their total interaction time)

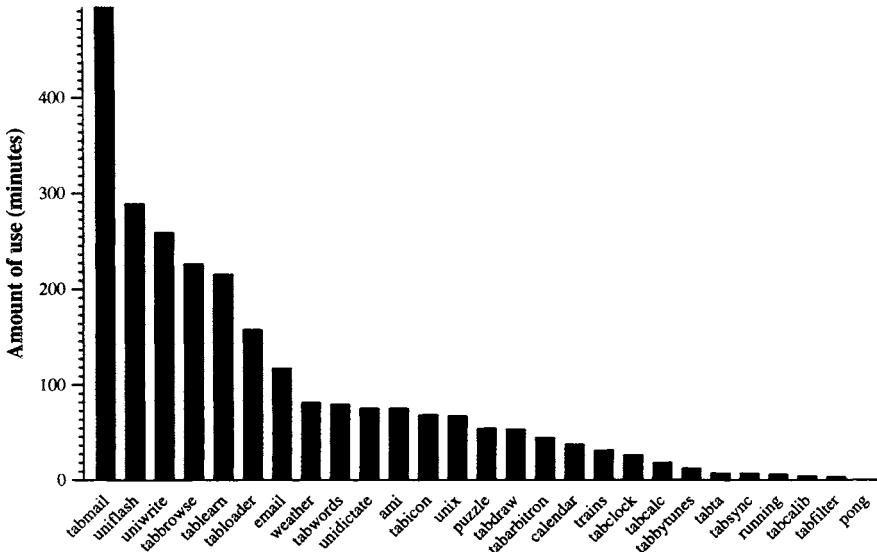


Figure 12 Histogram showing the total interaction time by users for each application in the tab system during the 3 month test period (not-including the shell, 1273 minutes, and the tshell, 1081 minutes).

in somebody else's office. Approximately 61% (25 people) of our community used the tab primarily in their own rooms, and 27% (11 people) used it primarily in a common area. Interestingly enough, for each pattern of use the preference was quite clear.

By pooling the results of Figure 15 we can determine that people used tabs in their own offices 57% of the time, in a common area 32% of the time , and in another office 11% of the time (see Figure 16). 7% of own-office interactions are in the presence of other tabs. 90% of common area interactions and 85% of other-office interactions are also in this category.

The multiple-user applications, group drawing and remote pointing, were not available for the duration of the use study. Group applications like this would have generated a much higher network-load in the common areas, but are likely uses of a ubiquitous mobile device.

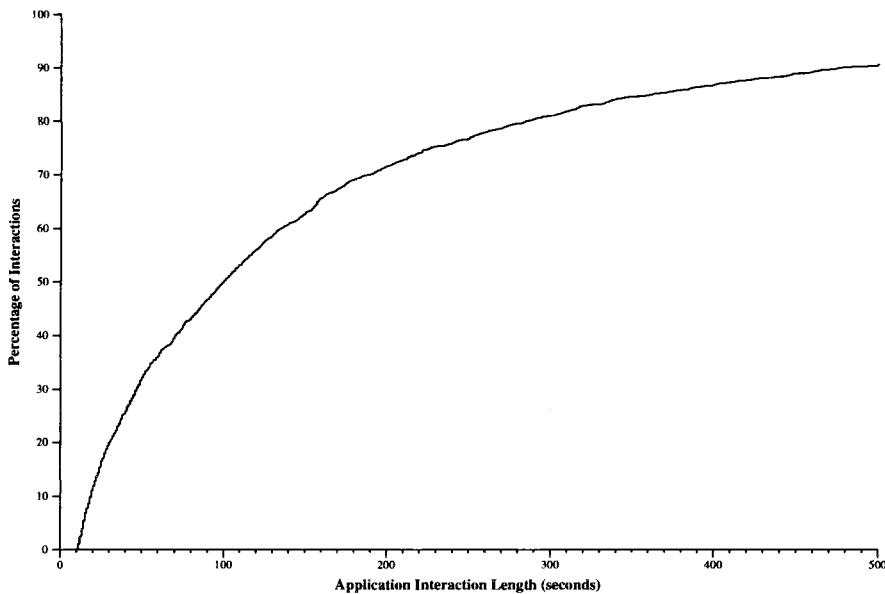


Figure 13 Graph showing the percentage of application interactions that were under a given time during the test period.

Figure 15 shows that there is not a typical use pattern among the study group. Our questionnaires showed that there were as many different expectations of the tab system as there were participants in the study. For example, researchers developing applications on the tab that expected to use the tab a great deal did not necessarily have the largest interaction times, even though they had to use the tab for their daily work. In contrast, some researchers who did not expect to use the tab found that visitor demonstrations of the device added significantly to their total usage time.

These results are important for overall system design because multiple tabs interacting in the same area have a strong impact on the available bandwidth. The PARCTAB system needs to be able to handle a usage pattern in which at least 42% of all interactions occur with multiple tabs present.

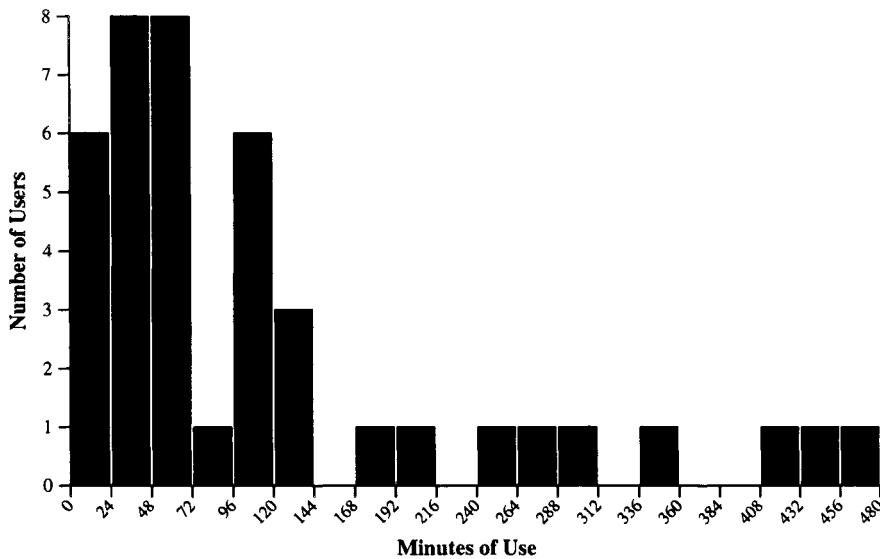


Figure 14 Histogram showing the number of users against their total interaction time divided into 20 equal divisions.

8.4 Discussion

Although the previous graphs give an indication of the way the tab was used it is important to acknowledge the limitations of this study in representing the use of the tab as a consumer item. First, the user group was too small for statistically significant results. Second, the system was still under development and the applications were not fully supported. Furthermore, participants in the study were not customers but rather laboratory staff using the tab as a prototype. It was up to them to invent ways to use the tab, develop new applications and create ways to incorporate the tab into established work patterns. As a result, we must qualify the numbers with anecdotal evidence and further discussion of the ways people used the tab. Some of these remarks are listed below:

Rich Gold: does not see any value in using a tab in his own office because a powerful workstation is at hand.

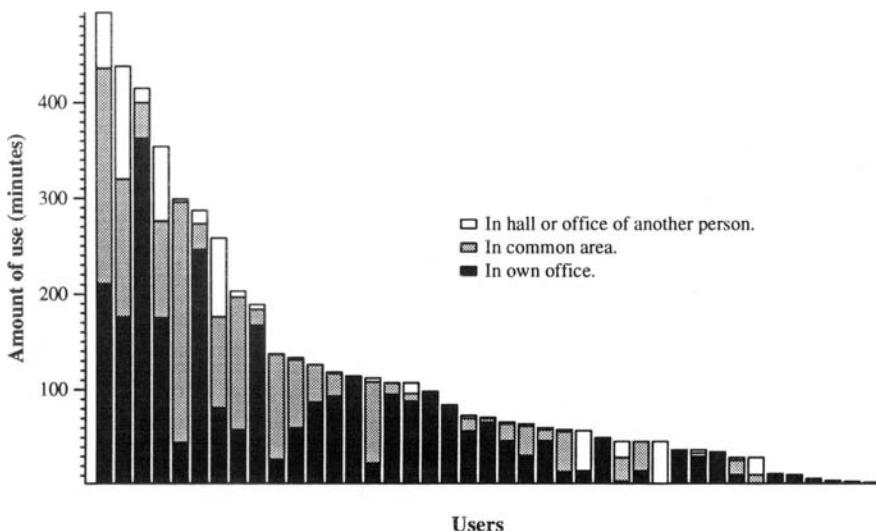


Figure 15 Histogram showing the total interaction time for each user in seconds split between three location types: a user's own office, a common area, or a hall or another person's office.

John Ellis: prefers to use the tab in his own office to read his e-mail so that he does not have to rearrange the windows on his workstation screen.

Dan Swinehart: found the tab system had a long response time, but found that the tab system was faster than Mosaic for finding the definition of a word.

Helen Davis: has used the email application and Unistrokes to take notes during seminars and then mailed them to herself.

A number of people found the PARCTAB too heavy or awkward to wear.

Two women tab users (Karin Petersen and Nancy Freige) remarked that the design of the belt clip was oriented towards a particular clothing style. For example, not all outfits include belts, and furthermore not all belts work well with clip on devices. Doug Terry also found the tab clip inadequate for his use. Instead he used a small zippered nylon and (infrared transparent) fishnet pouch to hold a tab so that it could be attached to his belt and continue to report his location.

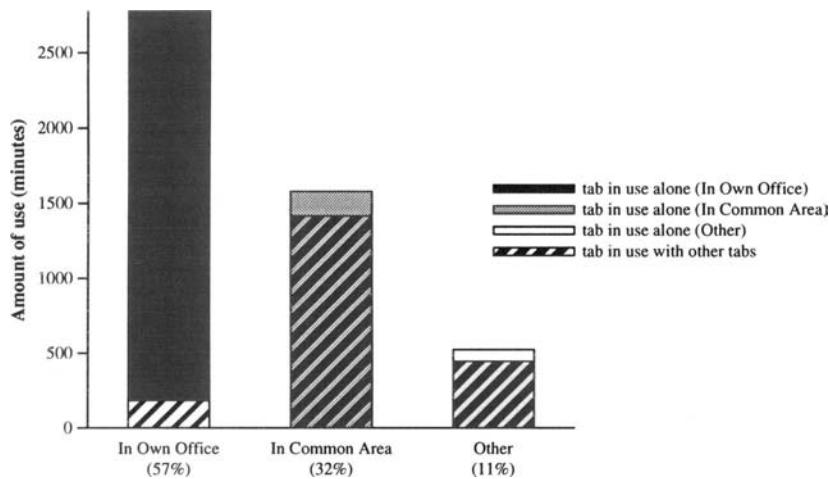


Figure 16 Histogram showing the total interaction time by all users for each of the three general areas: a user's own office, a common area, a hall or another person's office.

A researcher who preferred to remain anonymous commented on the difficulties of building new applications in Modula-3: 'I don't want to say anything against Modula-3 but if I have to learn a new language at the same time as trying to program a new [computer] I may not get much done.'

The ease of reading text on the small screen surprised most of the participants in the use study. At the beginning of the study we found almost 1/2 of the participants had commented that because of the low resolution of the screen they did not intend to read longer files.

As the list above indicates, it is difficult to suggest a 'typical' use of the PARCTAB. The PARCTAB system was an experiment that many people volunteered to participate in. It was shaped by their own ideas, needs and contributions. A direct consequence of building a system that can be used by a community is that it is possible to gain understanding of the real problems (see Section 9), issues to be addressed, and activities that need to be supported.

8.5 Research at other Sites

To gain more general experience we gave the tab system (including tabs, transceivers, and software) to a number of other research departments. The largest of these sites was the Rank Xerox Research Centre (Cambridge, England) with 12 transceivers and 10 PARCTABS. Flinders University (Adelaide, Australia) University of Washington, University of Toronto and Olivetti Research Ltd (Cambridge, England) also received small numbers of PARCTAB system components for their own research. RXRC produced a number of applications (see Section 7), and the University of Toronto now uses tabs to control the equipment in its “telepresence” room.

9 CONCLUSION

The PARCTAB system enables a unique set of applications that have used communication and context to enhance their operation. By designing a system and deploying it, we were able to gain some insight into the benefits and problems faced by mobile systems. The following sections draw some conclusions.

9.1 Design Perspective

The PARCTAB architecture depends on small-cell wireless communication. It thus combines portability with information about context. A downside of this approach was that the PARCTAB was not very useful out of contact with the network. Some of our users were dissatisfied that the tab had only very limited use when disconnected from the network. Perhaps the real value of a PDA comes from both connected and disconnected operation. One without the other leaves them dissatisfied.

Our system design was based on a distributed architecture containing many components. Although each component was relatively simple the complete system presented a level of complexity that made it difficult to debug. We learned to remove as many points of failure as possible to allow users to understand what was going on.

9.2 Bandwidth Limitations

One of our early design assumptions was that a 19.2k baud link was adequate for building the PARCTAB system. If users do not often share cells or do not, on average, operate their PARCTABS at the same time, the system can usually respond within 1 or 2 seconds. In meetings, however, these assumptions seldom hold true. Users tend to operate tabs at the beginning of meetings, at short breaks and perhaps when they are bored, resulting in synchronized use and poor performance.

We now recognize that such systems have to be engineered to deal with the maximum congestion that can result from the maximum number of mobile units in a room. Figures based on average usage patterns do not justify cutting corners.

9.3 Characteristics of User Generated Traffic

Another early design assumption was that applications would have repeating usage patterns of the form 1) event 2) screen update 3) delay, with the delay caused by the time it takes a user to read the screen. However the Unistroke interface changed this pattern. A Unistroke writer can make several strokes per second. In combination with other Unistroke traffic, this can generate a load greater than the IR network was designed to handle. As a result, we have begun work on improving the partitioning of applications between the PARCTAB and the rest of the system. The Unistroke recognizer has recently been ported to the PARCTAB firmware, allowing us to send packets of characters rather than a sequence of stylus positions. This approach uses significantly less bandwidth in both directions and will be included in a future PARCTAB release. Display keyboards could work the same way.

The largest impediment for people using Unistrokes was the slow response-time of the system when displaying a character after each stroke of the stylus. Many of the participants who had learnt Unistrokes, claimed to be able to write faster than the system could keep up. All of those who learnt Unistrokes felt that it was a superior form of text input.

9.4 Factors Affecting Acceptance

Whether or not a tab is adopted in the workplace turns out to depend on many factors: among them size, appearance, convenience, peer pressure, application types, and critical mass of applications. People, in general, have well established work habits that are a barrier to learning a new system. Applications that solve a real problem are however compelling, and a diversity of application type makes the tab a solution to many problems.

It has become clear that changing the nature of a single characteristic can tip the balance between acceptance and rejection of the device e.g., the design of a suitable belt/clothes clip. Small changes in design can have large effects and this makes it difficult to make predictions. Building a system intended for use is the only way to really find out.

We have discovered how difficult it can be to persuade people to make changes to their daily routine in order use a device like the PARCTAB. Furthermore, an individual's style of dress has a significant impact on whether a tab can be easily attached and worn like a pager. One user's tab fell off a belt in a parking lot, damaging the device, and making the user less willing to carry it.

Many people expressed an interest in a system that could be used both inside and outside the building, and if this had been the case, they might have adopted it in more readily. It is clear that a conventional radio broadcast scheme would allow greater mobility, but at the expense of bandwidth and the lack of context. A more comprehensive system might use a combination of nano-cellular communications for in-building use and a packet-radio scheme for outside use.

There were two important aspects of tab use in the CSL study that were demonstrated by the logging data. First, the brief period that applications were used (50% were under 100 seconds), and second, the generally infrequent usage-pattern.

Given that the typical behavior is of short user-interaction-times, we might be able to better support a user's needs by supplying more casual interfaces that summarize data on the tab top-level screen (e.g., time, weather, amount of mail to read etc), enabling a user to retrieve information at a glance. Perhaps icons that change state to represent the activity of their underlying applications would address this issue, replacing the desktop metaphor currently in use by a wrist-watch metaphor.

The total interaction-time combined for all tabs was not very large. This is as much a reflection on the context of use as any inherent difficulties with the tab. The researchers and support staff participating in this experiment work in a computer-saturated environment. They are never far from a workstation, and apart from attending meetings, their work practices typically do not rely on being mobile (see Figure 16, percentage of time spent in an office). This suggests that further work for integrating the tab into the office environment needs to be considered, for example, using the tab as another computer monitor. But it also suggests that in a manufacturing environment, or a hospital, tabs might support established mobile work-practices.

It should also be noted that the tab system is a prototype and is not supported to the same extent as an established product (e.g., no user manuals). In this case study, the users are participating in the development and therefore it is more appropriate to think of them as participants rather than users.

In the near future, a device capable of performing the PARCTAB's functions could be made about one third the thickness and one third the weight of the current version (3-4 mm thick and perhaps 70 grams). This may further encourage its use.

9.5 Application Development

We set out from the start to encourage the user community to become involved in writing applications. The original Modula-3 programming environment, although a state-of-the-art approach to building systems, was unfamiliar to many of the users. In some cases learning it was too much trouble for producing a relatively simple application. In addition, the compiler created large binaries (often greater than 3MB for each application), imposing a significant load on machine resources when many applications were active. Making it possible to write applications in Tcl/Tk and Hypercard was significant in broadening the interest of application developers.

9.6 Importance of User Interface

An innovative part of building the PARCTAB system has been the design of user interfaces that are suited to a small screen e.g., elision and Unistrokes. The latter is a powerful technique that can be used with pen-based computers of any size.

The design of the PARCTAB packaging was clearly successful. In particular, our users liked a design that was adapted to either right or left handed people. It was also clear that three physical buttons usually provided an unambiguous mode of use. Although it was tempting to design the user interface with more buttons, enforced simplicity has turned out to be a bonus.

9.7 Popular Applications

Our system provided many programs that could be used in the work environment. It is interesting to consider the four most commonly invoked. In first place was the electronic mail reader, providing access to e-mail that is normally only available at a workstation. Perhaps this is not surprising given that the study was carried out at a computer-science research laboratory. However, electronic mail is becoming more popular in the business community and this result might be significant in predicting a future market.

The weather program scored second highest. It is possible this shows an inherent fascination with weather, or the program may just be good demo-ware. We hope that this indicates a deeper interest in information that is up-to-date and easily accessed. In that case, a mobile interface to the World Wide Web or other information services might prove compelling.

In third place was the file browser, providing access to text and command files stored in the Unix Network Filing System. Since the entire study group works almost entirely with electronic documents which are available on-line, this is a likely result. Finally, in fourth place was the tab loader, which allows users to store information in the tab's local memory and use it outside the infrared network. It is not surprising it has also been popular.

Although the unistroke notetaker was not invoked very often, it accounted for a significant chunk of total tab usage. It is possible that note-taking could become a heavily-used application, especially if local processing of unistrokes yields the expected improvements in performance.

Of the remaining applications there is one result that appears to be out of place. The PARCTAB calendar/diary appeared mid-way through both the popularity and runtime results. In the initial questionnaire all but two of the users had stated that they intended to use the calendar manager regularly. Although there was some difficulty with the compatibility of electronic calendars in use, 80% of the participants could use the appropriate calendar manager on the

tab. Given that office environments have schedules that involve many meetings and numerous visitors, this result seems low. We have found, however, that users often have traditional solutions to this problem in place (e.g., pocket-book diaries). New solutions that are as good, or only marginally better (such as tab access to an on-line calendar) are not easily adopted.

9.8 System Benefits

One important contribution of the PARCTAB system has been the experimental infrastructure that allows users to prototype new application ideas. The system has been something of a catalyst in generating new ideas in the area of Ubiquitous Computing and has inspired novel applications. Because the infrastructure is easily assembled and can be exported to other test sites, we have also had the benefit of stimulating other research.

9.9 Future Work

Many system issues still need to be explored, for example, how to resolve conflicts during disconnected operation when related information has changed in both the mobile and the fixed part of the system [7, 35, 34]. Another area that needs exploring is how to partition system functionality across a wireless link with the aim of reducing communication latency. An extension of the existing work that would allow us to make better use of system context, is the design of a mechanism for the precise location of objects in a building. Ubiquitous computing could take advantage of precise location information: knowing which screen a user is currently looking at, for example, is invaluable when deciding how to present urgent information. Finally, the whole area of miniature user-interface research deserves further study and has the potential for many more innovations.

Ubiquitous computing has been the main inspiration for the PARCTAB project. The use of this system has allowed us to study context-sensitive applications. These prototype applications have demonstrated the potential for innovation in this area. In the future we expect to continue to carry out research with the PARCTAB, and also other hardware and software that will help define the future of ubiquitous computing. Our experience with the PARCTAB systems look very promising and brings us a step closer to realizing that future.

Acknowledgements

We wish to thank the many summer interns that have contributed to this project and made it fun to work on: Michael Tso, Nina Bhatti, Angie Hinrichs, David Maltz, Maria Okasaki, and George Fitzmaurice. We also wish to thank: Jennifer Collins and Sonos Models for facilitating the PARCTAB packaging; Bill Buxton (University of Toronto) for his advice concerning UI design; Terri Watson, Berry Kercheval and Ron Frederick for developing novel applications; Natalie Jeremijenko for collecting and processing results from the tab usage experiment; Olivetti Research Ltd (ORL) and Andy Hopper for collaborating with us while developing the communication hardware; Brian Bershad (University of Washington), Craig Mudge(Flinders) and Mike Flynn for their keen advice and collaboration; and Wayt Gibbs and Paul Wallich for editing this paper. Finally, we wish to thank and acknowledge Mik Lamming for his original contributions and support during the lifetime of the project.

REFERENCES

- [1] Norman Adams, Rich Gold, Bill N. Schilit, Michael Tso, and Roy Want. An infrared network for mobile computers. In *Proceedings USENIX Symposium on Mobile & Location-independent Computing*, pages 41–52. USENIX Association, August 1993.
- [2] Jeff Bachiochi. X-10 interfacing with plix. *Circuit Cellular INK*, pages 74–79, Oct/Nov. 1992.
- [3] William Buxton. Living in augmented reality: Ubiquitous media and reactive environments. *To appear in CACM*, 1995.
- [4] William Buxton and Tom Moran. *EuroPARC's Integrated interactive intermedia facility (iiif): early experiences*. North-Holland, 1990.
- [5] George Calhoun. *Digital Cellular Radio*. Artech House Inc, 1988.
- [6] Alan Demers, Scott Elrod, Christopher Kantarjiev, and Edward Richley. A nano-cellular local area network using near-field rf coupling. In *Proceedings of Virginia Tech's Fourth Symposium on Wireless Personal Communications*, pages 10.1–10.16, June 1994.
- [7] Alan Demers, Karin Petersen, Michael Spreitzer, Douglas Terry, Marvin M. Theimer, and Brent Welch. The bayou architecture: Support for data shar-

ing among mobile users. In *Proceedings Workshop on Mobile Computing Systems and Applications*. IEEE, December 1994.

- [8] Scott Elrod, Richard Bruce, Rich Gold, David Goldberg, Frank Halasz, William Janssen, David Lee, Kim McCall, Elin Pedersen, Ken Pier, John Tang, and Brent Welch. Liveboard: A large interactive display supporting group meetings, presentations and remote collaboration. In *Proc. of the Conference on Computer Human Interaction (CHI)*, pages 599–607, May 1992.
- [9] Scott Elrod, Gene Hall, Rick Costanza, Michael Dixon, and Jim des Rivières. Responsive office environments. *CACM*, 36(7):84–85, July 1993. In Special Issue, Computer-Augmented Environments.
- [10] Neil Fishman and Murray S. Mazer. Experience in deploying an active badge system. In *Proc. of IEEE Globecom Workshop on Networking of Personal Communications Applications*, December 1992.
- [11] Jim Fulton and Chris Kent Kantarjiev. An update on low bandwidth X (LBX). Technical Report CSL-93-2, Xerox Palo Alto Research Center, February 1993.
- [12] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *CACM*, 35(12):61–70, Dec 1992.
- [13] David Goldberg and Cate Richardson. Touch typing with a stylus. In *Proc. Conference on Human Factors in Computing Systems (INTERCHI)*, pages 80–87. ACM/SigCHI, Apr 1993.
- [14] Andy Harter and Andy Hopper. A distributed location system for the active office. *IEEE Network*, pages 62–70, January/February 1994.
- [15] Oliver Jones. *Introduction to the X Window System*. Prentice Hall, 1989.
- [16] Christopher Kent Kantarjiev, Alan Demers, Robert T. Krivacic Ron Frederick, and Mark Weiser. Experiences with X in a wireless environment. In *Proceedings USENIX Symposium on Mobile & Location-independent Computing*, pages 117–128. USENIX Association, August 1993.
- [17] Mik Lamming. Towards future personalised information environments. In *FRIEND21 Symposium on Next Generation Human Interfaces*, Tokyo Japan, 1994. Also available as RXRC TR 94-104, 61 Regent St., Cambridge, UK.

- [18] Mik Lamming, P. Brown, Kathy Carter, Marge Eldridge, Mike Flynn, Gifford Louie, Peter Robinson, and Abi Sellen. The design of a human memory prosthesis. *Computer Journal*, 37(3):153–163, 1994.
- [19] Mik Lamming and Mike Flynn. Forget-me-not: intimate computing in support of human memory. In *FRIEND21 Symposium on Next Generation Human Interfaces*, Tokyo Japan, 1994. Also available as RXRC TR 94-103, 61 Regent St., Cambridge, UK.
- [20] Robert Langreth. Total recall. *Popular Science*, pages 46–82, February 1995.
- [21] Greg Nelson. *System Programming with Modula-3*. Series in Innovative Technology. Prentice Hall, 1991.
- [22] William Newman and Mik Lamming. *Interactive System Design*. Addison-Wesley, 1995.
- [23] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.
- [24] Karin Petersen. Tcl/tk for a personal digital assistant. In *Proceedings of the USENIX Symposium on Very High Level Languages (VHLL)*, pages 41–56, Santa Fe, New Mexico, October 26-28 1994. USENIX Association.
- [25] Ken Pier and James A. Landay. Issues for location-independent interfaces. In *Xerox Parc Blue&White P92-00159*, December 1992.
- [26] Bill N. Schilit, Norman Adams, Rich Gold, Michael Tso, and Roy Want. The PARCTAB mobile computing system. In *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, pages 34–39. IEEE, October 1993.
- [27] Bill N. Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Proceedings Workshop on Mobile Computing Systems and Applications*. IEEE, December 1994.
- [28] Bill N. Schilit and Marvin M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, pages 22–32, September/October 1994.
- [29] Bill N. Schilit, Marvin M. Theimer, and Brent B. Welch. Customizing mobile application. In *Proceedings USENIX Symposium on Mobile & Location-Independent Computing*, pages 129–138. USENIX Association, August 1993.

- [30] Mike Spreitzer and Marvin Theimer. Providing location information in a ubiquitous computing environment. In *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*, pages 270–283, Asheville, NC, December 1993. SIGOPS, ACM.
- [31] Mike Spreitzer and Marvin Theimer. Scalable, secure, mobile computing with location information. *CACM*, 36(7):27, July 1993. In Special Issue, Computer-Augmented Environments.
- [32] Mike Spreitzer and Marvin Theimer. Architectural considerations for scalable, secure, mobile computing with location information. In *Proc. 14th Intl. Conf. on Distributed Computing Systems*, pages 29–38. IEEE, June 1994.
- [33] Andrew Tanenbaum. *Computer Networks*. Prentice Hall, 1981.
- [34] Douglas Terry, Alan Demers, Karin Petersen, Michael Spreitzer, Marvin M. Theimer, and Brent Welch. Session guarantees for weakly-consistent replicated data. In *Proc. 3rd International Conference on Parallel and Distributed Information Systems*, pages 140–149, September 1994.
- [35] Marvin M. Theimer, Alan Demers, Karin Petersen, Michael Spreitzer, Douglas Terry, and Brent Welch. Dealing with tentative data values in disconnected work groups. In *Proceedings Workshop on Mobile Computing Systems and Applications*. IEEE, December 1994.
- [36] Mario Tokoro and K. Tamaru. Acknowledging ethernet. *Compcon*, pages 320–325, October 1977.
- [37] Roy Want and Andy Hopper. Active badges and personal interactive computing objects. *IEEE Transactions on Consumer Electronics*, 38(1):10–20, Feb 1992.
- [38] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, Jan 1992.
- [39] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, September 1991.
- [40] Mark Weiser. Hot topic: Ubiquitous computing. *IEEE Computer*, pages 71–72, October 1993.
- [41] Mark Weiser. Some computer science issues in ubiquitous computing. *CACM*, 36(7):74–83, July 1993. In Special Issue, Computer-Augmented Environments.

- [42] Mark Weiser. The world is not a desktop. *Interactions*, pages 7–8, January 1994.
- [43] Mark Weiser, Alan Demers, Brent Welch, and Scott Shenkar. Scheduling for reduced CPU energy. In *Operating System Design and Implementation (OSDI)*, Monterey, CA, 1994.

SCALABLE SUPPORT FOR TRANSPARENT MOBILE INTERNETWORKING

David B. Johnson

*Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891
USA*

ABSTRACT

This paper considers the problem of providing transparent support for very large numbers of mobile hosts within a large internetwork such as the Internet. The availability of powerful mobile computing devices and wireless networking products and services is increasing dramatically, but internetworking protocols such as IP used in the Internet do not currently support host movement. To address this need, the Internet Engineering Task Force (IETF) is currently developing protocols for mobile hosts in the Internet. This paper analyzes the problem to be solved, reviews the current state of that effort, and discusses its scalability to very large numbers of mobile hosts in a large internetwork.

1 INTRODUCTION

The global Internet is growing at a tremendous rate. There are now about 5 million hosts connected to the Internet, and this number is doubling approximately every year. The average time between new networks connecting to the Internet is about 10 minutes. Initiatives such as the National Information Infrastructure and the increasing commercial uses of the Internet are likely to create even faster growth in the future.

Previously appeared in *Wireless Networks*, special issue on "Recent Advances in Wireless Networking Technology," 1995. Copyright ©1995 by Baltzer Science Publishers. Reprinted by permission.

At the same time, portable computing devices such as laptop and palmtop computers are becoming widely available at very affordable prices, and many new wireless networking products and services are becoming available based on technologies such as spread-spectrum radio, infrared, cellular, and satellite. Mobile computers today often are as capable as many home or office desktop computers and workstations, featuring powerful CPUs, large main memories, hundreds of megabytes of disk space, multimedia sound capabilities, and color displays. High-speed local area wireless networks are commonly available with speeds up to 2 megabits per second, and wide-area wireless networks are available that provide metropolitan or even nationwide service.

With these dramatic increases in portability and ease of network access, it becomes natural for users to expect to be able to access the Internet at any time and from anywhere, and to transparently remain connected and continue to use the network as they move about. However, internetworking protocols such as IP [23] used in the Internet do not currently support host mobility. A mobile user, today, must generally change IP addresses when connecting to the Internet at a different point or through a different network; the user must modify a number of configuration files and restart all network connections, making host movement difficult, time consuming, and error prone.

To address this need in the Internet, the Mobile IP Working Group of the Internet Engineering Task Force (IETF) has been working over the past few years to develop standard protocols to support mobile hosts operating in the Internet [6, 7, 8, 9, 10, 13, 14, 17, 18, 19, 20, 21, 22, 25, 29, 30, 31, 32, 34]. The IETF is the principal standards development body for protocols in the Internet. This work on IETF Mobile IP represents the contributions of many people within the Working Group, and development of these protocols is still underway. This paper analyzes the problem to be solved, reviews the current state of that effort, and discusses its scalability to very large numbers of mobile hosts in a large internetwork.

Section 2 of this paper describes the general problem of mobility management and packet routing to mobile hosts in a large internetwork. Section 3 gives a summary of the current state of the basic IETF Mobile IP protocol, and Section 4 describes extensions to this protocol also being developed within the IETF for optimizing packet routing to mobile hosts. Section 5 discusses the scalability of this work to very large numbers of mobile hosts, and Section 6 presents conclusions.

2 PROBLEM ANALYSIS

2.1 Internetwork Routing

In order to provide scalable routing support, internetworking protocols such as IP [23], ISO CLNP [27], NetWare IPX [33], and AppleTalk [28], use *hierarchical* addressing and routing schemes. For example, in IP, the network address of a host is divided into two levels of hierarchy: a *network number* identifying the network to which the host is connected, and a *host number* identifying the particular host within that network. Routers within the Internet know (and care) only how to route packets based on the network number of the destination address in each packet; once the packet reaches that network, it is then delivered to the correct individual host on that network.

Aggregating the routing decision at each level of the hierarchy in this way reduces the size of the routing tables that each router must maintain, reduces the size of the routing updates that routers must exchange, and simplifies the decisions at each router. Hierarchical addressing and routing has proven to be essential to keep up with the exponential growth of the Internet, in particular. The original two-level hierarchy of Internet addressing in IP has already been transparently extended at the bottom with *subnetting* [16] and at the top through use of *CIDR* [5]. In the IETF's "IPng" effort to develop the next generation of the IP protocol [2], support for many more levels of hierarchy than in the present version of IP is an explicit design goal [15].

However, this hierarchy in addressing and routing prevents packets from being routed correctly to a mobile host while it is away from its home network. Since a host's address logically encodes its location, without special handling for mobility, packets addressed to a mobile host will be routed by the Internet only to the mobile host's home network. This problem exists with any protocol using a hierarchical addressing and routing scheme, whether the hierarchy is provider-based or geographical.

2.2 Location Registry

It is important to be able to support packet routing to mobile hosts from existing correspondent hosts that have not been modified to support mobility. Given the very large number of hosts already deployed within the Internet, it seems quite likely that some will not be upgraded to support mobility for some time. Furthermore, some existing hosts may never be upgraded, for example because

the organizations owning them may lack the interest or resources to upgrade, or because the original vendor no longer offers support for particular products owned by some customers. The ability to support unmodified correspondent hosts also allows any correspondent host to communicate with any other host without being concerned whether or not it is currently mobile and away from its home network.

It therefore becomes logical to provide basic mobility support for a mobile host through a location registry recording the mobile host's current location, that can be accessed through the mobile host's home network. An unmodified correspondent host (or one that simply does not know that a particular mobile host is in fact mobile) will send IP packets for that mobile host in the same way as all IP packets are sent today. Such packets will thus reach the mobile host's home network, where they may be intercepted by some mobility support agent and forwarded to the mobile host's current location.

Requiring the sender to instead explicitly query the location registry before sending a packet is incompatible with the goals of supporting existing unmodified correspondent hosts and of not requiring the sender to be aware of whether a particular destination host is currently mobile. Accessing the location registry through the mobile host's home network also avoids any requirement for changes to the basic routing algorithms of the Internet, and allows each organization owning some network to manage this functionality for all of its own mobile hosts with this home network, improving scalability and easing manageability.

In addition, requiring the location registry to be explicitly queried in this way, either would require this overhead to be added for *all* destination addresses or would require restrictions on the assignment of IP addresses. If a host's address encodes information as to whether it is a *mobile* or a *stationary* host, then only packets destined for mobile host's need to cause the location registry to be queried. However, this encoding would require permanently designating each host into one of these two classes, greatly reducing flexibility and complicating host and network administration.

2.3 Packet Tunneling

Some mechanism is needed to cause a packet addressed to a mobile host to be routed to that host's current location rather than (only) to its home network. In order to avoid distributing routing information for a mobile host throughout

the Internet so that the new routing decision could be made at each hop, it must be possible to modify each packet for a mobile host in such a way that the routing infrastructure of the Internet will route the modified packet to a location identified in the packet. This type of packet forwarding is known as *tunneling*. For IP, tunneling may in general be done using an encapsulation protocol or through an IP option such as loose source routing [23].

In tunneling a packet from one node to another, only these two nodes (the two endpoints of the tunnel) need know that tunneling is taking place. Routers between the node tunneling the packet and the new destination node to which the packet is tunneled, simply route the packet at each hop in the same way as any ordinary IP packet. There is thus no need to modify existing routers, such as within the Internet backbone, nor to modify existing Internet routing algorithms.

2.4 Caching and Consistency

The mechanisms suggested above allow packets for a mobile host to be sent to it at its current location, but support forwarding only through an agent on the mobile host's home network. For example, if a mobile host, say MH1, is visiting some network, even packets from a correspondent host on this same network must be routed through the Internet to this agent on MH1's home network, only to then be tunneled back to the original network for delivery to MH1. If the correspondent host in this example is actually another mobile host, say MH2, then packets from MH1 to MH2 must likewise be routed through some agent on MH2's home network and back to the original network for delivery to MH2. This indirect routing places unnecessary overhead on the Internet, on each mobile host's home network, and on the agent providing forwarding service from each home network. Such indirect routing may also significantly increase the latency in packet delivery to a mobile host.

Correspondent hosts that have been modified to support mobility should be able to learn the current location of a mobile host with which they are communicating, and to then use this location to tunnel their own future packets directly to the mobile host. By caching this location, the expense of discovering this location can be avoided on each individual packet sent to the mobile host. However, this caching creates the problem of *cache consistency* when the mobile host then moves to a new location, since the correspondent host's cache will still point to the old location. In order to support smooth handoff from one location to another, the protocol must be able to update correspondent host's

caches, and should provide some support for packets that may be tunneled based on a temporarily out-of-date cache.

3 THE BASIC MOBILE IP PROTOCOL

This section provides an overview of the current state of the basic IETF Mobile IP protocol [20]. The protocol provides transparent routing of packets to a mobile host and requires no modification to existing routers or correspondent hosts. No support is provided, however, for caching a mobile host's location at correspondent hosts or for allowing correspondent hosts to tunnel packets directly to a mobile host's current location. These features are being developed within the IETF as a separate set of extensions to this basic protocol, and are discussed in Section 4.

3.1 Infrastructure

Each mobile host is assigned a unique *home address* in the same way as any other Internet host, within its *home network*. Hosts communicating with a mobile host are known as *correspondent hosts* and may, themselves, be either mobile or stationary. In sending an IP packet to a mobile host, a correspondent host always addresses the packet to the mobile host's home address, regardless of the mobile host's current location.

Each mobile host must have a *home agent* on its home network that maintains a registry of the mobile host's current location. This location is identified as a *care-of address*, and the association between a mobile host's home address and its current care-of address is called a *mobility binding*, or simply a *binding*. Each time the mobile host establishes a new care-of address, it must *register* the new binding with its home agent so that the home agent always knows the current binding of each mobile host that it serves. A home agent may handle any number of mobile hosts that share a common home network.

A mobile host, when connecting to a network away from its home network, may be assigned a care-of address in one of two ways. Normally, the mobile host will attempt to discover a *foreign agent* within the network being visited, using an *agent discovery* protocol. The mobile host then *registers* with the foreign agent, and the IP address of the foreign agent is used as the mobile host's care-of address. The foreign agent acts as a local forwarder for packets arriving for

the mobile host and for all other locally visiting mobile hosts registered with this foreign agent. Alternatively, if the mobile host can obtain a temporary local address within the network being visited (such as through DHCP [4]), the mobile host may use this temporary address as its care-of address.

While a mobile host is away from its home network, a mobile host's home agent acts to forward all packets for the mobile host to its current location for delivery locally to the mobile host. Packets addressed to the mobile host that appear on the mobile host's home network must be intercepted by the mobile host's home agent, for example by using "proxy" ARP [24] or through cooperation with the local routing protocol in use on the home network.

For each such packet intercepted, the home agent tunnels the packet to the mobile host's current care-of address. If the care-of address is provided by a foreign agent, the foreign agent removes any tunneling headers from the packet and delivers the packet locally to the mobile host by transmitting it over the local network on which the mobile host is registered. If the mobile host is using a locally obtained temporary address as a care-of address, the tunneled packet is delivered directly to the mobile host.

Home agents and foreign agents may be provided by separate nodes on a network, or a single node may implement the functionality of both a home agent (for its own mobile hosts) and a foreign agent (for other visiting mobile hosts). Similarly, either function or both may be provided by any of the existing IP routers on a network, or they may be provided by separate support hosts on that network.

3.2 Agent Discovery

The *agent discovery* protocol operates as a compatible extension of the existing *ICMP router discovery* protocol [3]. It provides a means for a mobile host to detect when it has moved from one network to another, and for it to detect when it has returned home. When moving into a new foreign network, the agent discovery protocol also provides a means for a mobile host to discover a suitable foreign agent in this new network with which to register.

On some networks, depending on the particular type of network, additional link-layer support may be available to assist in some or all of the purposes of the agent discovery protocol. A standard protocol must be defined for agent discovery, however, at least for use on networks for which no link-layer support

is available. By defining a standard protocol, mobile hosts are also provided with a common method for agent discovery that can operate in the same way over all types of networks. If additional link-layer support is available, it can optionally be used by mobile hosts that support it to assist in agent discovery.

Home agents and foreign agents periodically advertise their presence by multicasting an *agent advertisement* message on each network to which they are connected and for which they are configured to provide service. Mobile hosts listen for agent advertisement messages to determine which home agents or foreign agents are on the network to which they are currently connected. If a mobile host receives an advertisement from its own home agent, it deduces that it has returned home and registers directly with its home agent. Otherwise, the mobile host chooses whether to retain its current registration or to register with a new foreign agent from among those it knows of.

While at home or registered with a foreign agent, a mobile host expects to continue to receive periodic advertisements from its home agent or from its current foreign agent, respectively. If it fails to receive a number of consecutive expected advertisements, the mobile host may deduce either that it has moved or that its home agent or current foreign agent has failed. If the mobile host has recently received other advertisements, it may attempt registration with one of those foreign agents. Otherwise, the mobile host may multicast an *agent solicitation* message onto its current network, which should be answered by an agent advertisement message from each home agent or foreign agent on this network that receives the solicitation message.

3.3 Registration

Much of the basic IETF Mobile IP protocol deals with the issue of registration with a foreign agent and with a mobile host's home agent. When establishing service with a new foreign agent, a mobile host must register with that foreign agent, and must also register with its home agent to inform it of its new care-of address. When instead establishing a new temporarily assigned local IP address as a care-of address, a mobile host must likewise register with its home agent to inform it of this new address. Finally, when a mobile host returns to its home network, it must register with its home agent to inform it that it is no longer using a care-of address.

To register with a foreign agent, a mobile host sends a *registration request* message to the foreign agent. The registration request includes the address of

the mobile host and the address of its home agent. The foreign agent forwards the request to the home agent, which returns a *registration reply* message to the foreign agent. Finally, the foreign agent forwards the registration reply message to the mobile host. When registering directly with its home agent, either when the mobile host has returned home or when using a temporarily assigned local IP address as its care-of address, the mobile host exchanges the registration request and reply messages directly to its home agent.

Each registration with a home agent or foreign agent has associated with it a *lifetime* period, negotiated during the registration. After this lifetime period expires, the mobile host's registration is deleted. In order to maintain continued service from its home agent or foreign agent, the mobile host must re-register within this period. The lifetime period may be set to infinity, in which case no re-registration is necessary. When registering with its home agent on returning to its home network, a mobile host registers with a zero lifetime and deletes its current binding, since a mobile host needs no services of its home agent while at home.

3.4 Registration Authentication

All registrations with a mobile host's home agent must be authenticated in order to guard against malicious forged registrations that could arbitrarily redirect future packets destined to a mobile host. In particular, without authentication, an attacker could register a false care-of address for a mobile host, causing the mobile host's home agent to misroute packets destined for the mobile host. An attacker could, for example, reroute the mobile host's packets in order to eavesdrop on its traffic, alter any packets destined for the mobile host, or deny service to the mobile host by misdirecting its packets. Registration authentication must verify that the registration request legitimately originated with the mobile host, that it has not been altered in transit to the home agent, and that an old registration request is not being replayed (perhaps long after the mobile host was at that care-of address).

Although any authentication algorithm shared by a mobile host and its home agent may be used, the IETF protocol defines a standard authentication algorithm based on the MD5 message-digest function [26], using a secret key shared between these two nodes. MD5 is a *one-way* hash function, in that it is considered to be computationally infeasible to discover the input to the hash function given its output, or to find another sequence of input that produces the same output. A "keyed MD5" algorithm is used, in which the MD5 hash

over the bytes of the shared secret key and the important fields of the message is included in each registration message or reply; the secret key itself is not included in the message sent over the network. This authentication value allows the receiver to verify the source of the message and the fact that none of the important fields in the message (included in the hash) have been changed since the message was sent. If the hash matches at the receiver, the registration message must have been generated by a node knowing the secret key and must not have been modified in transit; without knowledge of the secret key included in the MD5 hash, no other node can modify or forge a registration message.

Administration of the shared secret key should be fairly simple, since both the mobile host and its home agent are owned by the same organization (both are assigned IP addresses in the home network owned by that organization). Manual configuration of the shared key may be performed, for example, any time the mobile host is at home, while other administration of these nodes is being performed.

Replay protection for registration messages may be provided under the IETF Mobile IP protocol using either *nonces* or *timestamps*. Using nonces, the home agent generates a random value and returns it to the mobile host (in cleartext) in its registration reply message, and the mobile host must include this same value in its next registration request message. If the value in the message does not match on the next registration attempt, for example because the mobile host has lost its saved state containing this value, the home agent returns a registration error and includes the correct new value in the registration reply. The next registration attempt by the mobile host should then succeed, and no other node can use this value in the message to forge a registration message, since it does not know the share secret key used in the message authentication that must be computed and included in each registration message. The use of timestamps for replay protection is similar, except that the timestamp included in the registration message must closely match the current time at the receiver.

3.5 Tunneling

The Mobile IP protocol allows the use of any tunneling method shared between a mobile host's home agent and its current foreign agent (or the mobile host itself when a temporary local IP address is being used as a care-of address). During registration with its home agent, a list of supported tunneling methods is communicated to the home agent. For each packet later tunneled to the mobile host, the home agent may use any of these supported methods.

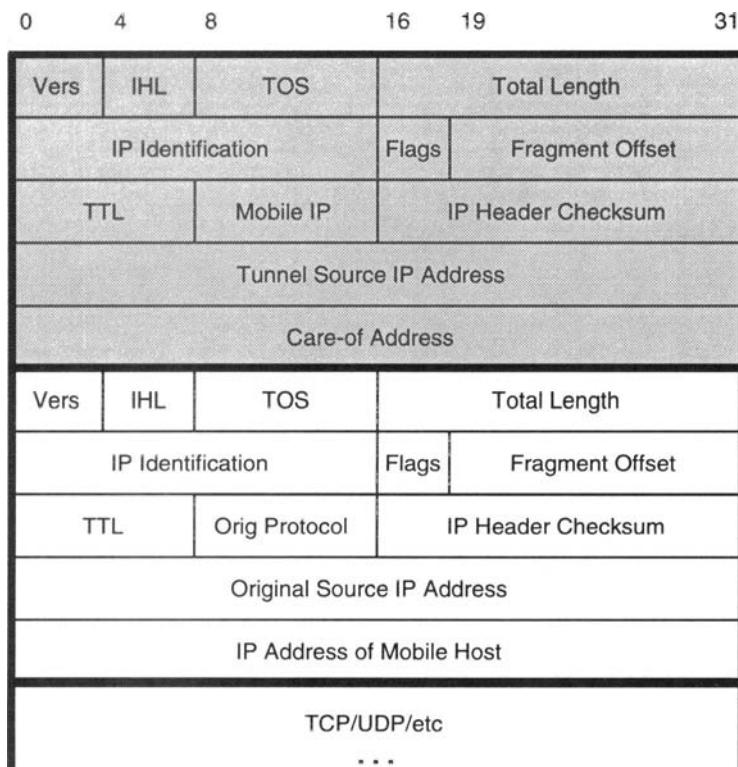


Figure 1 Mobile IP tunneling using “IP in IP” encapsulation

The protocol requires support for “IP in IP” encapsulation for tunneling, as illustrated in Figure 1. In this method, to tunnel an IP packet, a new IP header is wrapped around the existing packet; the source address in the new IP header is set to the address of the node tunneling the packet (the home agent), and the destination address is set to the mobile host’s care-of address. The new header added to the packet is shaded in gray in Figure 1. This type of encapsulation may be used for tunneling any packet, but the overhead for this method is the addition of an entire new IP header (20 bytes) to the packet.

Support is also recommended for a more efficient “minimal” tunneling protocol [10, 12], which adds only 8 or 12 bytes to each packet. This tunneling protocol is illustrated in Figure 2, with the new header added to the packet shaded in gray. Here, only the modified fields of the original IP header are

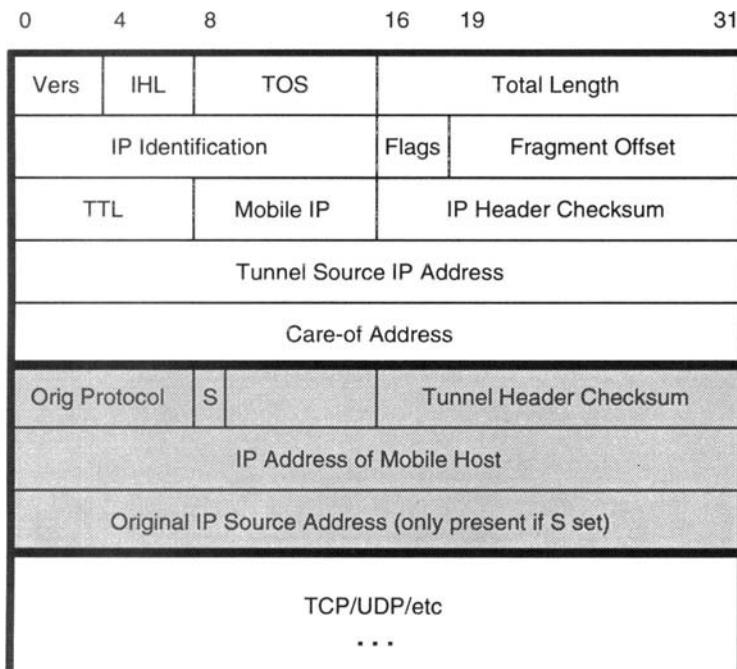


Figure 2 Mobile IP tunneling using the minimal tunneling protocol

copied into a new forwarding header added to the packet between the original IP header and any transport-level header such as TCP or UDP. The fields in the original IP header are then replaced such that the source address is set to the address of the node tunneling the packet (only if the packet is being tunneled by a node other than the original sender), and the destination address is set to the mobile host's care-of address. This type of encapsulation adds less overhead to each packet, but it cannot be used with packets that have already been fragmented by IP, since the small forwarding header does not include the fields needed to represent that the original packet is a fragment rather than a whole IP packet.

4 ROUTE OPTIMIZATION

The basic IETF Mobile IP protocol fulfills its primary goal of providing transparent packet routing to mobile hosts operating in the Internet. However, *all* packets for a mobile host away from home must be routed through the mobile host's home network and home agent, severely limiting the performance transparency of the protocol and creating a significant bottleneck to potential scalability.

As suggested in Section 2, what is needed is the ability for correspondent hosts to be able to cache the location of a mobile host and to then tunnel packets directly to the mobile host at its current location. This functionality has become known within the IETF as *route optimization*, and a group consisting of Andrew Myles of Macquarie University, Charles Perkins of IBM, and the author have been working particularly to develop this functionality within the IETF protocol [14]. This section provides an overview of the current state of the protocol extensions for route optimization.

4.1 Location Caching

Any node may optimize its own communication with mobile hosts by maintaining a *binding cache* in which it caches the binding of one or more mobile hosts. When sending a packet to a mobile host, if the sender has a binding cache entry for this mobile host, it may tunnel its own packet directly to the care-of address indicated in the cached binding. Likewise, a router when forwarding a packet may tunnel the packet directly to the destination mobile host's care-of address if the router has an entry in its binding cache for the destination IP address of the packet; such a router may thus optimize the mobile host communication for a group of nodes not supporting the route optimization extensions.

In the absence of any binding cache entry, packets destined for a mobile host will be routed to the mobile host's home network in the same way as any other IP packet, and are then tunneled to the mobile host's current care-of address by the mobile host's home agent. This is the only routing mechanism supported by the basic Mobile IP protocol. With route optimization, though, as a side effect of this indirect routing of a packet to a mobile host, the original sender of the packet is informed of the mobile host's current mobility binding (Section 4.3), giving the sender an opportunity to cache the binding.

A node may create a binding cache entry for a mobile host only when it has received and authenticated the mobile host's binding. Likewise, a node may update an existing binding cache entry for a mobile host, such as after the mobile host has moved to a new foreign agent, only when it has received and authenticated the mobile host's new binding.

A binding cache will, by necessity, have a finite size. Any node implementing a binding cache may manage the space in its cache using any local cache replacement policy such as LRU. If a packet is sent to a destination address for which the cache entry has been dropped from the cache, the packet will be routed normally to the mobile host's home network and will be tunneled to the mobile host's care-of address by its home agent. As when a binding cache entry is initially created, this indirect routing to the mobile host will result in the original sender of the packet being informed of the mobile host's current binding, allowing it to add this entry again to its binding cache.

Optimal routing of packets from a correspondent host can be achieved if the correspondent host implements a binding cache. A router implementing a binding cache can also provide routing assistance for packets that it forwards from correspondent hosts that do not implement the Mobile IP route optimization extensions. For example, a local network of nodes that do not implement route optimization could be supported by a common first-hop router that maintains a binding cache. Router software should be configurable, however, to allow disabling the maintenance of a binding cache, such as within backbone routers, where little or no benefit of caching could be obtained.

4.2 Foreign Agent Handoff

When a mobile host moves and registers with a new foreign agent, the basic Mobile IP protocol does not notify the mobile host's previous foreign agent that the host has moved. After the mobile host's new registration at its home agent, IP packets intercepted by the home agent are tunneled to the mobile host's new care-of address, but any packets in flight that had already been tunneled by the home agent to the old care-of address are lost and are assumed to be retransmitted by higher-level protocols if needed. The old foreign agent eventually deletes the mobile host's registration after the expiration of the lifetime period established when the mobile host registered with that foreign agent.

Route optimization extends the registration protocol to provide a means for a mobile host's previous foreign agent to be reliably notified that the mobile host has moved, and optionally to inform it of the mobile host's new binding. When registering with a foreign agent, a mobile host may establish a "registration key," acting as a session key for its registration with this foreign agent. When the mobile host later moves and registers a different care-of address, it may notify this previous foreign agent by sending it a *binding update* message; this binding update message is authenticated in the same way as registration messages between a mobile host and its home agent, but in this case, using the registration key established when it registered with that foreign agent as the shared secret key for the authentication. After being established, such a registration key could also optionally be used to encrypt packets sent between the mobile host and its foreign agent, in order to improve privacy in the common case in which they are connected by a wireless link, but such use has not yet been considered within the IETF.

Notifying the previous foreign agent that the mobile host has moved allows packets in flight to this foreign agent, as well as packets tunneled from correspondent hosts with out-of-date binding cache entries for the mobile host (they have not yet learned that the mobile host has moved), to be forwarded to the mobile host's new care-of address. When notified of the mobile host's new binding, the previous foreign agent may create a binding cache entry for the mobile host, acting as a "forwarding pointer" to its new location. This notification also allows any resources consumed by the mobile host's registration at the previous foreign agent (such as radio channel reservations) to be released immediately, rather than waiting for the mobile host's registration to expire.

Such a "forwarding pointer" binding cache entry at a mobile host's previous foreign agent is treated in the same way as any other binding cache entry. In particular, this binding cache entry may be deleted from the cache at any time. Suppose a node (such as this previous foreign agent) receives some packet that has been tunneled to this node, but this node is unable to deliver the packet locally to the destination mobile host (it is not the mobile host itself, and it does not believe that it is currently serving as a foreign agent for this mobile host). In this case, the node tunnels the packet to the mobile host's own address, which will cause the packet to reach the home network and be intercepted by the mobile host's home agent in the same way as any other packet addressed to the mobile host. The home agent will then extract and re-tunnel the packet to the mobile host's current location.

4.3 Binding Cache Updates

When a mobile host's home agent intercepts a packet from the home network and tunnels it to the mobile host, the home agent may deduce that the original sender of the packet has no binding cache entry for the destination mobile host. In this case, the home agent sends a *binding update* message to the sender, informing it of the mobile host's current binding. No acknowledgement for this binding update is needed, since any future packets intercepted by the home agent from this sender for the mobile host will serve to cause transmission of a new binding update.

Similarly, when a node receives a packet that was tunneled to this node, if the node has a binding cache entry for the destination IP address of the packet carried within the tunnel, the node may deduce that the original sender of the packet has an out-of-date binding cache entry for this destination mobile host (pointing to this node). In this case, the node sends a *binding warning* message to the original sender of the packet, advising it to send a *binding inquire* message to the mobile host's home agent to request the mobile host's current binding as a binding update. As with the binding update message from the home agent, no acknowledgement for this binding warning message is needed, as any future packets tunneled to the same node from this sender for the mobile host will serve to cause transmission of a new binding warning.

With the exception of the notification to a mobile host's previous foreign agent (which is sent by the mobile host itself), all binding update messages are sent by a mobile host's home agent, which is in complete control of which correspondent hosts it allows to learn the mobile host's binding. If, for any local administrative reasons, the home agent wants to keep a particular mobile host's current binding private (from all or only some correspondent hosts), it is not required to send a binding update that would otherwise be sent by the protocol.

Included in each binding update message sent by the home agent is an indication of the time remaining in the lifetime associated with the mobile host's current registration. Any binding cache entry established or updated in response to this binding update must be marked to be deleted after the expiration of this period. A node wanting to provide continued service with a particular binding cache entry may attempt to reconfirm that binding before the expiration of this lifetime period. Binding cache entry reconfirmation may be appropriate when the node has indications (such as an open transport-level connection to the mobile host) that the binding cache entry is still needed. This reconfirmation

is performed by the node actively requesting the mobile host's home agent to send a new binding update message to the node.

Each node must provide some mechanism to limit the rate at which it sends binding update or binding warning messages to the same node about any given binding. Some nodes will not implement the route optimization extensions of the Mobile IP protocol, and those that do may be limited in the number of bindings they can cache or the speed with which they can process these messages. A new binding update or binding warning message should not be sent for each individual packet described above that is received over a short period of time; rather, some minimum interval should be maintained between binding update or binding warning messages, and after a small number of these messages have been sent to the same node about some binding, the sending node must quickly increase the interval between new binding update or binding warning messages.

4.4 Binding Update Authentication

All messages that add or change an entry in a binding cache must be authenticated using the same type of authentication algorithm as is used in the basic Mobile IP protocol for registration with a mobile host's home agent (Section 3.4). This authentication verifies the source of the message and ensures that none of the important fields of the message have been changed since the message was sent.

In particular, a node receiving a binding update message must verify the message's authentication before altering the contents of its binding cache in response to the message. This requirement for authentication covers all binding update messages: those sent to build or update a binding cache entry in response to a packet routed indirectly to a mobile host, as well as those sent to notify a mobile host's previous foreign agent that it has moved. Without such authentication, a malicious node anywhere in the Internet could forge a binding update message, allowing it to arbitrarily intercept or redirect packets destined for any other node in the Internet.

In the basic Mobile IP protocol, only a mobile host's registration with its home agent must be authenticated, allowing the simple solution of a manually configured secret key shared between the mobile host and its home agent. For route optimization, a home agent must in general be able to send an authenticated binding update message to any other node in the Internet, since any

node may want to maintain a binding cache containing entries for one or more mobile hosts served by that home agent. This form of general authentication is currently complicated by the lack of a standard key management or authentication protocol in the Internet, and by the lack of any generally available key distribution infrastructure; patent restrictions and export controls on the necessary cryptographic algorithms have slowed development and deployment of such facilities in the Internet.

A number of restricted authentication schemes for route optimization are possible in the short term, however, before the necessary protocols and infrastructure are available. The route optimization extensions within the IETF [14] have currently been designed to utilize manually configured shared secret keys in the same way as the authentication used in registration in the basic Mobile IP protocol, but the required shared keys may be configured to reduce the number of pairwise keys that must be maintained. In particular, by manually establishing a shared secret key with a particular home agent, a node is able to receive authenticated binding updates (and thus to maintain binding cache entries) for all mobile hosts served by this home agent; if no shared secret key is available for some node, no binding update messages are sent by the home agent to that node, and only the basic Mobile IP protocol is used for packets sent to mobile hosts from that node.

This configuration of manually established shared secret keys is fairly natural, since the mobile hosts served by any particular home agent, in general, all belong to a single organization (that also owns the home agent and the home network). If the user of a node often collaborates with any number of people from this organization, establishing the shared secret key with this home agent may be worthwhile. The route optimization procedures described in Sections 4.2 and 4.3 have been designed with this restricted style of authentication in mind, and may be modified when more general authentication mechanisms become available.

This type of authentication is secure as long as the shared secret key remains secret, and it is not subject to export restrictions since it does not use encryption. A simpler style of authentication that also does not use encryption was proposed within the IETF for the IMHP protocol [13, 17, 22], and was also used in recent mobile routing work done at Harvard University [1]. This scheme relies on a general property of routing in the Internet in which nodes not connected to the normal routing path of a packet cannot eavesdrop on or reroute that packet. By including a randomly generated authenticator value in a packet sent to another node, the original sender can authenticate the reply from that node, by requiring that the same random value is returned in the

reply. Although this simpler scheme requires no configuration of shared secret keys, it is less secure, since this general property of Internet routing security has been severely weakened by increasing attacks in recent years; in addition, this scheme is further weakened, since any of the links over which such an authentication may take place may be wireless, enhancing the ability of any attacker to eavesdrop on the exchange containing the authenticator value.

5 PROTOCOL SCALABILITY

The combination of the basic IETF Mobile IP protocol described in Section 3 and the extensions for route optimization described in Section 4 can provide highly scalable support for packet routing to large numbers of mobile hosts in the Internet. This section considers the different factors affecting the scalability of the protocol.

5.1 The Home Network

Each organization owning an IP network supports all mobile hosts for which this is the home network. As new networks are added to the Internet, each deploys its own home agent to support its own mobile hosts. This arrangement allows mobility support within the home network to scale as new organizations and new networks connect to the Internet, avoiding any centralized support bottleneck. Since a home agent maintains the location registry and tunnels packets only for the mobile hosts for which this is the home network, this approach allows these functions to scale with the number of networks containing mobile hosts.

Each organization may also control the level of expense or effort which they expend to support their own mobile hosts, and their own mobile hosts directly benefit from these expenditures. For example, an organization wanting to provide higher performance or more reliable access to the home agent for any of its mobile hosts may install higher bandwidth or additional links connecting their own home network to the Internet. The functionality of the home agent may also be replicated or distributed on multiple nodes on the home network; as long as a consistent view of the bindings of this home network's mobile hosts is maintained, such arrangements are entirely at the option of the organization owning the network and need not affect other nodes within the Internet. The

home agent functionality and the home network may be scaled to support any number of mobile hosts owned by this organization.

While a mobile host is at home, it is treated in the same way as any ordinary IP host, and no overhead is added to packets sent to it while at home. When the mobile host leaves home and registers a care-of address, its home agent begins tunneling packets for it, binding cache entries are gradually created at different correspondent hosts or routers, and they then begin tunneling packets for the mobile host directly to the mobile host's current location. As the mobile host moves from one care-of address to another, the binding caches are updated as needed. When the mobile host later returns home, this same mechanism causes these binding cache entries to be deleted; packets destined to this mobile host are then sent in the same way as any IP packets sent to an ordinary stationary host that has never been mobile.

It thus becomes feasible to upgrade all hosts, at any convenient time, to be "mobile capable," with no performance penalty to the network or to the host for the extra capability of being mobile [11]. Any mobile capable host could then become mobile at any future time as needed simply by leaving its home network and registering elsewhere. This property simplifies the installation of new hosts, since no decision need be made as to whether each host will need to be mobile at any future time.

5.2 The Foreign Network

Each organization owning an IP network that allows mobile hosts to visit deploys its own foreign agent to support mobile hosts visiting that network. This arrangement allows mobility support within the foreign network to scale as new organizations and new networks connect to the Internet. Since a foreign agent maintains a list of only those mobile hosts currently registered with it, and only locally delivers packets for these mobile hosts, this approach allows these functions to scale with the number of networks that allow mobile hosts to visit.

In addition, each organization owning an IP network allowing mobile hosts to visit may control its own resource allocation within that network as needed by any local policies of that organization. For example, a foreign agent may be configured to limit the number of simultaneous visitors that it allows to register; if additional mobile hosts request registration, the foreign agent may return an error to each indicating that registration has been denied due to local resource allocation limits. Any organization may install additional or

more powerful foreign agents or higher bandwidth local networks in order to provide any desired level of support for visiting users. Each organization may also impose any administrative policies on the provision of service to visiting mobile hosts. For example, they may only allow mobile hosts for which prior billing arrangements have been established to register.

By deploying one or more foreign agents, the protocol places no new demands on IP address space allocation, avoiding the limits to scalability that would otherwise be imposed by the current limits on available IP address space. Any organization wanting to provide service for visiting mobile hosts but not willing to deploy a foreign agent may support any number of visitors by reserving a portion of their local IP address space for dynamic allocation as care-of addresses for visiting mobile hosts.

5.3 Binding Caches

The deployment and operation of a binding cache in any node is only an optimization to the protocol, and no binding caches are required, although the use of binding caches is highly desirable. Each binding cache may scale to any size as needed by any local administrative policies, but no specific binding cache size is imposed by the protocol. Similarly, any local cache replacement policy may be used to manage the space within the cache.

If the binding cache at some node is too small to be able to store a cached binding for each mobile host with which this node is actively communicating, the local cache replacement policy determines which entries are retained in the cache. For example, the use of LRU replacement will keep the most recently used entries in the cache. Other possible cache replacement policies might weight each entry by the number of times it had been recently accessed, or by some administratively assigned priority based on a list of preferred hosts for which bindings should be cached. Such decisions are entirely local to the node (and organization) implementing the binding cache.

The use of binding caches improves the scalability of the protocol by avoiding the need to send most packets to and from the mobile host's home network, and by avoiding the need for the home agent in the mobile host's home network to handle each packet. The binding cache in a correspondent host maintains cache entries only for the individual mobile hosts with which that correspondent host is communicating. This approach scales well, as each individual correspondent host will at any time only be communicating with a limited number of mobile

hosts. Furthermore, since in general the set of mobile hosts with which a correspondent host is communicating will change only slowly over time, any reasonable cache replacement policy such as LRU should work well.

5.4 Impact on the Network

No changes to the routing infrastructure of the Internet are required to support Mobile IP. By tunneling packets to a mobile host, all routers through which the tunneled packet must pass treat the packet exactly as any ordinary IP packet, using existing Internet routing algorithms. The routing scalability of the Internet is thus maintained, since each router need not know the location of any individual mobile hosts, even though it may forward packets to them; only the two endpoints of the tunnel need know that tunneling is taking place or need care that mobility is the purpose of the tunneling. The Mobile IP protocol can thus be deployed incrementally, with each organization adding home agents or foreign agents as the need arises. Any or all hosts and routers can be upgraded at any time, if desired, to support binding caches.

By using route optimization, the overall overhead on the Internet can be minimized. Routing packets indirectly to a mobile host through the mobile host's home network and home agent places unnecessary overhead on all links and nodes along this path, but route optimization allows this longer, indirect path to be avoided. Route optimization also reduces the resource demands on each home network, and avoids any possible performance bottleneck at the home network or at the home agent.

6 CONCLUSION

Recent increases in the availability of mobile computers and wireless networks provides the opportunity to integrate these technologies seamlessly into the Internet. Mobile users should be able to move about, transparently remaining connecting to the Internet, utilizing the best available network connection at any time, whether wired or wireless. For example, a mobile host in its owner's office may be connected to an Ethernet, but when disconnected and carried away, it could transparently switch to a connection through a high-speed local area wireless network. While moving around within the building, the host could switch transparently from one wireless subnet to another, and when leaving the building, could again switch transparently to a wide-area wireless data service.

The current work in the IETF Mobile IP Working Group provides a good approach to reaching this vision of seamless transparent mobility. These protocols can efficiently scale to very large numbers of mobile hosts operating in a large internetwork. Such scalability will become crucial as the Internet continues its exponential growth, and as mobile users begin to account for a growing fraction of this population.

Acknowledgements

This paper has benefited greatly from discussions with many other participants in the Mobile IP Working Group of the Internet Engineering Task Force (IETF). I would particularly like to thank Andrew Myles and Charlie Perkins for their collaboration in our work within the IETF. I would also like to thank the anonymous referees for their comments and suggestions which have helped to improve the clarity of the paper. The protocols described in this paper are a product of the Mobile IP Working Group of the IETF, but the views and conclusions expressed here are those of the author.

This research was supported in part by the Wireless Initiative of the Information Networking Institute at Carnegie Mellon University, and by the National Science Foundation under CAREER Award NCR-9502725.

REFERENCES

- [1] Trevor Blackwell, Kee Chan, Koling Chan, Thomas Charuhas, James Gwertzman, Brad Karp, H. T. Kung, W. David Li, Dong Lin, Robert Morris, Robert Polansky, Diane Tang, Cliff Young, and John Zao. Secure shortcut routing for Mobile IP. In *Proceedings of the USENIX Summer 1994 Technical Conference*, June 1994.
- [2] Scott Bradner and Allison Mankin. The recommendation for the IP Next Generation protocol. Internet Request For Comments RFC 1752, January 1995.
- [3] Stephen E. Deering. ICMP router discovery messages. Internet Request For Comments RFC 1256, September 1991.
- [4] Ralph Droms. Dynamic Host Configuration Protocol. Internet Request For Comments RFC 1541, October 1993.

- [5] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): an address assignment and aggregation strategy. Internet Request For Comments RFC 1519, September 1993.
- [6] John Ioannidis, Dan Duchamp, and Gerald Q. Maguire Jr. IP-based protocols for mobile internetworking. In *Proceedings of the SIGCOMM '91 Conference: Communications Architectures & Protocols*, pages 235–245, September 1991.
- [7] John Ioannidis, Gerald Q. Maquire Jr, and Steve Deering. Protocols for supporting mobile IP hosts. Internet Draft, June 1992. Work in progress.
- [8] John Ioannidis and Gerald Q. Maguire Jr. The design and implementation of a mobile internetworking architecture. In *Proceedings of the Winter 1993 USENIX Conference*, pages 491–502, January 1993.
- [9] David B. Johnson. Mobile host internetworking using IP loose source routing. Technical Report CMU-CS-93-128, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, February 1993.
- [10] David B. Johnson. Transparent Internet routing for IP mobile hosts. Internet Draft, July 1993. Work in progress.
- [11] David B. Johnson. Ubiquitous mobile host internetworking. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 85–90, October 1993.
- [12] David B. Johnson. Scalable and robust internetwork routing for mobile hosts. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 2–11, June 1994.
- [13] David B. Johnson, Andrew Myles, and Charles Perkins. The Internet Mobile Host Protocol (IMHP). Internet Draft, February 1994. Work in progress.
- [14] David B. Johnson, Charles Perkins, and Andrew Myles. Route optimization in Mobile IP. Internet Draft, March 1995. Work in progress.
- [15] Frank Kastenholz and Craig Partridge. Technical criteria for choosing IP:the next generation (IPng). Internet Draft, May 1994. Work in progress.
- [16] J. Mogul and J. Postel. Internet standard subnetting procedure. Internet Request For Comments RFC 950, August 1985.

- [17] Andrew Myles, David B. Johnson, and Charles Perkins. A mobile host protocol supporting route optimization and authentication. *IEEE Journal on Selected Areas in Communications*, special issue on Mobile and Wireless Computing Networks, 13(5):839–849, June 1995.
- [18] Andrew Myles and Charles Perkins. Mobile IP (MIP). Internet Draft, September 1993. Work in progress.
- [19] John Penners and Yakov Rekhter. Simple Mobile IP (SMIP). Internet Draft, September 1993. Work in progress.
- [20] Charles Perkins, editor. IP mobility support. Internet Draft, May 1995. Work in progress.
- [21] Charles Perkins and Yakov Rekhter. Support for mobility with connectionless network layer protocols (transport layer transparency). Internet Draft, January 1993. Work in progress.
- [22] Charles E. Perkins, Andrew Myles, and David B. Johnson. The Internet Mobile Host Protocol (IMHP). In *Proceedings of INET'94/JENC5: The Annual Conference of the Internet Society*, held in conjunction with 5th Joint European Networking Conference, pages 642–1–642–9, June 1994.
- [23] J. B. Postel, editor. Internet Protocol. Internet Request For Comments RFC 791, September 1981.
- [24] J. B. Postel. Multi-LAN address resolution. Internet Request For Comments RFC 925, October 1984.
- [25] Yakov Rekhter and Charles Perkins. Short-cut routing for mobile hosts. Internet Draft, July 1992. Work in progress.
- [26] Ronald L. Rivest. The MD5 message-digest algorithm. Internet Request For Comments RFC 1321, April 1992.
- [27] Marshall T. Rose. *The Open Book: A Practical Perspective on OSI*. Prentice Hall, Englewood Cliffs, NJ, 1990.
- [28] Gursharan S. Sidhu, Richard F. Andrews, and Alan B. Oppenheimer. *Inside AppleTalk*. Addison Wesley, Reading, Massachusetts, 1990.
- [29] C. Sunshine and J. Postel. Addressing mobile hosts in the ARPA Internet environment. Internet Engineering Note IEN 135, March 1980.

- [30] Fumio Teraoka, Kim Claffy, and Mario Tokoro. Design, implementation, and evaluation of Virtual Internet Protocol. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 170–177, June 1992.
- [31] Fumio Teraoka and Keisuke Uehara. The virtual network protocol for host mobility. Internet Draft, April 1993. Work in progress.
- [32] Fumio Teraoka, Yasuhiko Yokote, and Mario Tokoro. A network architecture providing host migration transparency. In *Proceedings of the SIGCOMM '91 Conference: Communications Architectures & Protocols*, pages 209–220, September 1991.
- [33] Paul Turner. NetWare communications processes. *NetWare Application Notes*, Novell Research, pages 25–81, September 1990.
- [34] Hiromi Wada, Tatsuya Ohnishi, and Brian Marsh. Packet forwarding for mobile hosts. Internet Draft, November 1992. Work in progress.

LOCATION MANAGEMENT FOR NETWORKS WITH MOBILE USERS

B. R. Badrinath and Tomasz Imielinski

Department of Computer Science

Rutgers University, New Brunswick, NJ 08903

USA

ABSTRACT

Location management is one of the the most fundamental problems facing mobile networking today. How does the network know where the intended recipient of a message is currently located? Where should the information about the current location of a user be stored? Who should be responsible for determining user's location? These are examples of questions which location management attempts to address. Different location management schemes for networks such as Internet and analog voice networks such as cellular have been proposed. Unfortunately, the picture that has emerged so far is very confusing with several overlapping terms introduced that contribute to a ballooning terminology. It does not help of course that the problem is being actively addressed by two large and distinct communities: the Internet community and the cellular/PCN community with each group favoring its own terminology. The objective of this paper is to offer a general and hopefully a simplifying view of location management. We will try to explain a number of different approaches to location management using a small set of simple terms. We will show differences and similarities between different methods and generally discuss the most important performance parameters. We will also discuss future research directions.

1 INTRODUCTION

Location management is one the most fundamental problems facing mobile networking today. In this paper, we present an overview of location management schemes with an emphasis on the basic approaches used in cellular networks and the Internet. We will assume a general cellular architecture, where the system is viewed as a collection of cells, as discussed in the chapter 1. The

physical location of a mobile user is defined as the *cell* or the *Mobile Support Station* (MSS) under which the user currently resides. The coverage of a cell varies from a few miles, as in the current cellular networks, to a few meters, as proposed for the future Personal Communications Network (PCN). Additionally, each network user has a unique identity such as ID number (e.g., social security number), IP address, or terminal equipment number that enables the network to identify the user. This forms the *logical identifier* of a user.

The mapping between the logical identifier of a user and his physical location is called *binding*. If binding does not depend on time and is fixed, then the problem of routing information to the recipient can be done using standard routing techniques.¹ The need for location management occurs when binding changes over time.

As we have mentioned before, location management is actively studied in two different environments: the Internet environment and the cellular telephony environment. The Internet oriented work stems from the efforts of the MobileIP working group[15]. Work on MobileIP schemes deals with *connectionless*, packet oriented communication, while the cellular/PCN work deals with a *connection oriented* environment which typical for voice applications. In the cellular/PCN context the location of the recipient of a call is determined during the call set up (connection setup). Once the location of the recipient is established, the path of the route is fixed. In the Internet, initial packets sent to a mobile host may not be “aware” of the current location and hence take a detour, while subsequent packets may be routed in a more optimal way when the location is determined. There is no separation of location management and connection management in MobileIP, while such separation exists in cellular/PCN solutions.

The structure of cellular/PCN networks is typically hierarchical with a hierarchy of location servers situated “above” the MSS (base stations) Figure 1. The Internet does not typically conform to a hierarchical topology.²

Finally, in terms of actual experience, there is a huge discrepancy between the use of MobileIP and the use of location management schemes in cellular telephone networks. The cellular telephone system has been in operation for a number of years, with millions of mobile cellular phone users across the country. On the other hand, there are only “anecdotal” mobile user experiences

¹These include either IP routing or call set up methods used in standard telephone networks

²However, there has been a proposal to introduce border routers to handled location updates in Mobile IP[1]

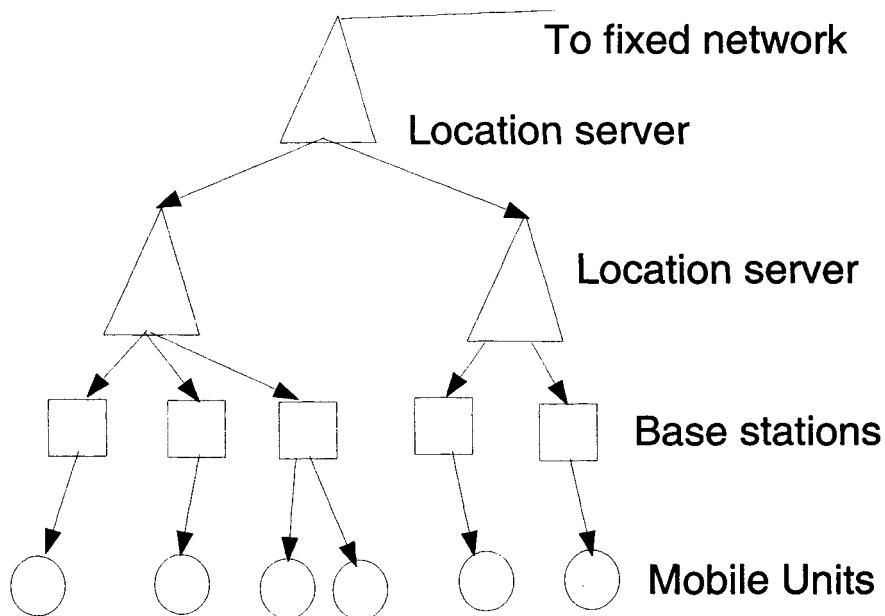


Figure 1 Structure of PCN architecture

in Internet³. There has been simply no practical experience with MobileIP proposals involving massive numbers of users, due to the lack of appropriate infrastructure supporting wireless and mobile data communication today. Moreover, as we elaborate later, there has been very little work reported on modeling performance of MobileIP proposals. Thus, one should treat the work on MobileIP as still being in its embryonic form.

One should also point out the fundamental philosophical difference that exists between the Internet and cellular/PCN communities. The Internet community essentially believes that the fixed network is "free" both in terms of bandwidth (which eventually will be unlimited) as well as switching costs. Hence, Internet based solutions typically express little concern over the actual network costs of location management. The Cellular/PCN community, on the other hand, is quite concerned about both switching cost and bandwidth overhead of location management on the fixed network[12]. While the optimistic predictions about the Internet may eventually turn out to be true, MobileIP researchers have yet to face the real world challenges of user mobility. Acceptable location management solutions should support massive numbers of users crossing the Lincoln tunnel during rush hour (at 4 P.M.), and not just a few professors and students moving between different campuses of a university!

The key problem of location management is maintaining the binding between the logical identifier and the physical location of the user. The following are the key issues:

- Where is the binding stored?
- Who is responsible for keeping the binding up to date?
- What happens if the binding is not available or is outdated?

The term *location directory* usually describes the directory used for storing this binding information. Storing the location directory faces typical dilemmas of distributed access: should the location directory be stored centrally or be distributed? Should the replication be dynamic or static? Solutions include maintaining one centralized location directory, a set of home directories stored at each user's *home location* (distributed without being replicated) and even location directories that are replicated[5, 15].

The responsibility of keeping the binding information up to date can fall either on the mobile user, who will send *location updates* to the location directory,

³This refers to early Columbia MobileIP users.

or on the sender, who may poll the user's location and update the location directory accordingly. This issue is strongly related to the resolution of another question: what happens when the location directory is not up to date? Two basic choices include declaring failure or performing search in order to determine the actual user location. Additionally, a combination of search and directory lookup can be used, as described in [14]. Below, we review some of the existing location management solutions starting with proposals for handling mobility in the Internet.

2 LOCATION MANAGEMENT IN THE INTERNET

Below, we establish a basic framework for location management inspired by the recent terminology used by Internet Engineering Task Force (IETF). We will subsequently use this framework to explain some of the existing MobileIP and cellular solutions. In the Internet, the identity of a user (IP address) normally reflects the physical location. This correspondence is obviously lost for mobile users. The problem of location management in the Internet is maintaining this correspondence for mobile users.

2.1 Sony's Virtual IP Proposal:

In this proposal[5, 6, 7], every mobile host has a virtual address and a physical address identified by the tuple $\langle VN, PN \rangle$. The virtual address, indicated by the VN part of the binding, is the permanent address of the mobile host. The physical address, indicated by the PN part of the binding, is acquired by the mobile host when it moves to a new network and stays the same as long as the host stays with in the same network. The physical address and the virtual address are identical for a host attached to the home network or if the physical address of the host is unknown.

The binding for mobile hosts is maintained at the home gateway of the mobile host. Binding is updated by the mobile whenever it obtains a new physical address. To send a packet, the sender uses the virtual address of the target host and thus normal IP routing suffice to relay the packets to the home gateway. The location directory at the home gateway is then used to determine the current physical address. If the binding information in the location directory indicates a PN that is different from VN, the packets are stamped with the

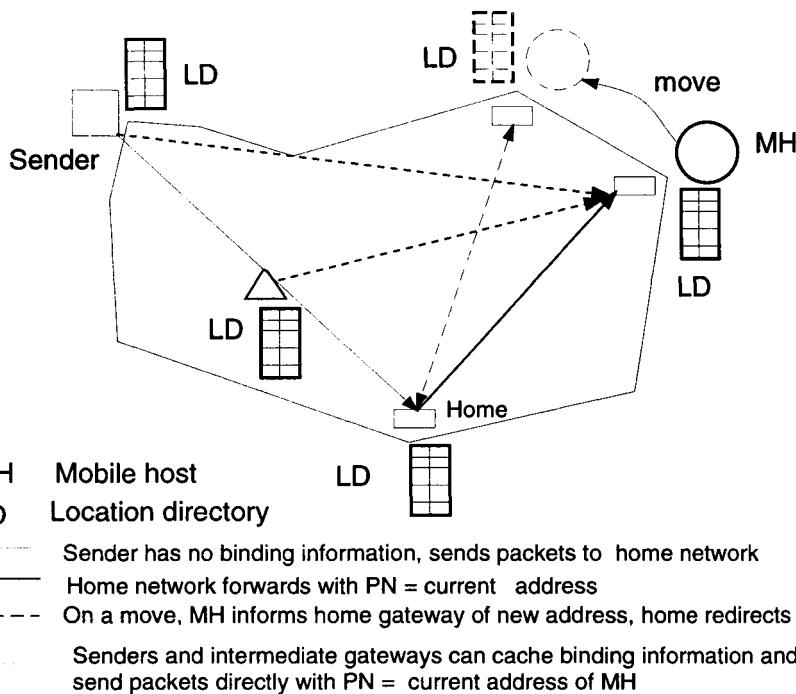


Figure 2 Location management in SONY VIP proposal

current PN and forwarded to the destination network of the mobile host. Figure 2 is a schematic that depicts various paths taken by packets under SONY's scheme.

Intermediate gateways that forward location updates directed from the mobile host to the home gateway can cache the binding as well. In this manner, binding information gets propagated to parts of the network that need information to route the packets to the mobile host. Once the binding information is available at a given location, the packets from that location to the mobile hosts are routed directly without having to go to the home gateway. Unlike the bindings in the home gateway, bindings in the intermediate gateways periodically expire or get invalidated by special location management packets.

To summarize, the SONY scheme stores binding at the home location of the given user and also replicates it dynamically along the route from the sender

to the target host. Notice that the physical location of the user in this scheme is not necessarily defined as the current cell but rather as the temporary IP address. This address need not correspond to the current cell or MSS.

2.2 IBM Proposal

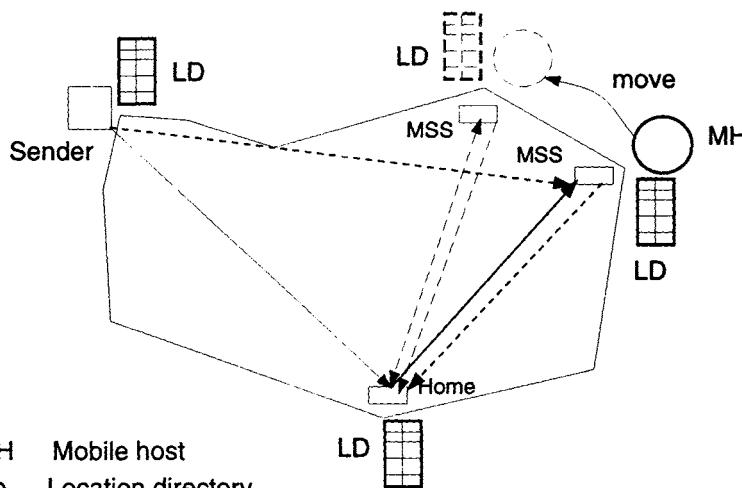
In this scheme[18], the physical location of the mobile host is defined as the current cell or, more precisely, as the IP address of its base station - MSS. The logical identifier is still the (permanent) IP address of the mobile user. The binding is defined as the mapping between the permanent IP address of the mobile host and the IP address of the current MSS. In this scheme the physical location corresponds to the IP address of the current MSS where as in the Sony scheme it is an acquired temporary IP address. Another difference is in how caching is done; in the Sony scheme, the binding information is cached by routers in the network while in IBM scheme, the binding information is cached by the senders. None of the two schemes uses search when binding information is not available or obsolete. Packets are simply dropped in this case.

The binding is kept in a location directory which is stored at the home gateway of the mobile host. Loose source routing (LSR option) is used for actual *redirection* of packets sent originally to the home gateway of the mobile host. When a sender uses the IP address of the target host, normal routing relays the packets to the mobile host's home gateway. The binding stored at the home gateway is then used to redirect. This is done by inserting the IP address of the mobile host's current MSS as the first hop in the LSR option.

Since a mobile host can also use redirection (LSR option) and reply to the sender, the current binding can be cached by the sender. Updating the binding information, as in Sony's VIP scheme, is the responsibility of the mobile host. A location update is sent every time the mobile host registers with a new MSS. Binding information is further replicated in the network via any reply packets received from the mobile host by its correspondent (sending) hosts. Figure 3 is a schematic that depicts various paths taken by packets under the IBM scheme.

2.3 Columbia Proposal

In this scheme[8], binding, relationship between the logical identity of the user and the physical location, is defined as in the IBM scheme. However, there is no concept of a home location directory. Rather, the location directory is



MH Mobile host
 LD Location directory

- Sender has no binding information, sends packets to home network
- Home MSS using LSR option redirects packets to current MSS
- MSS has no binding, return it to Home MSS
- MH updates Home MSS; Home MSS using LSR option redirects to new MSS
- Sender has binding information (use LSR option to send directly)

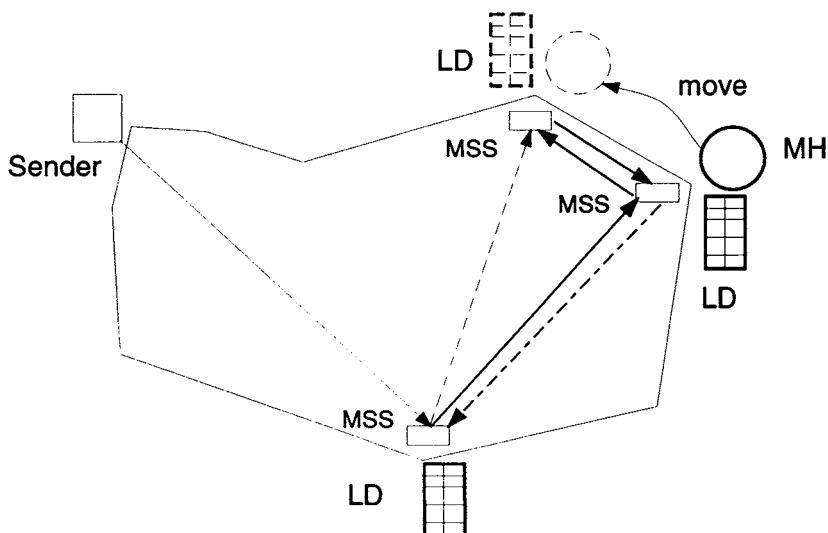
Figure 3 Location management in IBM proposal

maintained at several Mobile Support Stations (MSS) which cover the mobile subnet within a campus. MSSs advertise reachability to the mobile subnet and normal IP routing forwards the packets to one of several MSSs that constitutes the mobile subnet. The location directory at the MSS is then used to determine the MSS with which the mobile host is currently registered. The MSS that receives a packet encapsulates the packet with the IP address of the current MSS as the destination. This packet called the IP within IP packet is routed to the destination MSS. The destination MSS on receiving the packet decapsulates the IP within IP packet and forwards the inner packet to the mobile host.

When a mobile host moves and registers with a new MSS, it is his responsibility to inform the new MSS about its previous MSS. The new MSS then sends a forward pointer to the old MSS. The previous (old) MSS not only forwards any packets wrongly routed to it but also sends a message (sender redirect) to the MSS that wrongly forwarded the packet. This sender redirect message enables the MSS with stale binding to update the location directory with the new binding. It is also possible that a given MSS may not have any binding information for a particular mobile host. In this case, the MSS resorts to search; i.e., sends a message to all MSSs to check if the MH is registered. The MSS with which the MH is registered responds. This enables the MSS which initiated the search to insert a new entry in the location directory. Thus, Columbia Mobile IP scheme is unique in that it resorts to search as well as a directory look up to locate a mobile host. The binding information stored at the MSSs periodically expires unless it is refreshed by a packet or by the explicit search mechanism. Senders and intermediate gateways do not cache binding information.

In the case of Columbia MobileIP proposal, moves are classified as intra-campus moves and inter-campus moves. For the intra-campus moves, the logical identifier of the mobile host remains the same and only the binding (the current MSS with which a particular mobile host has registered) changes. For intercampus moves, the logical identifier of the mobile host changes and the mobile host acquires a new temporary address known as the *nonce* address. Packets are sent first to the home network of the recipient. The Home MSS then tunnels the packet to the foreign network(FN) using the new identity for the mobile host or the nonce address. Thus, it is possible for packets originating from a mobile host MH1 in a foreign network FN to another mobile host MH2 attached to the same network as MH1 to traverse to the home MSS of MH2 and then come back to the FN. Similarly, the reply packets from MH2 go to the home MSR of MH1 and are then tunneled back to the FN to which MH1 is attached.

Summarizing, in Columbia MobileIP proposal, there is no concept of a home location directory. Instead, a set of MSSs maintain the binding information. If



MH Mobile host

LD Location directory

— Sender has no binding information, sends packets to home network

— MSS receiving the packet forwards to current MSS

— On a move, new MSS informs old MSS, old MSS forwards packets

— Old MSS sends redirect to source MSS

— Source MSS forwards subsequent packets to new MSS

Figure 4 Location management in Columbia MobileIP proposal

binding information is not valid, search is used by the MSSs to find the up to date binding. Finally, the logical identity of the mobile user is local only to his campus and changes for intercampus, wide area moves. Figure 4 is a schematic that depicts various paths taken by packets under the IBM scheme.

2.4 IETF proposal

The definition of binding in the IETF proposal is the same as in the previously discussed MobileIP proposals. In the basic IETF proposal, each mobile host belongs to a home network and is assigned a home agent. The home agent is responsible for maintaining information about the current location of the mobile host and tunneling packets to that location. When a mobile host moves to a new network (called the foreign network), it can either acquire a temporary IP address (change identity) or the mobile host can register with a foreign agent (maintain identity). The basic IETF protocol allows for both these possibilities. The temporary address or the IP address of the foreign agent is known as the care-of-address and the home agent keeps track of the mobile host's care-of-address. The problem of location management is to maintain the mapping between the permanent address or identity and either the current care-of-address or the IP address of the foreign agent with which the mobile host has registered. The bindings in the location directory are of the form <IP, care-of-address>.

When a mobile host moves, either it registers with a new foreign agent or obtains a care-of-address (acquired locally through DHCP). When a mobile host registers with a foreign agent, the foreign agent sends the binding information <IP, IP address of foreign agent> to the home agent. The home agent inserts this binding information in the location directory. When a mobile host acquires a new care-of-address as opposed to registering with a foreign agent, the mobile host sends the binding information <IP, care-of-address> directly to the home agent. The binding information stored at the home agent periodically expires unless it is refreshed by the mobile host by re-registration. The lifetime of the registration is negotiated during registration and it is the responsibility of the mobile host to re-register before the binding expires.

Packets from a sender to a mobile host are initially routed to the home agent as the only known identity of the mobile host is an address that belongs to the home network. The binding information at the home agent is then used to tunnel the packets either to the foreign agent or directly to the mobile host using the care-of-address. In the latter, the mobile host acts as its own foreign

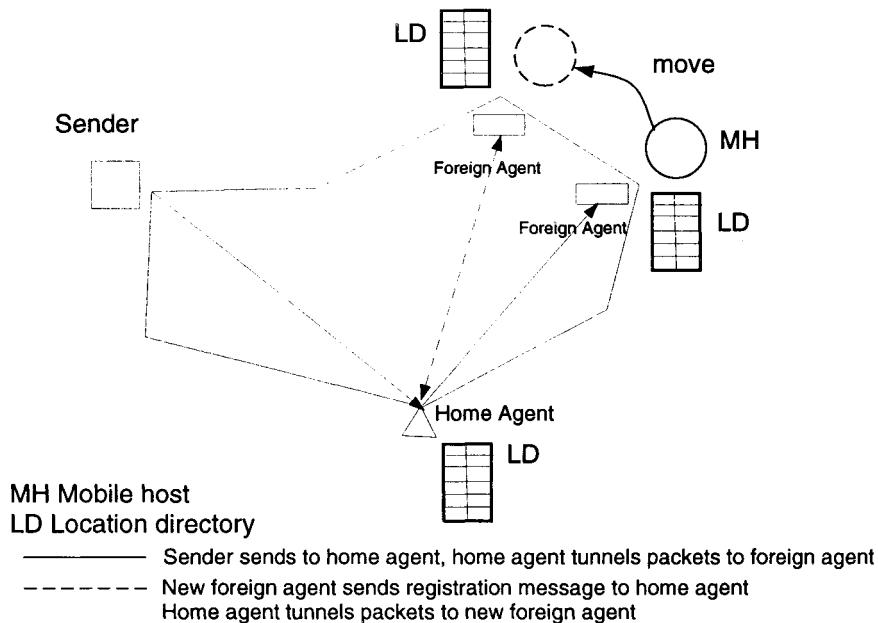


Figure 5 Location management in IETF proposal

agent. For proper location management, every time the mobile host registers with a new foreign agent or acquires a new address, updates to the binding have to be sent to the home agent. Figure 5 is a schematic that depicts various paths taken by packets under the IBM scheme.

There are a number of extensions to the basic IETF proposal that are being explored. These include route optimization and the use of cache agents. Route optimization allows a source to cache the binding information and tunnel packets directly to the mobile host. Cache agents (similar to the special gateways of Sony proposal) are allowed to store binding information and tunnel packets directly to the current location of the mobile host. The problem of how to maintain cache consistency, who should update the cache entries and on what basis should the updates be propagated are still being investigated.

The work on MobileIP is still in progress. So far little attention has been paid to issues of performance. In fact, there does not even exist a agreed upon performance model. There is very little experimental or simulation data. Particularly, it is not even clear how the different methods or the eventual MobileIP standard will scale to millions of users making wide area moves.

Proposal	Binding Information	Where is it stored?	How is binding information used?	How is consistency maintained?
SONY	<VN, PN>	at home gateway, source, and intermediate gateways	Use PN to route, Normally PN = VN, when away PN reflects current location	Explicit delete, or when inconsistency of mapping is detected
IBM	<IP, IP of MSS>	at home MR, source	Use IP of MSR as a loose source route	Explicit update
Columbia	<IP, IP of MSS>	at MSRs of the mobile subnet	Encapsulate using IP of MSR	Timer expiry and sender redirect
IETF	<IP, care-of-address> or <IP, IP of foreign agent>	at home agent at home agent and at foreign agent	Encapsulate using care-of-address Encapsulate using IP of foreign agent	Timeout and reregistration

Table 1 Comparison of location management in MobileIP proposals

Some argue that it is premature to talk about standard solutions when no real infrastructure exists and no experience is reported. We will discuss the performance issues and some of our concerns in more details later in the paper.

3 LOCATION MANAGEMENT IN CELLULAR TELEPHONE NETWORKS AND IN PCN

As we pointed out before, one of the main distinctions between cellular networks and Internet is that the cellular network services are connection oriented and the network has a hierarchical topology. Below, we briefly review location management schemes in the American Mobile Phone System (AMPS), Cellular Digital Packet Data (CDPD), and finally, proposals for the future Personal Communication Network (PCN).

Location management in AMPS

AMPS system configuration, for all practical purposes, is similar to the one discussed at the beginning of the paper. Currently, the average size of a cell is of the order of 1-2 miles in diameter[13]. Several base stations are connected to a switch also known as the Mobile Telephone Switching Office (MTSO). MTSOs are interconnected and connected to a public switched telephone network as well.

The logical identifier of the subscriber is defined by a unique terminal equipment number assigned to the cellular phone. As soon as the phone is turned on, it is registered with the MTSO for that area.

In order to find the location of the callee, base stations within the home area (home MTSO) are paged for the terminal equipment number. The appropriate cell then responds completing the connection establishment. Thus, the initial set up is based on search that may involve potentially a large number of base stations. As the mobile user moves from one base station to another, any active connection is automatically handed over to the new base station. If no call is in progress, no updates are needed for cell crossing. However, if the mobile user moves to a new area, then it has to re-register with a new MTSO. This process is called *roaming* and is still user initiated rather than being automated by the switching network. The new MTSO, then informs the home MTSO about the new location of the user.

Location management in CDPD

In CDPD, the mobile unit (called the mobile end system (MES)) communicates with a base station (known as mobile data base station (MDBS)). Several MDBSs are connected to a MDIS (mobile data intermediate system)[17].

The identity of the subscriber is defined by a unique terminal equipment number or an IP address assigned by the MDIS. As soon as the MES is turned on, the MES is registered with the MDIS for that area. Packets to the MES are first routed to the MDIS which in turn routes them to the MES via the current MDBS. As the mobile users move from one base station to another, the connection is automatically handed over to the new base station. Unlike in cellular telephony, where the base station is responsible for handoff, in CDPD, the MES initiates the handoff. When a mobile user moves to a new MDIS, it has to re-register with the new MDIS and the appropriate forwarding of packets from the old MDIS to the new one is provided.

Notice, that both AMPS and CDPD use a general concept of a “home agent” although, admittedly, in a rather primitive form. Additionally, CDPD uses the two-level hierarchical network topology. The advantages of hierarchical location management are further explored in the PCN proposals as described below.

Location management in PCN

The future Personal Communication Network is expected to have a high density of mobile subscribers moving across possibly very small cells (picocells). A central database will be very inefficient to handle the high volume of database traffic due to location updates. Similarly, searching for a user by paging all the base stations will not scale with number of base stations and the number of users. Hence, most of the architectures proposed for PCN including GSM (Global System for Mobile Telecommunications, chosen for PCN in Europe) use a set of databases that are distributed in a hierarchical network[2]. There are two basic databases called HLR (Home Location Register) and VLR (Visitor Location register) which store the location information of the mobile users. The HLR plays a role similar to that of the home agent in Mobile IP and VLR plays a role similar to that of the foreign agent in Mobile IP. A user who wants to register away from his home area has to register with the VLR and the VLR updates the HLR. A pointer to this VLR is maintained at the HLR to route incoming calls. Thus, the HLR has to be informed when the user registers with a new VLR. When a user enters a new location area, it registers in the VLR of the new location area and deregisters from the VLR of the old location area. With VLRs and HLRs, a more directed search for a mobile user can be initiated by paging just the base stations under the home database (HLR) or visitor database (VLR).

Several location management methods have been proposed for the PCN to take advantage of the hierarchical topology of the network. In this multilevel structure, leaves correspond to base stations (MSS) and the higher level nodes represent location servers. Instead of sending location updates on every move (crossing a base station), location updates are sent only when the mobile terminal enters a region covered by a different location server. Thus, these methods deliberately leave a (controlled) level of uncertainty about the user’s exact location. The imprecision is eventually eliminated by different types of locating algorithms or search strategies.

The two important decisions for a search strategy in the PCN are: i) where to begin the search and ii) how to progress? Depending upon these decisions, there are three general schemes described in [16].

1. *Flat*: this is an extreme approach. Starting from the root of the location server hierarchy, the entire network is searched. However, the search process is highly parallel as search at each level can be conducted concurrently. The latency of the search (time taken to locate) is equal to the height of the tree and the cost is equal to the number of children for the root.
2. *Expanding*: this is an incremental approach. First, the home location server is queried. If the user is not located here, then the parent of the home location server is queried, which in turn queries all its children and so on. Thus at each step, more number of location servers are queried until the user's current location server is found or all the location servers are queried. Since the search is done progressively, on an average, a fewer number of location servers are queried. The latency, however, is high because the search is carried out level by level in a sequential manner.
3. *Hybrid*: here also, the home location server is queried first. If the user is not found, then the parent of the home location server (in [16] a random location server is picked) initiates a search of all its children (except the home location server). If the desired user is not found here as well (within the vicinity of the home location server) then the search is initiated from the root, resulting in searching the entire network. The hybrid scheme can locate quickly the type of users who when not at the home location server happen to be found far away from it.

The question of how to locate users in PCN is also addressed in [14]. Here a hierarchical architecture based on the metropolitan area network (MAN) with four levels of databases is considered. At the lowest level there is a LAN of base stations. The other three levels are: access MAN, backbone MAN, and metropolitan MAN. The root is the metropolitan area (MA) database. There can be several metropolitan areas connected by a central office. Each subscriber has a home access MAN and when the user is outside the home MA, the home access MAN has a pointer to the general location of the user. To locate a callee, a ripple search scheme is proposed. Search always proceeds in a sequence of steps involving databases on different levels of the hierarchy. First, the search is confined to the hierarchy of the caller's local MA, and then directed to the caller's home MA. If this search sequence fails to locate the callee, only then is the directory database searched. The directory database contains information about the home access MAN for every user. Hence, this database is huge and access to it will be expensive. Once the home access MAN is found, the user can always be located. This scheme is based on the principle of call localization – “callee is located close to the caller.”

Other strategies for locating mobile users have been proposed in [3, 14]. Auerbach and Peleg [3] provide a general formal model for tracking mobile users. Their strategy is based on a hierarchy of regional directories, where each directory is based on a decomposition of the network into regions. On a move, only nearby directories are updated to point directly to the new address. In order to provide access to users who use remote directories, a forwarding pointer is left at the previous location, directing the search to the new location.

4 PERFORMANCE ISSUES

In this section we address performance evaluation of location management schemes. This is a complex problem because it involves several parameters.

In general, the aim of the location management is to minimize the overhead on the network traffic. This additional traffic is generated by location updates and associated control messages (such as pointer redirection etc.). Search messages also contribute to additional network traffic. To evaluate costs incurred in location management, two distinct environments that have to be treated differently are: the wireless environment between the mobile terminal and the local MSS, and the fixed network environment between the MSS and the final destination. The resources of the wireless link are definitely more precious than those of fixed network. Thus, it is important to reduce the size of the control packets which are transmitted on the wireless uplink channels by the mobile terminals. An additional constraint that plays an important role here is energy, since the uplink transmissions can be energy consuming. Thus, minimizing the number of location updates is important also from the perspective of saving energy.

The study reported in [12] analyzes the projected impact of location updates on the network traffic for the PCN. The extensive simulations in [12] show that the solutions based on VLR and HLR will not scale well when the number of mobile users will increase by an order of magnitude and when voice and data services will be provided. It is concluded that if location updates were to occur on each cell crossing, the resulting signaling load would have a major impact on the load of the network. This additional signaling traffic on the SS7 signaling system (capacity of 56kbps) is expected to be 4-11 times greater for cellular than for ISDN and 3-4 times greater for PCN than for cellular[12]. The signaling load due to updating alone contributes 70% additional load. Thus, location update will become a major bottleneck at the switches (such as SS7) and mechanisms to control the cost of location updates are needed.

On the other hand, saving on location updates contributes to increased cost due to network search. Whenever the location of a user is unknown, the network has to perform a search. Host mobility may further contribute to packet loss in the case when a foreign agent does not have an entry for the mobile destination host. Packets may suffer from additional latency as they may be redirected to the home agent, which then sends it to the new foreign agent. In an experiment conducted on our wireless testbed which consisted of three MSSs⁴ running Columbia Mobile IP, we found that the round trip delay for a packet from a mobile host to a fixed host and back was about 15 msec when the cache was valid and about 70 msec when a search was needed (when the cache has expired). While the search is in progress, any packets received at a MSS are dropped when the router does not have a valid entry to forward a packet. To determine the number of packets lost when the search is in progress, a sequence of packets was sent to an echo server running on the mobile, and the sequence number of the first packet to return was determined. With a expired cache, the first packet to return had a sequence number of 4 or 5. Thus, when the search was in progress, 3 to 4 packets were lost. In these experiments, packets were sent every 10 msec.

In general, the ratio of the search cost to the location update cost will influence the choice of the location management scheme. For example, in environments where search is very expensive, methods with frequent location updates will be favored. If the reverse is true, then schemes that perform search often will be preferred. *Latency* incurred in mobile networks is yet another important factor. How long does it take to establish a connection or send the first packet to a mobile host? If latency should be minimized then there should be minimal network search. Thus, methods with frequent location updates will be favored. The particular policy to be used will also be influenced by the relative frequency with which a particular user is “called” or sent messages. This depends on the usage profile and is discussed in more details in the next section.

Other performance parameters to consider include total size of the cache space which is necessary at routers. The number of database transactions per unit of time, necessary to update locations in location directories is also important. The large number of location updating transactions could be a limiting factor depending on how many concurrent transactions a database server can handle. Location management methods which “too eagerly” lose packets by giving up on finding a mobile host (for example, by not performing network search, in case there is no accurate binding information) can have a very negative effect on transport layer protocols[4]. Packet loss will contribute to decreasing

⁴All of them were 33 MHz 386 PC-ATs

throughput⁵. Lack of exact location information may result in non-optimal routing of packets. As the location information is updated, packets may traverse along a more optimal route. Thus, mobility affects the round trip delay of packets. This, in turn, may result in spurious retransmissions at the transport layer. Also, applications that are designed to handle a particular loss rate may fail if this expected loss rate is exceeded due to dropped packets for lack of exact location information.

It is important to perform simulation studies and eventually real experiments with various mobile IP location management schemes to measure their performance. In particular, the impact of location updates has to be carefully evaluated for different MobileIP proposals.

In the next section we describe how location management schemes can benefit from incorporating mobility profiles of users.

5 FUTURE: ADAPTIVE LOCATION MANAGEMENT

Location management can greatly benefit from the knowledge of user mobility patterns, namely from:

- Knowing where the receivers are

This information describes locations where a given user is most likely to be found at a given time. Notice that the mobility patterns are very effectively used by humans today - we usually know (have a "built in" heuristic) when a callee is most likely to be found at home, and when s/he is expected to be at work. This information affects the order in which we attempt to contact a callee. We would like these functions to be performed automatically by the network rather than by a user.

- Knowing where the usual senders are: spatial distribution of the likely callers.

This information helps in deciding how to distribute the location information about a particular user. The general rule should be to inform the most likely callers (senders) most often about location changes. The set of callers to inform can be determined by the cost of traffic generated by the

⁵Also it may unnecessarily trigger the exponential backoff feature of the TCP

need to send location updates relative to the cost of establishing a connection without knowing the callee's location[19]. A caller that makes a lot of calls to a mobile user should be informed about the current location to reduce the overall cost due to location management. If user's mobility pattern is known, the best idea is to inform only upon *exceptional* moves.

Other solutions involve adaptively reducing the number of location updates by not informing about every single change in location but rather maintaining *incomplete information* about the location of the user. But when should we inform? The answer depends on the *mobility profile* of the user:

Example

Figure 6 shows a hypothetical daily routine of a commuter who drives between the work and home areas once a day and moves within each of the areas as well. The following options can be considered:

1. The network has to page (page is the term used for broadcasting the address of the user in order to find him) the commuter across both work and home areas every time a call (message) is made (sent) to the commuter.
The cost would be 7 paging messages per call since there are 7 base stations in the two areas.
2. The commuter informs the network every time he moves between any two locations.
The cost would be 8 updates for work moves (assuming a 8 hour stay in the campus area) and 8 updates for home moves (assuming a 16 hour stay in the home area). The cost of a call would be 1 paging message.
3. The commuter informs the network only when he moves from the campus area to home and back.
The cost would be 2 updates, 1 for each crossing and the call cost is either 3 or 4 paging messages depending on commuter's location at the time of the call.

It is clear that the last strategy, assuming the mobility pattern just described, is superior to the first and second strategies⁶. In general, any given user could have a *partition* associated with his profile to reduce the overall volume of

⁶Assuming that the paging cost and the update cost are the same

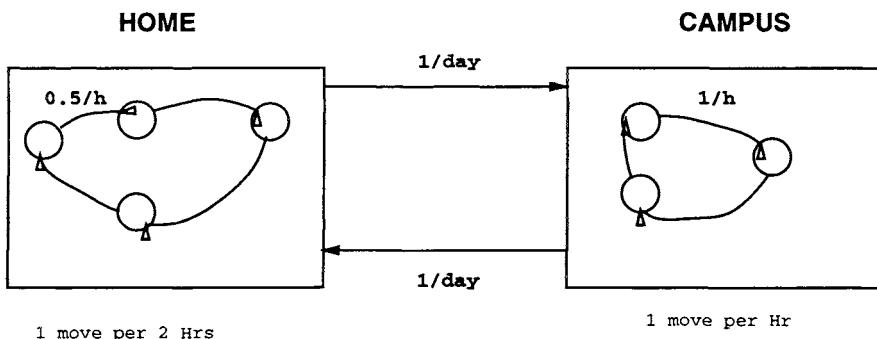


Figure 6 User Profile

messages (including paging messages and location updates). Partitions can be defined both at the global and local level. At the global level, partitions will consist of location servers. At the local level, partitions will consist of base stations. In our example we have assumed that both home and campus areas are located under a common location server. Therefore, the partitions described here are *local*. If, campus and home areas were located under different location servers, then the partition described in the example would be considered *global*.

In [10], we provide a more extensive discussion and experimental results demonstrating usefulness of partitions. In particular, given the mobility pattern of a user, we show how to come up with an “optimal” partition with respect to the overall cost of messages.

Another important characteristic of the user is what we term the *call to mobility* ratio⁷. The call to mobility ratio is computed by dividing the average number of calls made to the user by the average number of cell crossings, the user makes in a given period of time. Two possible strategies can be applied to locate users: paging (page all the base stations under a location server) or pointer forwarding (contacting a known base station and then following pointers that are updated by the moving user). Which among the two is better depends on the ratio between the number of calls and the number of moves made by the user. For low call to mobility ratios, the paging scheme is beneficial compared to the pointer forwarding scheme. Indeed, if the user is not called often, then he unnecessarily pays for location updates (pointer updates). The pointer forwarding scheme

⁷This discussion assumes the connection oriented, cellular model but can be restated for the connection free environment by replacing the “call” by a “message.”

becomes beneficial for high call to mobility ratios. In such a case, the cost of location updates is “amortized” over the large number of calls - virtually eliminating all search cost in this case.

This short discussion clearly demonstrates that there is no single “optimal” location strategy for all users. For example, depending on the call/mobility ratio, either the paging or pointer forwarding method will be beneficial for a given user. Similarly, different users will be associated with different partitions, leading to different location update strategies. Since mobility is clearly a function of time, more sophisticated schemes will involve time-dependent characteristics of the profiles.

6 CONCLUSIONS

The objective of this paper was to offer a general, and hopefully simplifying, view of location management in cellular telephone networks and in the Internet. We have explained a number of different approaches to location management using a small set of simple terms. We have shown differences and similarities between different methods. Finally we have discussed informally the performance issues with a discussion of future research directions in location management.

REFERENCES

- [1] A. Aziz, “A scalable and efficient intra-domain tunneling mobile-ip scheme,” ACM SIGCOMM, Computers Communication Review, January 1994.
- [2] Jan A. Audestad, “GSM general overview of network functions,” In Proceedings of the International conference on digital land mobile radio communications, Venice 1987.
- [3] Baruch Awerbuch and David Peleg, “Concurrent online tracking of mobile users”, Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols, October 1991.
- [4] R. Caceres and L. Iftode, “The effects of mobility on reliable transport protocols,” In Proceedings of the 14th International Conference on Distributed Computing Systems, June 1994, pp. 12–20.

- [5] Fumio Teraoka, Yasuhiko Yokote, and Mario Tokoro "A network architecture providing host migration transparency," In proceedings of the ACM SIGCOMM, September 1991.
- [6] Fumio Teraoka, Kim Claffy, and Mario Tokoro "Design, implementation, and evaluation of virtual internet protocol," In proceedings of the 12 ICDCS, June 1992, pp. 170–177.
- [7] Fumio Teraoka, "Mobile IP: The VIP approach," IETF presentation, Boston, July 1992.
- [8] J. Ioannidis and G. Maguire, "The design and implementation of a mobile internetworking architecture," In USENIX Winter 1993 technical conference January 1993.
- [9] Andrew Myles and David Skellern, Comparison of Mobile Host Protocols for IP. Internetworking: research and experience, Vol. 4, pp. 175–194, 1993.
- [10] T. Imielinski and B. R. Badrinath, "Querying in Highly mobile distributed environments," In the Proceedings of the 18th VLDB, August 1992, pp. 41–52.
- [11] Y. Rekhter and C. Perkins, "Optimal routing for mobile hosts using IP's loose source route option," Internet Draft, October 1992.
- [12] Kathleen S. Meier-Hellstern, Eduardo Alonso, and Douglas Opiel, "The Use of SS7 and GSM to support high density personal communications," Third Winlab workshop on third generation wireless information networks, April 1992, pp. 49–57.
- [13] William Lee "Mobile cellular Telecommunication systems," McGraw-Hill, 1989.
- [14] L J Ng, R. W. Donaldson, and A. D. Malyan, "Distributed architectures and databases for intelligent personal communication networks," Proc of the ICWC, June 1992.
- [15] David Johnson, "Wireless and mobile networking," Tutorial notes, 15th ICDCS, May 1995.
- [16] David J. Goodman and Binay Sugla, "Signalling system draft," Unpublished manuscript.
- [17] Frank Quick and Kumar Balachandran, "An overview of cellular packet data (CDPD) system, in The fourth international symposium on personal, indoor and mobile radio communications, Yokohama, Japan, September 1993, pp. E 2.7.1 – E 2.7.

- [18] C. Perkins, P. Bhagwat, "A mobile networking system based on internet protocol," IEEE personal communications, Vol. 1, No. 1, pp 32—41, January 1994.
- [19] S. Rajagopalan and B. R. Badrinath, "An adaptive location management strategy for Mobile IP," First Intl. Conf. on Mobile Computing and Networking, Nov. 1995.

DYNAMIC SOURCE ROUTING IN AD HOC WIRELESS NETWORKS

David B. Johnson and David A. Maltz

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891
USA

ABSTRACT

An *ad hoc* network is a collection of wireless mobile hosts forming a temporary network without the aid of any established infrastructure or centralized administration. In such an environment, it may be necessary for one mobile host to enlist the aid of other hosts in forwarding a packet to its destination, due to the limited range of each mobile host's wireless transmissions. This paper presents a protocol for routing in ad hoc networks that uses *dynamic source routing*. The protocol adapts quickly to routing changes when host movement is frequent, yet requires little or no overhead during periods in which hosts move less frequently. Based on results from a packet-level simulation of mobile hosts operating in an ad hoc network, the protocol performs well over a variety of environmental conditions such as host density and movement rates. For all but the highest rates of host movement simulated, the overhead of the protocol is quite low, falling to just 1% of total data packets transmitted for moderate movement rates in a network of 24 mobile hosts. In all cases, the difference in length between the routes used and the optimal route lengths is negligible, and in most cases, route lengths are on average within a factor of 1.01 of optimal.

1 INTRODUCTION

Mobile hosts and wireless networking hardware are becoming widely available, and extensive work has been done recently in integrating these elements into traditional networks such as the Internet. Oftentimes, however, mobile users will want to communicate in situations in which no fixed wired infrastructure such as this is available, either because it may not be economically practical

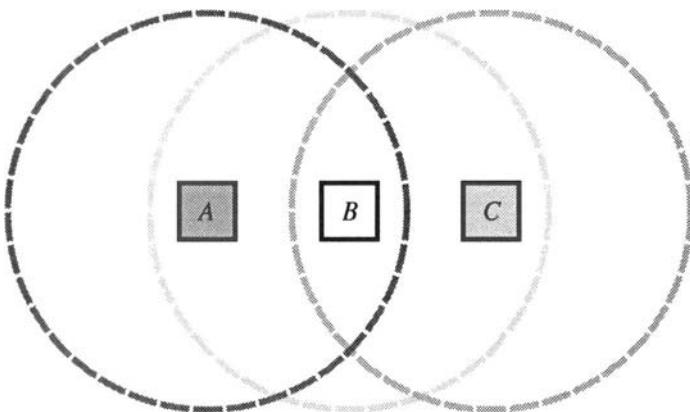


Figure 1 A simple ad hoc network of three wireless mobile hosts

or physically possible to provide the necessary infrastructure or because the expediency of the situation does not permit its installation. For example, a class of students may need to interact during a lecture, friends or business associates may run into each other in an airport terminal and wish to share files, or a group of emergency rescue workers may need to be quickly deployed after an earthquake or flood. In such situations, a collection of mobile hosts with wireless network interfaces may form a temporary network without the aid of any established infrastructure or centralized administration. This type of wireless network is known as an *ad hoc network*.

If only two hosts, located closely together, are involved in the ad hoc network, no real routing protocol or routing decisions are necessary. In many ad hoc networks, though, two hosts that want to communicate may not be within wireless transmission range of each other, but could communicate if other hosts between them also participating in the ad hoc network are willing to forward packets for them. For example, in the network illustrated in Figure 1, mobile host C is not within the range of host A's wireless transmitter (indicated by the circle around A) and host A is not within the range of host C's wireless transmitter. If A and C wish to exchange packets, they may in this case enlist the services of host B to forward packets for them, since B is within the overlap between A's range and C's range. Indeed, the routing problem in a real ad hoc network may be more complicated than this example suggests, due to the inherent nonuniform propagation characteristics of wireless transmissions and due to the possibility that any or all of the hosts involved may move at any time.

Routing protocols in conventional wired networks generally use either *distance vector* or *link state* routing algorithms, both of which require periodic routing advertisements to be broadcast by each router. In distance vector routing [9, 17, 26, 27, 29], each router broadcasts to each of its neighbor routers its view of the distance to all hosts, and each router computes the shortest path to each host based on the information advertised by each of its neighbors. In link state routing [10, 16, 18], each router instead broadcasts to all other routers in the network its view of the status of each of its adjacent network links, and each router then computes the shortest distance to each host based on the complete picture of the network formed from the most recent link information from all routers. In addition to its use in wired networks, the basic distance vector algorithm has also been adapted for routing in wireless ad hoc networks, essentially treating each mobile host as a router [11, 19, 25].

This paper describes the design and performance of a routing protocol for ad hoc networks that instead uses *dynamic source routing* of packets between hosts that want to communicate. Source routing is a routing technique in which the sender of a packet determines the complete sequence of nodes through which to forward the packet; the sender explicitly lists this route in the packet's header, identifying each forwarding "hop" by the address of the next node to which to transmit the packet on its way to the destination host. Source routing has been used in a number of contexts for routing in wired networks, using either statically defined or dynamically constructed source routes [4, 5, 12, 20, 22, 28], and has been used with statically configured routes in the Tucson Amateur Packet Radio (TAPR) work for routing in a wireless network [14]. The protocol presented here is explicitly designed for use in the wireless environment of an ad hoc network. There are no periodic router advertisements in the protocol. Instead, when a host needs a route to another host, it dynamically determines one based on cached information and on the results of a *route discovery* protocol.

We believe our dynamic source routing protocol offers a number of potential advantages over conventional routing protocols such as distance vector in an ad hoc network. First, unlike conventional routing protocols, our protocol uses no periodic routing advertisement messages, thereby reducing network bandwidth overhead, particularly during periods when little or no significant host movement is taking place. Battery power is also conserved on the mobile hosts, both by not sending the advertisements and by not needing to receive them (since a host could otherwise reduce its power usage by putting itself into "sleep" or "standby" mode when not busy with other tasks). Distance vector and link state routing, on the other hand, must continue to send advertisements even when nothing changes, so that other mobile hosts will continue

to consider those routes or network links as valid. In addition, many of the “links” between routers seen by the routing algorithm may be redundant [11]. Wired networks are usually explicitly configured to have only one (or a small number) of routers connecting any two networks, but there are no explicit links in an ad hoc network, and all communication is by broadcast transmissions. The redundant paths in a wireless environment unnecessarily increase the size of routing updates that must be sent over the network, and increase the CPU overhead required to process each update and to compute new routes.

In addition, conventional routing protocols based on link state or distance vector algorithms may compute some routes that do not work. In a wireless environment, network transmission between two hosts does not necessarily work equally well in both directions, due to differing propagation or interference patterns around the two hosts [1, 15]. For example, with distance vector routing, even though a host may receive a routing advertisement from another mobile host, packets it might then transmit back to that host for forwarding may not be able to reach it. Our protocol does not require transmissions between hosts to work bidirectionally, although we do make use of it when afforded, for example, by MAC-level protocols such as MACA [13] or MACAW [2] that ensure it.

Finally, conventional routing protocols are not designed for the type of dynamic topology changes that may be present in ad hoc networks. In conventional networks, links between routers occasionally go down or come up, and sometimes the cost of a link may change due to congestion, but routers do not generally move around dynamically. In an environment with mobile hosts as routers, though, convergence to new, stable routes after such dynamic changes in network topology may be slow, particularly with distance vector algorithms [20]. Our dynamic source routing protocol is able to adapt quickly to changes such as host movement, yet requires no routing protocol overhead during periods in which such changes do not occur.

Section 2 of this paper details our assumptions about the network and the mobile hosts. The basic operation of our dynamic source routing protocol is described in Section 3, and optimizations to this basic operation are described in Section 4. In Section 5, we present a preliminary evaluation of the performance of our protocol, based on a packet-level simulation. In Section 6, we discuss related protocols for wireless network routing and for source routing, and in Section 7, we present conclusions and future work.

2 ASSUMPTIONS

We assume that all hosts wishing to communicate with other hosts within the ad hoc network are willing to participate fully in the protocols of the network. In particular, each host participating in the network should also be willing to forward packets for other hosts in the network.

We refer to the number of hops necessary for a packet to reach from any host located at one extreme edge of the network to another host located at the opposite extreme, as the *diameter* of the network. For example, the diameter of the ad hoc network depicted in Figure 1 is two. We assume that the diameter of an ad hoc network will be small but may often be greater than one.

Hosts within the ad hoc network may move at any time without notice, but we assume that the speed with which hosts move is moderate with respect to the packet transmission latency and wireless transmission range of the particular underlying network hardware in use. In particular, we assume that hosts do not continuously move so rapidly as to make the flooding of every packet the only possible routing protocol.

We assume that hosts can enable a *promiscuous* receive mode on their wireless network interface hardware, causing the hardware to deliver every received packet to the network driver software without filtering based on destination address. Although we do not require this facility, it is common in current LAN hardware for broadcast media including wireless, and some of our optimizations take advantage of it. Use of promiscuous mode does increase the software overhead on the CPU, but we believe that wireless network speeds are more the inherent limiting factor to performance in current and future systems. We believe that portions of the protocol are also suitable for implementation directly in hardware or within a programmable network interface unit to avoid this overhead on the CPU.

3 BASIC OPERATION

3.1 Overview

To send a packet to another host, the sender constructs a *source route* in the packet's header, giving the address of each host in the network through which the packet should be forwarded in order to reach the destination host. The

sender then transmits the packet over its wireless network interface to the first hop identified in the source route. When a host receives a packet, if this host is not the final destination of the packet, it simply transmits the packet to the next hop identified in the source route in the packet's header. Once the packet reaches its final destination, the packet is delivered to the network layer software on that host.

Each mobile host participating in the ad hoc network maintains a *route cache* in which it caches source routes that it has learned. When one host sends a packet to another host, the sender first checks its route cache for a source route to the destination. If a route is found, the sender uses this route to transmit the packet. If no route is found, the sender may attempt to discover one using the *route discovery* protocol. While waiting for the route discovery to complete, the host may continue normal processing and may send and receive packets with other hosts. The host may buffer the original packet in order to transmit it once the route is learned from route discovery, or it may discard the packet, relying on higher-layer protocol software to retransmit the packet if needed. Each entry in the route cache has associated with it an expiration period, after which the entry is deleted from the cache.

While a host is using any source route, it monitors the continued correct operation of that route. For example, if the sender, the destination, or any of the other hosts named as hops along a route move out of wireless transmission range of the next or previous hop along the route, the route can no longer be used to reach the destination. A route will also no longer work if any of the hosts along the route should fail or be powered off. This monitoring of the correct operation of a route in use we call *route maintenance*. When route maintenance detects a problem with a route in use, route discovery may be used again to discover a new, correct route to the destination.

This section describes the basic operation of route discovery and route maintenance. Optimizations to this basic operation of the protocol are then described in Section 4.

3.2 Route Discovery

Route discovery allows any host in the ad hoc network to dynamically discover a route to any other host in the ad hoc network, whether directly reachable within wireless transmission range or reachable through one or more intermediate network hops through other hosts. A host initiating a route discovery broadcasts

a *route request* packet which may be received by those hosts within wireless transmission range of it. The route request packet identifies the host, referred to as the *target* of the route discovery, for which the route is requested. If the route discovery is successful the initiating host receives a *route reply* packet listing a sequence of network hops through which it may reach the target.

In addition to the address of the original initiator of the request and the target of the request, each route request packet contains a *route record*, in which is accumulated a record of the sequence of hops taken by the route request packet as it is propagated through the ad hoc network during this route discovery. Each route request packet also contains a unique *request id*, set by the initiator from a locally-maintained sequence number. In order to detect duplicate route requests received, each host in the ad hoc network maintains a list of the \langle initiator address, request id \rangle pairs that it has recently received on any route request.

When any host receives a route request packet, it processes the request according to the following steps:

1. If the pair \langle initiator address, request id \rangle for this route request is found in this host's list of recently seen requests, then discard the route request packet and do not process it further.
2. Otherwise, if this host's address is already listed in the route record in the request, then discard the route request packet and do not process it further.
3. Otherwise, if the target of the request matches this host's own address, then the route record in the packet contains the route by which the request reached this host from the initiator of the route request. Return a copy of this route in a *route reply* packet to the initiator.
4. Otherwise, append this host's own address to the route record in the route request packet, and re-broadcast the request.

The route request thus propagates through the ad hoc network until it reaches the target host, which then replies to the initiator. The original route request packet is received only by those hosts within wireless transmission range of the initiating host, and each of these hosts propagates the request if it is not the target and if the request does not appear to this host to be redundant. Discarding the request because the host's address is already listed in the route record guarantees that no single copy of the request can propagate around a

loop. Also discarding the request when the host has recently seen one with the same \langle initiator address, request id \rangle removes later copies of the request that arrive at this host by a different route.

In order to return the *route reply* packet to the initiator of the route discovery, the target host must have a route to the initiator. If the target has an entry for this destination in its route cache, then it may send the route reply packet using this route in the same way as is used in sending any other packet (Section 3.1). Otherwise, the target may reverse the route in the route record from the route request packet, and use this route to send the route reply packet. This, however, requires the wireless network communication between each of these pairs of hosts to work equally well in both directions, which may not be true in some environments or with some MAC-level protocols. An alternative approach, and the one we have currently adopted, is for this host to piggyback the route reply packet on a route request targeted at the initiator of the route discovery to which it is replying. This use of piggybacking is described in Section 4.2.

3.3 Route Maintenance

Conventional routing protocols integrate route discovery with route maintenance by continuously sending periodic routing updates. If the status of a link or router changes, the periodic updates will eventually reflect the changes to all other routers, presumably resulting in the computation of new routes. However, using route discovery, there are no periodic messages of any kind from any of the mobile hosts. Instead, while a route is in use, the route maintenance procedure monitors the operation of the route and informs the sender of any routing errors.

Since wireless networks are inherently less reliable than wired networks [1], many wireless networks utilize a hop-by-hop acknowledgement at the data link level in order to provide early detection and retransmission of lost or corrupted packets. In these networks, route maintenance can be easily provided, since at each hop, the host transmitting the packet for that hop can determine if that hop of the route is still working. If the data link level reports a transmission problem for which it cannot recover (for example, because the maximum number of retransmissions it is willing to attempt has been exceeded), this host sends a *route error* packet to the original sender of the packet encountering the error. The route error packet contains the addresses of the hosts at both ends of the hop in error: the host that detected the error and the host to which it was attempting to transmit the packet on this hop. When a route error packet

is received, the hop in error is removed from this host's route cache, and all routes which contain this hop must be truncated at that point.

If the wireless network does not support such lower-level acknowledgements, an equivalent acknowledgement signal may be available in many environments. After sending a packet to the next hop mobile host, the sender may be able to hear that host transmitting the packet again, on its way further along the path, if it can operate its wireless network interface in promiscuous mode. For example, in Figure 1, host *A* may be able to hear *B*'s transmission of the packet on to *C*. This type of acknowledgement is known as a *passive acknowledgement* [11]. In addition, existing transport or application level replies or acknowledgements from the original destination could also be used as an acknowledgement that the route (or that hop of the route) is still working. As a last resort, a bit in the packet header could be included to allow a host transmitting a packet to request an explicit acknowledgement from the next-hop receiver. If no other acknowledgement signal has been received in some time from the next hop on some route, the host could use this bit to inexpensively probe the status of this hop on the route.

As with the return of a route reply packet, a host must have a route to the sender of the original packet in order to return a route error packet to it. If this host has an entry for the original sender in its route cache, it may send the route error packet using that route. Otherwise, this host may reverse the route from the packet in error (the route by which the packet reached this host) or may use piggybacking as in the case of a route reply packet (Section 4.2). Another option in the case of returning a route error packet is for this host to save the route error packet locally in a buffer, perform a route discovery for the original sender, and then send the route error packet using that route when it receives the route reply for this route discovery. This option cannot be used for returning a route reply packet, however, since then neither host would ever be able to complete a route discovery for the other, if neither initially had a route cache entry for the other.

Route maintenance can also be performed using end-to-end acknowledgements rather than the hop-by-hop acknowledgements described above, if the particular wireless network interfaces or the environment in which they are used are such that wireless transmissions between two hosts do not work equally well in both directions. As long as some route exists by which the two end hosts can communicate (perhaps different routes in each direction), route maintenance is possible. In this case, existing transport or application level replies or acknowledgements from the original destination, or explicitly requested network level acknowledgements, may be used to indicate the status of this host's route to

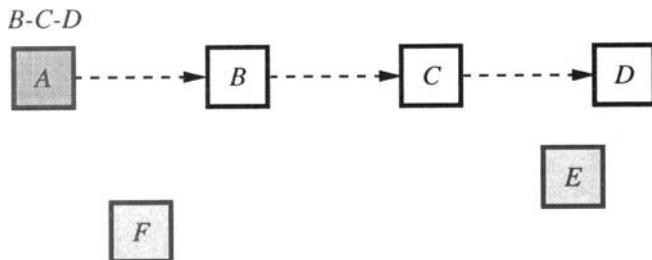


Figure 2 An example ad hoc network illustrating use of the route cache

the other host. With hop-by-hop acknowledgements, the particular hop in error is indicated in the route error packet, but with end-to-end acknowledgements, the sender may only assume that the last hop of the route to this destination is in error.

4 OPTIMIZATIONS

A number of optimizations are possible to the basic operation of route discovery and route maintenance as described in Section 3.2, that can reduce the number of overhead packets and can improve the average efficiency of the routes used on data packets. This section discusses some of those optimizations.

4.1 Full Use of the Route Cache

The data in a host's route cache may be stored in any format, but the active routes in its cache in effect form a tree of routes, rooted at this host, to other hosts in the ad hoc network. For example, Figure 2 shows an ad hoc network of five mobile hosts, in which mobile host *A* has earlier completed a route discovery for mobile host *D* and has cached the route to *D* through *B* and *C*. Since hosts *B* and *C* are on the route to *D*, host *A* also learns the route to both of these hosts from its route discovery for *D*. If *A* later performs a route discovery and learns the route to *E* through *B* and *C*, it can represent this in its route cache with the addition of the single new hop from *C* to *E*. If *A* then learns it can reach *C* in a single hop (without needing to go through *B*), *A*

can use this new route to C to also shorten the routes to D and E in its route cache.

A host can add entries to its route cache any time it learns a new route. In particular, when a host forwards a data packet as an intermediate hop on the route in that packet, the forwarding host is able to observe the entire route in the packet. Thus, for example, when host B forwards packets from A to D , B can add the route information from that packet to its own route cache. If a host forwards a route reply packet, it can also add the route information from the route record being returned in that route reply, to its own route cache. Finally, since all wireless network transmissions are inherently broadcast, a host may be able configure its network interface into promiscuous receive mode, and can then add to its route cache the route information from any data or route reply packet it can overhear.

A host may use its route cache to avoid propagating a route request packet received from another host. In particular, suppose a host receives a route request packet for which it is not the target and is not already listed in the route record in the packet, and for which the pair (initiator address, request id) is not found in its list of recently seen requests; if the host has a route cache entry for the target of the request, it may append this cached route to the accumulated route record in the packet, and may return this route in a route reply packet to the initiator without propagating (re-broadcasting) the route request. Thus, for example, if mobile host F needs to send a packet to mobile host D , it will initiate a route discovery and broadcast a route request packet. If this broadcast is received by A , A can simply return a route reply packet to F containing the complete route to D consisting of the sequence of hops A , B , C , and D .

A particular problem can occur, however, when several mobile hosts receive the initiator's broadcast of the route request packet, and all reply based on routes found in their route caches. In Figure 2, for example, if both A and B receive F 's route request broadcast, they will both be able to reply from their route caches, and will both send their replies at about the same time since they both received the broadcast at about the same time. Particularly when more than the two mobile hosts in this example are involved, these simultaneous replies from the mobile hosts receiving the broadcast may create packet collisions among some or all of these replies and may cause local congestion in the wireless network. In addition, it will often be the case that the different replies will indicate routes of different lengths. For example, A 's reply will indicate a route to D that is one hop longer than that in B 's reply.

We avoid the problems of many simultaneous replies and attempt to eliminate replies indicating routes longer than the shortest reply, by causing each mobile host to delay slightly before replying from its cache. Before replying from its route cache, a host performs the following actions:

1. Pick a delay period $d = H \times (h - 1 + r)$, where h is the length in number of network hops for the route to be returned in this host's reply, r is a random number between 0 and 1, and H is a small constant delay to be introduced per hop.
2. Delay transmitting the route reply from this host for a period of d .
3. Within this delay period, promiscuously receive all packets at this host. If a packet is received by this host during the delay period addressed to the target of this route discovery (the target is the final destination address for the packet, through any sequence of intermediate hops), and if the length of the route on this packet is less than h , then cancel the delay and do not transmit the route reply from this host; this host may infer that the initiator of this route discovery has already received a route reply, giving an equal or better route.

Another problem that can occur when hosts reply to route requests from their cache, is the formation of a loop in the route sent in the route reply packet. The route record in the route request cannot contain a loop, and no entry in a route cache ever is set to a route containing a loop, yet the concatenation of the route record and the entry from the replying host's route cache for the target may contain a loop. For example, in Figure 2, if host B does not have a route cache entry for D , it will need to initiate a route discovery before sending a packet to D . In this case, A could immediately reply from its route cache with the route to D through B and C . This, however, would form the concatenated route of $A-B-C-D$ for B to then use in sending packets to D , creating a loop from B to A and then back to B . In order to avoid this problem, if a host receives a route request and is not the target of the request but could reply from its cache, the host instead discards the request if the route in its reply would contain a loop; this restriction also implies that a host will only reply from its cache with a route in which the host itself is on the route, at the end of the route recorded in the route request packet and at the beginning of the path obtained from the host's route cache.

As a last optimization involving full use of the route cache, we have added the ability for the initiator of a route request to specify in the request packet, the

maximum number of hops over which the packet may be propagated. If another host near the initiator has a cache entry for the target of a route request, the propagation of many redundant copies of the route request can be avoided if the initiator can explicitly limit the request's propagation when it is originally sent. Currently, we use this ability during route discovery as follows:

1. To perform a route discovery, initially send the route request with a hop limit of one. We refer to this as a nonpropagating route request.
2. If no route reply is received from this route request after a small timeout period, send a new route request with the hop limit set to a predefined "maximum" value for which it is assumed that all useful routes in the ad hoc network are less than this limit (currently 10).

This procedure uses the hop limit on the route request packet to inexpensively check if the target is currently within wireless transmitter range of the initiator or if another host within range has a route cache entry for this target (effectively using the caches of this host's neighbors as an extension of its own cache). Since the initial request is limited to one network hop, the timeout period before sending the propagating request can be quite small. This mechanism could also be used to implement an "expanding ring" search for the target, in which the hop limit is gradually increased in subsequent retransmissions of the route request for this target, but we have not yet experimented with this approach.

4.2 Piggybacking on Route Discoveries

As described in Section 3.2, when one host needs to send a packet to another, if the sender does not have a route cached to the destination host, it must initiate a separate route discovery, either buffering the original packet until the route reply is returned, or discarding it and relying on a higher-layer protocol to retransmit it if needed. The delay for route discovery and the total number of packets transmitted can be reduced by allowing data to be piggybacked on route request packets. Since some route requests may be propagated widely within the ad hoc network, though, the amount of data piggybacked must be limited. We currently use piggybacking when sending a route reply or a route error packet, since both are naturally small in size; small data packets such as the initial SYN packet opening a TCP connection [23] could also easily be piggybacked, but we have not yet experimented with this option.

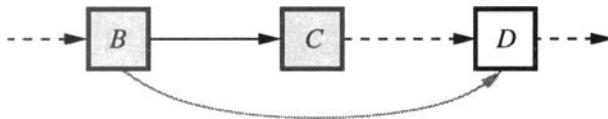


Figure 3 Mobile host *D* notices that the route can be shortened

One problem, however, arises when piggybacking on route request packets. If the route request is received by some host and is replied to based on the host's route cache without propagating the request (Section 4.1), the piggybacked data would be lost when the host discards the route request. In this case, before discarding the packet, the host must construct a new packet containing the piggybacked data from the route request packet, setting the route in this packet from the route being returned in the route reply. The route should appear as if the new packet had been sent by the initiator of the route discovery and had been forwarded normally to this host: the first portion of the route is taken from the accumulated route record in the route request packet, and the remainder of the route is taken from this host's route cache. The sender address in the packet should also be set to the initiator of the route discovery.

4.3 Reflecting Shorter Routes

While two hosts are communicating with each other using cached routes, it is desirable for the hosts to begin using shorter routes if the hosts move sufficiently closer together. In many cases, the basic route maintenance procedure is sufficient to accomplish this, since if one of the hosts moves enough to allow the route to be shortened, it will likely also move out of transmission range of the first hop on the existing route.

An improvement to this method of reflecting shorter routes is possible if hosts operate their network interfaces in promiscuous receive mode. Suppose somewhere in the forwarding of a packet, mobile host *B* transmits a packet to *C*, with *D* being the next hop after *C* in the route in the packet, as illustrated in Figure 3. If *D* receives this packet, it can examine the packet header to see that the packet reached it from *B* in one hop rather than two as intended by the route in the packet. In this case, *D* may infer that route may be shortened to exclude the intermediate hop through *C*. *D* then sends an unsolicited route reply packet to the original sender of the packet, informing it that it can now reach *D* in one hop from *B*. As with other route reply packets, other hosts which also receive this route reply (in particular, other hosts also operating in

(promiscuous receive mode) may also incorporate this change into their route caches. We believe that this method will ensure that the shortest routes will be used, although we have not yet added this method to our simulator.

4.4 Improved Handling of Errors

One common error condition that must be handled in an ad hoc network is the case in which the network effectively becomes partitioned. That is, two hosts that wish to communicate are not within transmission range of each other, and there are not enough other mobile hosts between them to form a sequence of hops through which they can forward packets. If a new route discovery were to be initiated for each packet sent by a host in this situation, a large number of unproductive route request packets would be propagated throughout the subset of the ad hoc network reachable from this host. In order to reduce the overhead from such route discoveries, we use exponential backoff to limit the rate at which new route discoveries may be initiated from any host for the same target. If the host attempts to send additional data packets to this same host more frequently than this limit, the subsequent packets may be buffered until a route reply is received, but they do not initiate a new route discovery until the minimum allowable interval between new route discoveries for this target has been reached. This limitation on the maximum rate of route discoveries for the same target is similar to the mechanism required by Internet hosts to limit the rate at which ARP requests are sent to any single IP address [3].

An additional optimization possible to improve the handling of errors is to use promiscuous receive mode to allow hosts to eavesdrop on route error packets being sent to other hosts. For example, Figure 4 shows the return of a route error packet to mobile host *A* from host *B*. If hosts *C*, *D*, and *E* are operating in promiscuous receive mode, they will be able to receive the route error packet. Since a route error packet names both ends of the route hop causing the error, any host receiving the error packet can update its route cache to reflect the fact that the two hosts indicated in the packet can no longer directly communicate. A host receiving a route error packet can simply search its route cache for any routes using this hop, and for each such route found, the route is truncated at this hop. All hosts on the route before this hop are still reachable on this route, but subsequent hosts are not.

It is possible, however, that a route error packet being returned to the original sender of a data packet may take a different route to the sender than the data packet took to the point at which the error was encountered. For example,

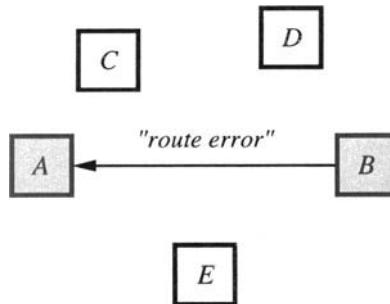


Figure 4 Mobile host *B* is returning a route error packet to *A*

in a network environment in which radio transmission between two hosts does not work equally well in both directions, a route discovery used by the host returning the route error packet may discover a different route back to the original sender. Thus, some hosts that may have cached the route in error may not be able to receive the route error packet, even using promiscuous receive mode. This situation can be improved by extending the handling of route error packets such that, once the route error packet reaches the original sender of the data packet, that original sender also retransmits the route error packet back to the point of error along the path originally used for the data packet, if this path differs from the one along which it received the route error packet.

A last optimization to improve the handling of errors is to support the caching of “negative” information in a host’s route cache. Suppose, in Figure 4, that none of these optimizations for handling errors are in use. When *A* receives *B*’s route error packet, it may initiate a route discovery in order to find a new route to the same target. However, if hosts *C*, *D*, or *E* have an entry in their route cache for this target, they will likely reply to *A* from their cache with a cached copy of the same route that *A* just removed from its cache. If instead, *A* could enter into its cache when it receives *B*’s route error packet, an indication that this hop is not currently working (rather than simply removing this hop from any routes currently in its cache), then *A* could ignore future replies from *C*, *D*, and *E* that include this hop from their caches. A short expiration period must be placed on this negative cached information, since while this entry is in its cache, *A* will otherwise refuse to allow this hop in any route entries in its cache, even if this hop begins working again.

We have not currently included this caching of negative information in our simulation, due to the difficulty of picking a suitable expiration period, and

since it appears to not be necessary in most cases, if hosts also promiscuously receive route error packets. For example, in Figure 4, if C , D , and E also receive B 's route error packet, they will have removed this hop from their caches before A 's new route discovery is initiated, thus avoiding the problem.

5 PERFORMANCE EVALUATION

5.1 The Simulation

To evaluate the performance of our dynamic source routing protocol, we constructed a packet-level simulator which allowed us to observe and measure the protocol's performance under a variety of conditions. In addition to a number of parameter choices in the protocol, the simulator allowed us to vary certain environmental factors such as the number of mobile hosts, the pattern and speed of host movement, and the distribution of the hosts in space. The simulator implements the basic protocol along with all optimizations described in Section 4 with the exceptions of reflecting shorter routes (Section 4.3) and the caching of negative information (Section 4.4).

The basic simulation parameters were chosen to model an ad hoc network consisting of a collection of mobile hosts moving around in a medium-sized room. The area in which the hosts move is square, 9 meters on a side. Each host moves with a velocity between 0.3 and 0.7 meters per second (somewhere between a slow walk and a quick stroll), and each transceiver has a range of 3 meters. These parameters could represent, for example, a group of hosts using diffuse infrared transceivers, or since the parameters are related and can be scaled linearly, they could also represent a number of cars with radio-equipped portables driving (very quickly) around an area of 1 square kilometer.

Each host is initially placed at a random position within in the simulation area. As the simulation progresses, each host pauses at its current location for a period, which we call the *pause time*, and then randomly chooses a new location to move to and a velocity between 0.3 and 0.7 meters per second at which to move there. Each host continues this behavior, alternately pausing and moving to a new location, for the duration of the simulation. Using this model, hosts appear to wander through the room with their restlessness determined by the pause time constant.

Whenever a host transmits a packet, some method must be used to determine which of the surrounding hosts will receive a copy of the packet. While our simulation's transmission model is admittedly simple, it still allows us to estimate the basic performance of the protocol. In the simulation, each host can be the originator of up to 3 conversations at a time, with the other participant in the conversation chosen randomly from among the other hosts. In actual use, we would expect hosts to communicate mostly with a small common subset of the available hosts (such as servers), which would reduce the number of route discoveries required. While a host is the initiator of less than three conversations, it initiates a new conversation with a randomly chosen partner after an average period of 15 seconds (using an exponential distribution). Each conversation lasts for a predetermined number of packets, the number being chosen from a geometric distribution with an average of 1000 packets. Again, in actual use, we would expect some (or all) of the conversations to be of longer duration than this, depending on the mix of network application programs in use. Short conversations, however, give a more conservative measure of the performance of the protocol, since more route discoveries are required as each host more frequently changes which other hosts it is communicating with.

During a conversation, packets are sent to the partner at exponentially distributed sending times with a uniformly chosen average rate between 2 and 5 packets per second. Packet lengths are chosen with a distribution such that 70% of the packets are long packets (1000 bytes) and the remainder are short packets (32 bytes). To give the appearance of a two way conversation, the partner sends a return packet back to the originator for every packet it receives. These return packets have the same length distribution as the originator's packets.

Packet transmission in the simulator is based on a network using link level acknowledgements at each hop. Transmissions to a host that is out of range always fail while transmissions to a host in range fail with a probability of 5%. The data link layer in the simulator will make up to 2 retransmission attempts if the first transmission fails. Each time a sender transmits a packet, the other hosts in range of the sender each have a 95% chance of overhearing the packet. The bandwidth for transmitting data is 100 Kbytes per second.

There are a number of parameters such as timeouts and holdoff periods which must be configured within the protocol. The values used in the simulator are shown in Table 1. Although we have attempted to choose reasonable values for these parameters and believe that our results are not particularly sensitive to these choices, we have not yet experimented with the effects of possible alternate choices.

Table 1 Parameter values used in the simulation

Parameter	Value
Period between nonpropagating route requests	5 sec.
Nonpropagating route request time out	100 msec.
Route request time out	500 msec.
Route request slot length	500 msec.
Maximum route request period	10 sec.
Route reply holdoff per-hop delay	4 msec.

The simulator does not attempt to model channel contention. Given a suitable data link layer protocol such as MACAW [2], the chance of data being lost to collision during periods of channel contention grows small, although at the expense of more delay in packet sending and receipt as each host waits for its turn to use the bandwidth. Therefore, while our model should accurately show the numbers of packets that will be received, we cannot completely evaluate the latency of any individual packet exchange.

The simulator also does not model possible one-way links between hosts. This choice is necessarily implied by our desire to implement the protocol on top of a data-link layer with link level acknowledgements. However, as described above, the protocol will still work in the absence of link layer acknowledgements.

A final minor limitation of our simulation is that our simulated environment is assumed to be devoid of obstacles to transmission or movement. Further, transmission failures are assumed to be uniformly distributed and independent, which does not take into account spatially localized failures due to sources such as microwave ovens, in the case of radio, or windows and reflections, in the case of infrared.

5.2 Results

We executed 20 runs of the simulator for each of a number of different movement rates and numbers of mobile hosts in the simulated ad hoc network. Each run simulated 4000 seconds of execution (just over one hour), with each mobile host moving and communicating as described in Section 5.1. The movement rate of the mobile hosts was determined by the pause time described above, with pause

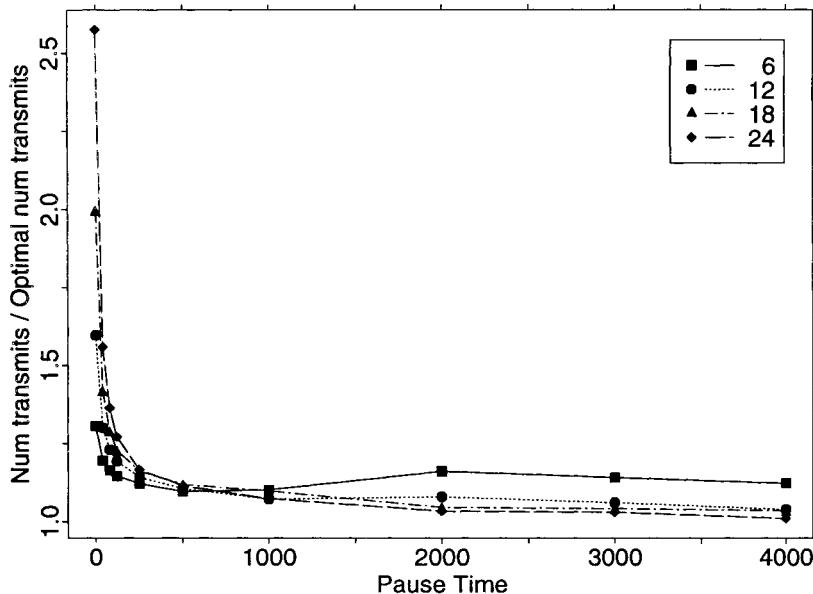


Figure 5 Average total number of transmissions performed relative to optimal (20 runs)

times ranging from 0 to 4000. With a pause time of 0, all hosts are constantly in motion, whereas with a pause time of 4000, hosts do not move during the run from their initial randomly chosen locations. The ad hoc network in each run consisted of either 6, 12, 18, or 24 mobile hosts. We present here the average over the 20 runs of the simulator for each of these cases; standard deviation for all cases was within 7% and in general was 2% or less of the average value for each case.

Figure 5 shows the total number of packet transmissions performed relative to the optimal number of transmissions for the data packets sent during the simulation. The optimal number of transmissions is the number of hops for each data packet needed to get from the sender of a packet to the intended receiver, if perfect routing decisions are made for each packet and if no transmission errors occur. The total number of packets actually transmitted includes the number of hops for each data packet based on the source route used by the sender, plus all packet transmissions used for route request, route reply, and route error packets. This ratio shows the work efficiency of the protocol: a

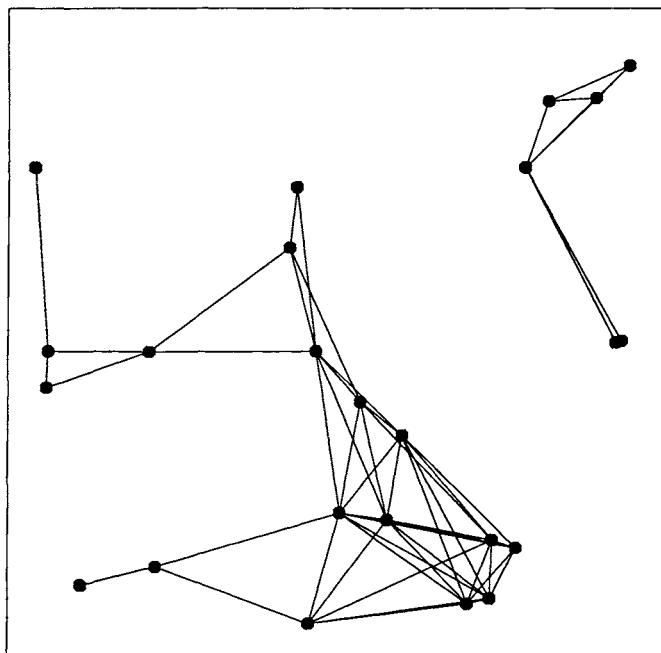


Figure 6 Example of disconnected clusters with 24 hosts

value of 1.0 indicates a perfectly efficient protocol with no overhead packets present.

For all but the shortest pause times in the simulated environment, the total number of packets transmitted by the protocol is very close to optimal, and falls to an overhead of about 1% (a ratio of 1.01) for pause times greater than 1000 with 24 mobile hosts, as shown in Figure 5. For very short pause times, representing very frequent host movement, the protocol overhead is higher, reaching a maximum ratio of 2.6 for a pause time of 0, representing all hosts in constant motion. In such situations, source routes become invalid quickly after they are discovered, and additional overhead is spent discovering new routes. However, because the route maintenance procedure can quickly detect when a route in use is no longer working, nearly all data packets can be successfully delivered even in periods of such extreme host movement. Distance vector protocols, on the other hand, would be unable to keep up with the rate of change in the network and would be unable to deliver most data packets.

The performance results for protocol overhead presented here are affected by the occurrence of disconnected components of the mobile hosts within the area of the ad hoc network. When placing a number of mobile hosts at random locations within the simulation area, there is a chance that some groups of hosts will be unable to communicate with other groups of hosts since all hosts in each group are out of wireless transmission range of all hosts in other groups. This effectively forms a partition of the ad hoc network. For example, Figure 6 illustrates a typical placement of 24 mobile hosts, which happened to form two disconnected components. With fewer mobile hosts spread over the same area, we have observed similar (but worse) occurrences of disconnected components.

For each data packet sent with the receiver outside the sender's disconnected component, the basic protocol would initiate a route discovery, although we have included an optimization to limit the rate of new discoveries using an exponential backoff, as described in Section 4.4. The remaining extra route discoveries still performed in such situations show in increased protocol overhead, such as in the higher overhead values for 6 and 12 hosts shown in Figure 5, although the number of such extra route discoveries due disconnected components is greatly reduced by this optimization.

Figure 7 shows the average length of a source route used in sending a data packet relative to the optimal route length for the packets sent during the simulation. The optimal route length here is the number of hops needed to reach the destination if perfect routing information were available. In Figure 7, a different scale has been used on the y-axis than was used in Figure 5 in order to show the relevant detail in the shape of each graph. The ratio of the length of routes used relative to optimal shows the degree to which the protocol finds and maintains optimal routes as the mobile hosts move about. As shown here, the protocol finds and uses close to optimal routes, even without the route shortening optimization using promiscuous receive mode described in Section 4.3. The difference in length between the routes used and the optimal route lengths is negligible. In most cases, the route lengths used are on average within a factor of 1.01 of optimal, and in all cases are within a factor of 1.09 of optimal.

6 RELATED WORK

Research on packet routing in wireless networks of mobile hosts dates back at least to 1973 when the U.S. Defense Advanced Research Projects Agency began

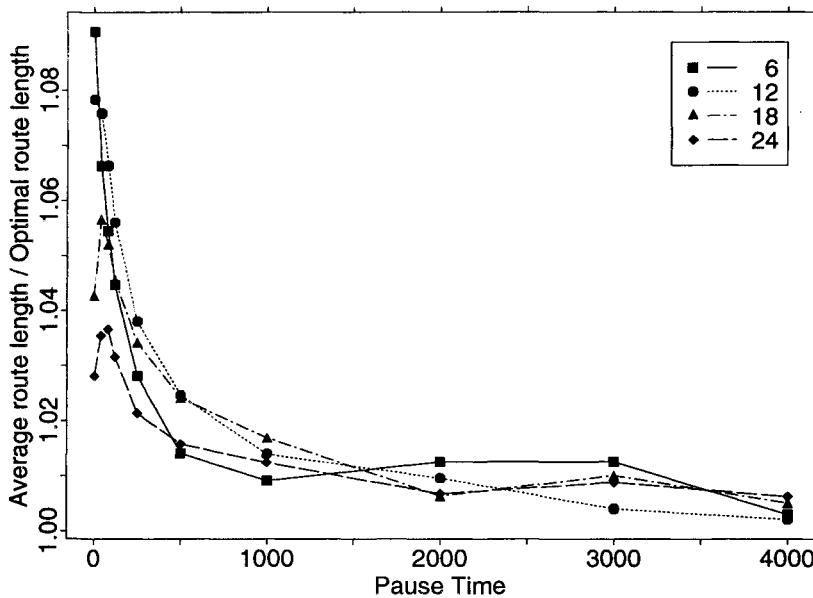


Figure 7 Average route length used relative to optimal (20 runs)

the DARPA Packet Radio Network (PRNET) project [11] and its successor the Survivable Adaptive Networks (SURAN) project [15]. PRNET supports the automatic set up and maintenance of packet switched communication routes in a network of moderately mobile hosts communicating via radios. The PRNET routing protocol uses a form of distance vector routing, with each node broadcasting a routing update packet (called a packet-radio organization packet, or PROP, in PRNET) every 7.5 seconds. The header of each data packet contains the source and destination node addresses, the number of hops taken so far from the source, and the number of hops remaining to reach the destination (based on the sender's routing table). Nodes promiscuously receive all packets and may update their routing tables based on this header information. The data link protocol uses hop-by-hop acknowledgements, using either explicit (active) acknowledgements or passive acknowledgements from these promiscuously received packets.

The amateur radio community has also worked extensively with routing in wireless networks of (sometimes) mobile hosts [14], holding an annual packet radio computer networking conference sponsored by the American Radio Relay League (ARRL) since 1981. Amateur packet radio networking originally

used only source routing with explicit source routes constructed by the user, although some had considered the possibility of a more dynamic source routing scheme [7]. A system known as NET/ROM was also developed to allow the routing decisions to be automated, using a form of distance vector routing protocol rather than source routing [6, 8]. NET/ROM also allows updating of its routing table based on the source address information in the headers of packets that it receives.

A particular problem with the use of distance vector routing protocols in networks with hosts that move, is the possibility of forming routing loops. In order to eliminate this possibility, Perkins and Bhagwat have recently proposed adding sequence numbers to routing updates in their Destination-Sequenced Distance Vector (DSDV) protocol [19]. These sequence numbers are used to compare the age of information in a routing update, and allow each node to preferentially select routes based on the freshest information. DSDV also uses triggered routing updates to speed route convergence. In order to damp route fluctuation and reduce congestion from large numbers of triggered updates after a route changes, each node in DSDV maintains information about the frequency with which it sees route changes and may delay some routing updates.

Our dynamic source routing protocol is similar in approach to some source routing protocols used in wired networks, such as in the IEEE 802 SRT bridge standard [20], in FLIP [12], and in SDRP [5, 28]. Our route request packet serves essentially the same role in route discovery as an “all paths explorer” packet. However, in wired networks, a bridge can copy an all paths explorer from one network interface onto each of its other interfaces and be sure that the explorer will flood the network in an orderly and complete way. Our protocol includes optimizations such as caching \langle initiator address, request id \rangle pairs to efficiently flood explorers through a wireless network. We also make extensive use of caching and can effectively make use of promiscuous receive mode in the network interface to optimize route discovery.

The Internet Address Resolution Protocol (ARP) [21] is used to find the MAC address of a host on the same LAN as the sender. ARP is somewhat similar to our nonpropagating route request packets, except that a mobile host may answer the route request from its cache whereas ARP requests are normally only answered by the target host itself. In cases in which several LANs have been bridged together, the bridge may run “proxy” ARP [24], which allows the bridge to answer an ARP request on behalf of another host. In this sense, our nonpropagating route requests are also similar to proxy ARP in that they expand the effective size of a single host’s route cache by allowing it to cheaply

make use of the caches of neighboring hosts to reduce the need for propagating request packets.

One tradeoff between source routing and distance vector routing is in the handling of partitioned networks, as mentioned in Section 5.2. Under dynamic source routing, if a host wishes to communicate with an unreachable host, the rate at which route requests are made will be reduced by a back off mechanism; the protocol, however, continues to make periodic efforts to find a route to the unreachable host, consuming some network resources. Under distance vector routing, with the assumption that routes have had time to converge once the host became unreachable, no network resources are spent trying to send packets to an unreachable host, as none of the hosts in the sender's partition of the network have a routing table entry for the destination. In practice, the problem of attempting to route packets to unreachable hosts is less significant, as connection-oriented protocols typically give up after a certain timeout or number of attempts, and human users will likewise give up eventually on any attempt to reach an unreachable host. In our simulation, however, a host continues to attempt sending packets to the destination host for the entire duration of each simulated conversation.

In general, when hosts move quickly enough and frequently enough, the best strategy that any routing protocol can use is to flood data packets throughout the network in hopes that the moving host will run into one of the many copies. On the other hand, when host movement is very slow or infrequent, ideally no overhead should be required for routing, since none of the routing information changes. Dynamic source routing moves easily between these two regimes, driven by the rate at which route requests are initiated. If a host with which another host wants to communicate is moving very quickly, source routing floods the network with a route request and then sends a data packet as soon as the destination is found and a route reply returned. Hosts uninterested in talking to this quickly moving host may see some of the available bandwidth consumed by the route requests but will have their own routes left unaffected. Likewise, if host motion slows, the rate at which routes cease working will slow and routing overhead will thus be reduced due to less frequent need for new route discoveries.

7 CONCLUSION

This paper has presented a protocol for routing packets between wireless mobile hosts in an ad hoc network. Unlike routing protocols using distance vector or link state algorithms, our protocol uses dynamic source routing which adapts quickly to routing changes when host movement is frequent, yet requires little or no overhead during periods in which hosts move less frequently. Based on results from a packet-level simulation of mobile hosts operating in an ad hoc network, the protocol performs well over a variety of environmental conditions such as host density and movement rates. For all but the highest rates of host movement simulated, the overhead of the protocol is quite low, falling to just 1% of total data packets transmitted for moderate movement rates in a network of 24 mobile hosts. In all cases, the difference in length between the routes used and the optimal route lengths is negligible, and in most cases, route lengths are on average within a factor of 1.02 of optimal.

We are currently expanding our simulations to incorporate some additional optimizations and to quantify the effects of the individual optimizations on the behavior and performance of the protocol. We are also continuing to study other routing protocols for use in ad hoc networks, including those based on distance vector or link state routing, as well as the interconnection of an ad hoc network with a wide-area network such as the Internet, reachable by some but not all of the ad hoc network nodes. Although this paper does not address the security concerns inherent in wireless networks or packet routing, we are currently examining these issues with respect to attacks on privacy and denial of service in the routing protocol. Finally, we are beginning implementation of the protocol on notebook computers for use by students in an academic environment.

Acknowledgements

This research was supported in part by the Wireless Initiative of the Information Networking Institute at Carnegie Mellon University, and by the National Science Foundation under CAREER Award NCR-9502725. David Maltz was also supported in part by an IBM Cooperative Fellowship.

REFERENCES

- [1] David F. Bantz and Frédéric J. Bauchot. Wireless LAN design alternatives. *IEEE Network*, 8(2):43–53, March/April 1994.
- [2] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. MACAW: A media access protocol for wireless LAN's. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 212–225, August 1994.
- [3] Robert T. Braden, editor. Requirements for Internet hosts. Internet Request For Comments RFC 1122, October 1989.
- [4] Roy C. Dixon and Daniel A. Pitt. Addressing, bridging, and source routing. *IEEE Network*, 2(1):25–32, January 1988.
- [5] Deborah Estrin, Daniel Zappala, Tony Li, Yakov Rekhter, and Kannan Varadhan. Source Demand Routing: Packet format and forwarding specification (version 1). Internet Draft, January 1995. Work in progress.
- [6] Daniel M. Frank. Transmission of IP datagrams over NET/ROM networks. In *ARRL Amateur Radio 7th Computer Networking Conference*, pages 65–70, October 1988.
- [7] Bdale Garbee. Thoughts on the issues of address resolution and routing in amateur packet radio TCP/IP networks. In *ARRL Amateur Radio 6th Computer Networking Conference*, pages 56–58, August 1987.
- [8] James Geier, Martin DeSimio, and Byron Welsh. Network routing techniques and their relevance to packet radio networks. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 105–117, September 1990.
- [9] C. Hedrick. Routing Information Protocol. Internet Request For Comments RFC 1058, June 1988.
- [10] International Standards Organization. Intermediate system to intermediate system intra-domain routing exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473). ISO DP 10589, February 1990.
- [11] John Jubin and Janet D. Tornow. The DARPA packet radio network protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

- [12] M. Frans Kaashoek, Robbert van Renesse, Hans van Staveren, and Andrew S. Tanenbaum. FLIP: An internetwork protocol for supporting distributed systems. *ACM Transactions on Computer Systems*, 11(1):73–106, February 1993.
- [13] Phil Karn. MACA—A new channel access method for packet radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 134–140, September 1990.
- [14] Philip R. Karn, Harold E. Price, and Robert J. Diersing. Packet radio in the amateur service. *IEEE Journal on Selected Areas in Communications*, SAC-3(3):431–439, May 1985.
- [15] Gregory S. Lauer. Packet-radio routing. In *Routing in Communications Networks*, edited by Martha E. Steenstrup, chapter 11, pages 55–76. Prentice-Hall, Englewood Cliffs, New Jersey, 1995.
- [16] John M. McQuillan, Ira Richer, and Eric C. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications*, COM-28(5):711–719, May 1980.
- [17] John M. McQuillan and David C. Walden. The ARPA network design decisions. *Computer Networks*, 1(5):243–289, August 1977.
- [18] J. Moy. OSPF version 2. Internet Request For Comments RFC 1247, July 1991.
- [19] Charles E. Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, August 1994.
- [20] Radia Perlman. *Interconnections: Bridges and Routers*. Addison-Wesley, Reading, Massachusetts, 1992.
- [21] David C. Plummer. An Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit Ethernet addresses for transmission on Ethernet hardware. Internet Request For Comments RFC 826, November 1982.
- [22] J. B. Postel, editor. Internet Protocol. Internet Request For Comments RFC 791, September 1981.
- [23] J. B. Postel, editor. Transmission Control Protocol. Internet Request For Comments RFC 793, September 1981.

- [24] J. B. Postel. Multi-LAN address resolution. Internet Request For Comments RFC 925, October 1984.
- [25] Nachum Shacham and Jil Westcott. Future directions in packet radio architectures and protocols. *Proceedings of the IEEE*, 75(1):83–99, January 1987.
- [26] Gursharan S. Sidhu, Richard F. Andrews, and Alan B. Oppenheimer. *Inside AppleTalk*. Addison Wesley, Reading, Massachusetts, 1990.
- [27] Paul Turner. NetWare communications processes. *NetWare Application Notes*, Novell Research, pages 25–81, September 1990.
- [28] Kannan Varadhan, Deborah Estrin, Steve Hotz, and Yakov Rekhter. SDRP route construction. Internet Draft, July 1994. Work in progress.
- [29] Xerox Corporation. Internet transport protocols. Xerox System Integration Standard 028112, December 1981.

ROUTING OVER MULTI-HOP WIRELESS NETWORK OF MOBILE COMPUTERS

Charles E. Perkins and Pravin Bhagwat[†]

IBM Research - Yorktown Heights, NY

[†]*University of Maryland - College Park, MD*

ABSTRACT

An *ad-hoc* network is the cooperative engagement of a collection of Mobile Hosts without the required intervention of any centralized Access Point. In this paper we present an innovative design for the operation of such ad-hoc networks. The basic idea of the design is to operate each Mobile Host as a specialized router, which periodically advertises its view of the interconnection topology with other Mobile Hosts within the network. This amounts to a new sort of routing protocol. We have investigated modifications to the basic Bellman-Ford routing mechanisms, as specified by the Routing Information Protocol, making it suitable for a dynamic and self-starting network mechanism as is required by users wishing to utilize ad-hoc networks. Our modifications address some of the previous objections to the use of Bellman-Ford, related to the poor looping properties of such algorithms in the face of broken links and the resulting time dependent nature of the interconnection topology describing the links between the Mobile Hosts. Finally, we describe the ways in which the basic network-layer routing can be modified to provide MAC-layer support for ad-hoc networks.

Previously appeared in Proceedings of ACM SIGCOMM'94 under the title "Highly Dynamic Destination-Sequenced Distance Vector Routing (DSDV) for Mobile Computers." Copyright ©1994 by the Association for Computing Machinery, Inc. Reprinted by permission. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept, ACM Inc., fax +1 (212) 869-0481, or (permissions@acm.org).

1 INTRODUCTION

Recently, there has been tremendous growth in the sales of laptop and portable computers. These smaller computers, nevertheless, can be equipped with hundreds of megabytes of disk storage, high resolution color displays, pointing devices, and wireless communications adapters. Moreover, since many of these small (in size only) computers operate for hours with battery power, users are free to move about at their convenience without being constrained by wires.

This is a revolutionary development in personal computing. Battery powered, untethered computers are likely to become a pervasive part of our computing infrastructure. As people begin to have mobile computers handy, for whatever purposes, sharing information between the computers will become a natural requirement. Currently, such sharing is made difficult by the need for users to perform administrative tasks and set up static, bidirectional links between their computers. However, if the wireless communications systems in the mobile computers support a broadcast mechanism, much more flexible and useful ways of sharing information can be imagined. For instance, any number of people could conceivably enter a conference room and agree to support communications links between themselves, without necessarily engaging the services of any pre-existing equipment in the room (i.e., without requiring any pre-existing communications infrastructure). Thus, one of our primary motivations is to allow the construction of temporary networks with no wires and no administrative intervention required. In this paper, such a interconnection between the mobile computers will be called an *ad-hoc* network, in conformance with current usage within the IEEE 802.11 subcommittee [1].

Ad-hoc networks differ significantly from existing networks. First of all, the topology of interconnections may be quite dynamic. Secondly, most users will not wish to perform any administrative actions to set up such a network. In order to provide service in the most general situation, we do not assume that every computer is within communication range of every other computer. This lack of complete connectivity would certainly be a reasonable characteristic of, say, a population of mobile computers in a large room which relied on infrared transceivers to effect their data communications.

From a graph theoretic point of view, an ad-hoc network is a graph, $G(N, E(t))$, which is formed by denoting each mobile host by a node and drawing an edge between two nodes if they are in direct communication range of each other. The set of edges, $E(t)$, so formed, is a function of time, and it keeps changing as nodes in the ad-hoc network move around. The topology defined by such a

network can be very arbitrary since there are no constraints on where mobiles could be located with respect to each other. Routing protocols for existing networks [2, 3, 4] have not been designed specifically to provide the kind of self-starting behavior needed for ad-hoc networks. Most protocols exhibit their least desirable behavior when presented with a highly dynamic interconnection topology. Although we thought that mobile computers could naturally be modeled as *routers*, it was also clear that existing routing protocols would place too heavy a computational burden on each mobile computer. Moreover, the convergence characteristics of existing routing protocols did not seem good enough to fit the needs of ad-hoc networks. Lastly, the wireless medium differs in important ways from wired media, which would require that we make modifications to whichever routing protocol we might choose to experiment with. For instance, mobile computers may well have only a single network interface adapter, whereas most existing routers have network interfaces to connect two separate networks together. Since we had to make lots of changes anyway, we decided to follow our ad-hoc network model as far as we could and ended up with a substantially new approach to the classic distance-vector routing.

2 OVERVIEW OF ROUTING METHODS

In our environment, the problem of routing is essentially the distributed version of the shortest path problem [2]. Each node in the network maintains for each destination a preferred neighbor. Each data packet contains a destination node identifier in its header. When a node receives a data packet, it forwards the packet to the preferred neighbor for its destination. The forwarding process continues until the packet reaches its destination. The manner in which routing tables are constructed, maintained and updated differs from one routing method to another. Popular routing methods, however, attempt to achieve the common objective of routing packets along the optimal path. The next-hop routing methods can be categorized into two primary classes: *link-state* and *distance-vector*.

2.1 Link-State

The link-state approach is closer to the centralized version of the shortest path computation method. Each node maintains a view of the network topology with a cost for each link. To keep these views consistent, each node periodically broadcasts the link costs of its outgoing links to all other nodes using a protocol

such as flooding. As a node receives this information, it updates its view of the network topology and applies a shortest-path algorithm to choose its next hop for each destination. Some of the link costs in a node's view can be incorrect because of long propagation delays, partitioned network, etc. Such inconsistent views of network topologies might lead to formation of routing loops. These loops, however, are short-lived, because they disappear in the time it takes a message to traverse the diameter of the network [3].

2.2 Distance-Vector

In distance-vector algorithms, every node i maintains, for each destination x , a set of distances $\{d_{ij}^x\}$ where j ranges over the neighbors of i . Node i treats neighbor k as a next-hop for a packet destined for x if d_{ik}^x equals $\min_j \{d_{ij}^x\}$. The succession of next hops chosen in this manner lead to x along the shortest path. In order to keep the distance estimates up-to-date, each node monitors the cost of its outgoing links and periodically broadcasts, to each one its neighbors, its current estimate of the shortest distance to every other node in the network.

The above distance-vector algorithm is the classical Distributed Bellman-Ford (DBF) algorithm [5]. Compared to link-state method, it is computationally more efficient, easier to implement and requires much less storage space. However, it is well known that this algorithm can cause the formation of both short-lived and long-lived loops [6]. The primary cause for formation of routing loops is that nodes choose their next-hops in a completely distributed fashion based on information which can possibly be stale and, therefore, incorrect. Almost all proposed modifications to DBF algorithm [7, 8, 9] eliminate the looping problem by forcing all nodes in the network to participate in some form of internodal coordination protocol. Such internodal coordination mechanisms might be effective when topological changes are rare. However, within an ad-hoc mobile environment, enforcing any such internodal coordination mechanism will be difficult due to the rapidly changing topology of the underlying routing network.

Simplicity is one of the primary attributes which makes any routing protocol *preferred* over others for implementation within operational networks. RIP [4] is a classical example. Despite the *counting-to-infinity* problem it has proven to be very successful within small size internetworks. The usefulness of RIP within ad-hoc environment, however, is limited as it was not designed to handle rapid topological changes. Furthermore, the techniques of *split-horizon* and *poisoned-reverse* [4] are not useful within the wireless environment due to the broadcast

nature of the transmission medium. Our design goal therefore has been to design a routing method for ad-hoc networks which preserves the simplicity of RIP, yet at the same time avoids the looping problem. Our approach is to tag each route table entry with a sequence number so that nodes can quickly distinguish stale routes from the new ones and thus avoid formation of routing loops.

3 DESTINATION-SEQUENCED DISTANCE VECTOR (DSDV) PROTOCOL

Our proposed routing method allows a collection of mobile computers, which may not be close to any base station and can exchange data along changing and arbitrary paths of interconnection, to afford all computers among their number a (possibly multi-hop) path along which data can be exchanged. In addition, our solution must remain compatible with operation in cases where a base station is available. By the methods outlined in this paper, not only will routing be seen to solve the problems associated with ad-hoc networks, but in addition we will describe ways to perform such routing functions at Layer 2, which traditionally has not been utilized as a protocol level for routing.

Protocol Overview

Packets are transmitted between the stations of the network by using routing tables which are stored at each station of the network. Each routing table, at each of the stations, lists all available destinations, and the number of hops to each. Each route table entry is tagged with a sequence number which is originated by the destination station. To maintain the consistency of routing tables in a dynamically varying topology, each station periodically transmits updates, and transmits updates immediately when significant new information is available. Since we do not assume that the mobile hosts are maintaining any sort of time synchronization, we also make no assumption about the phase relationship of the update periods between the mobile hosts. These packets indicate which stations are accessible from each station and the number of hops necessary to reach these accessible stations, as is often done in distance-vector routing algorithms. It is not the purpose of this paper to propose any new metrics for route selection other than the freshness of the sequence numbers

associated with the route; cost or other metrics might easily replace the number of hops in other implementations. The packets may be transmitted containing either layer 2 (MAC) addresses or layer 3 (network) addresses.

Routing information is advertised by broadcasting or multicasting the packets which are transmitted periodically and incrementally as topological changes are detected – for instance, when stations move within the network. Data is also kept about the length of time between arrival of the *first* and the arrival of the *best* route for each particular destination. Based on this data, a decision may be made to delay advertising routes which are about to change soon, thus damping fluctuations of the route tables. The advertisement of possibly unstable routes is delayed in order to reduce the number of rebroadcasts of possible route entries that normally arrive with the same sequence number.

Route Advertisements

The DSDV protocol requires each mobile station to advertise, to each of its current neighbors, its own routing table (for instance, by broadcasting its entries). The entries in this list may change fairly dynamically over time, so the advertisement must be made often enough to ensure that every mobile computer can almost always locate every other mobile computer of the collection. In addition, each mobile computer agrees to relay data packets to other computers upon request. This agreement places a premium on the ability to determine the shortest number of hops for a route to a destination; we would like to avoid unnecessarily disturbing mobile hosts if they are in sleep mode. In this way a mobile computer may exchange data with any other mobile computer in the group even if the target of the data is not within range for direct communication. If the notification of which other mobile computers are accessible from any particular computer in the collection is done at layer 2, then DSDV will work with whatever higher layer (e.g., Network Layer) protocol might be in use.

All the computers interoperating to create data paths between themselves broadcast the necessary data periodically, say once every few seconds. In a wireless medium, it is important to keep in mind that broadcasts are limited in range by the physical characteristics of the medium. This is different than the situation with wired media, which usually have a much more clearly defined range of reception.

Routing Table Entry Structure

The data broadcast by each mobile computer will contain its new sequence number and the following information for each new route:

- The destination's address;
- The number of hops required to reach the destination; and
- The sequence number of the information received regarding that destination, as originally stamped by the destination;

The transmitted routing tables will also contain the hardware address, and (if appropriate) the network address, of the mobile computer transmitting them, within the headers of the packet. The routing table will also include a sequence number created by the transmitter. Routes with more recent sequence numbers are always preferred as the basis for making forwarding decisions, but not necessarily advertised. Of the paths with the same sequence number, those with the smallest metric will be used. By the natural way in which the routing tables are propagated, the sequence number is sent to all mobile computers which may each decide to maintain a routing entry for that originating mobile computer.

Routes received in broadcasts are also advertised by the receiver when it subsequently broadcasts its routing information; the receiver adds an increment to the metric before advertising the route, since incoming packets will require one more hop to reach the destination (namely, the hop from the transmitter to the receiver). Again, we do not explicitly consider here the changes required to use metrics which do not use the hop count to the destination.

Wireless media differ from traditional wired networks because asymmetries produced by one-way "links" are more prevalent. Receiving a packet from a neighbor, therefore, does not indicate the existence of a single-hop data path back to that neighbor across the wireless medium. In order to avoid problems caused by such one-way links, each mobile node may not insert routing information received from a neighbor unless that neighbor shows that it can receive packets from the mobile node. Thus, effectively, our routing algorithms only uses links which are bidirectional.

One of the most important parameters to be chosen is the time between broadcasting the routing information packets. However, when any new or substan-

tially modified route information is received by a Mobile Host, the new information will be retransmitted soon (subject to constraints imposed for damping route fluctuations), effecting the most rapid possible dissemination of routing information among all the cooperating Mobile Hosts. This quick re-broadcast introduces a new requirement for our protocols to converge as soon as possible. It would be calamitous if the movement of a Mobile Host caused a storm of broadcasts, degrading the availability of the wireless medium.

Responding to Topology Changes

Mobile Hosts cause broken links as they move from place to place. The broken link may be detected by the layer-2 protocol, or it may instead be inferred if no broadcasts have been received for a while from a former neighbor. A broken link is described by a metric of ∞ (i.e., any value greater than the maximum allowed metric). When a link to a next hop has broken, any route through that next hop is immediately assigned an ∞ metric and assigned an updated sequence number. Since this qualifies as a substantial route change, such modified routes are immediately disclosed in a broadcast routing information packet. Building information to describe broken links is the only situation when the sequence number is generated by any Mobile Host other than the destination Mobile Host. Sequence numbers generated to indicate ∞ will be one greater than the last finite sequence number received from the destination. When a node receives an ∞ metric, and it has an equal or later sequence number with a finite metric, it triggers a route update broadcast to disseminate the important news about that destination. In this way routes containing any finite sequence numbers will supersede routes generated with the ∞ metric.

In a very large population of Mobile Hosts, adjustments will likely be made in the time between broadcasts of the routing information packets. In order to reduce the amount of information carried in these packets, two types will be defined. One will carry all the available routing information, called a "full dump". The other type will carry only information changed since the last full dump, called an "incremental". By design, an incremental routing update should fit in one network protocol data unit (NPDU). The full dump will most likely require multiple NPDU's, even for relatively small populations of Mobile Hosts. Full dumps can be transmitted relatively infrequently when no movement of Mobile Hosts is occurring. When movement becomes frequent, and the size of an incremental approaches the size of a NPDU, then a full dump can be scheduled (so that the next incremental will be smaller). It is expected that mobile nodes will implement some means for determining which route changes

are significant enough to be sent out with each incremental advertisement. For instance, when a stabilized route shows a different metric for some destination, that would likely constitute a significant change that needed to be advertised after stabilization. If a new sequence number for a route is received, but the metric stays the same, that would be unlikely to be considered as a significant change.

Route Selection Criteria

When a Mobile Host receives new routing information (usually in an incremental packet as just described), that information is compared to the information already available from previous routing information packets. Any route with a more recent sequence number is used. Routes with older sequence numbers are discarded. A route with a sequence number equal to an existing route is chosen if it has a "better" metric, and the existing route discarded, or stored as less preferable. The metrics for routes chosen from the newly received broadcast information are each incremented by one hop. Newly recorded routes are scheduled for immediate advertisement to the current Mobile Host's neighbors. Routes which show a more recent sequence number may be scheduled for advertisement at a time which depends on the average settling time for routes to the particular destination under consideration.

Timing skews between the various Mobile Hosts are expected. The broadcasts of routing information by the Mobile Hosts are to be regarded as somewhat asynchronous events, even though some regularity is expected. In such a population of independently transmitting agents, some fluctuation could develop using the above procedures for updating routes. It could turn out that a particular Mobile Host would receive new routing information in a pattern which causes it to consistently change routes from one next hop to another, even when the destination Mobile Host has not moved. This happens because there are two ways for new routes to be chosen; they might have a later sequence number, or they might have a better metric. A Mobile Host could conceivably always receive two routes to the same destination, with a newer sequence number, one after another (via different neighbors), but always get the route with the worse metric first. Unless care is taken, this will lead to a continuing burst of new route transmittals upon every new sequence number from that destination. Each new metric is propagated to every Mobile Host in the neighborhood, which propagates to their neighbors and so on.

One solution is to delay the advertisement of such routes, when a Mobile Host can determine that a route with a better metric is likely to show up soon. The route with the later sequence number must be available for use, but it does not have to be advertised immediately unless it is a route to a destination which was previously unreachable. Thus, there will be two routing tables kept at each Mobile Host: one for use with forwarding packets, and another to be advertised via incremental routing information packets. To determine the probability of imminent arrival of routing information showing a better metric, the Mobile Host has to keep a history of the weighted average time that routes to a particular destination fluctuate until the route with the best metric is received. Received route updates with infinite metrics are not included in this computation of the settling time for route updates. We hope that such a procedure will allow us to predict how long to wait before advertising new routes.

Operating DSDV at Layer 2

The addresses stored in the routing tables will correspond to the layer at which this ad-hoc networking protocol is operated. That is, operation at Layer 3 will use network layer addresses for the next hop and destination addresses, and operation at Layer 2 will use Layer 2 Media Access Control (MAC) addresses.

Using MAC addresses for the forwarding table does introduce a new requirement, however. The difficulty is that Layer 3 network protocols provide communication based on network addresses, and a way must be provided to resolve these Layer 3 addresses into MAC addresses. Otherwise, a multiplicity of different address resolution mechanisms would be put into place, and a corresponding loss of bandwidth in the wireless medium would be observed whenever the resolution mechanisms were utilized. This could be substantial since such mechanisms would require broadcasts and retransmitted broadcasts by every Mobile Host in the ad-hoc network. Thus, unless special care is taken, every address resolution might look like a glitch in the normal operation of the network, which may well be noticeable to any active users.

The solution proposed here, for operation at Layer 2, is to include Layer 3 protocol information along with the Layer 2 information. Each destination host would advertise which Layer 3 protocols it supports, and each Mobile Host advertising reachability to that destination would include along, with the advertisement, the information about the Layer 3 protocols supported at that destination. This information would only have to be transmitted when

it changes, which occurs rarely. Changes would be transmitted as part of each incremental dump. Since each Mobile Host could support several Layer 3 protocols (and many will), this list would have to be variable in length.

Extending Base Station Coverage

Mobile computers will frequently be used in conjunction with base stations, which allow them to exchange data with other computers connected to the wired network. By participating in the DSDV protocol, base stations can extend their coverage beyond the range imposed by their wireless transmitters. When a base station participates in DSDV, it is shown as a default route in the tables transmitted by a mobile station. In this way, mobile stations within range of a base station can cooperate to effectively extend the range of the base station to serve other stations outside the range of the base station, as long as those other mobile stations are close to some other mobile station that is within range.

4 EXAMPLES OF DSDV IN OPERATION

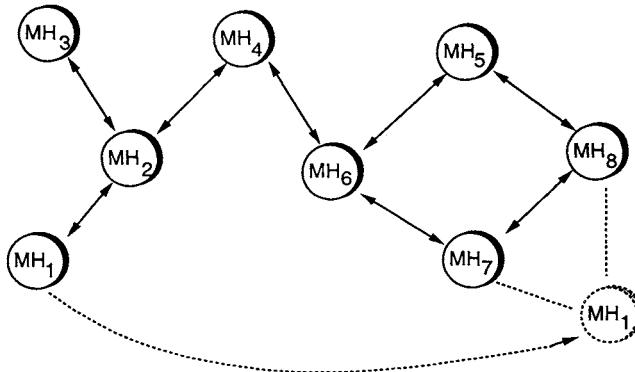


Figure 1 Movement in an ad-hoc network

Consider MH_4 in Figure 1. Table 1 shows a possible structure of the forwarding table which is maintained at MH_4 . Suppose the address¹ of each Mobile Host is represented as MH_i . Suppose further that all sequence numbers are denoted

¹If DSDV is operated at level 2 then MH_i denotes the MAC address, otherwise it denotes a level 3 address

Destination	NextHop	Metric	Sequence number	Install	Stable_data
MH_1	MH_2	2	S406_ MH_1	T001_ MH_4	Ptr1_ MH_1
MH_2	MH_2	1	S128_ MH_2	T001_ MH_4	Ptr1_ MH_2
MH_3	MH_2	2	S564_ MH_3	T001_ MH_4	Ptr1_ MH_3
MH_4	MH_4	0	S710_ MH_4	T001_ MH_4	Ptr1_ MH_4
MH_5	MH_6	2	S392_ MH_5	T002_ MH_4	Ptr1_ MH_5
MH_6	MH_6	1	S076_ MH_6	T001_ MH_4	Ptr1_ MH_6
MH_7	MH_6	2	S128_ MH_7	T002_ MH_4	Ptr1_ MH_7
MH_8	MH_6	3	S050_ MH_8	T002_ MH_4	Ptr1_ MH_8

Table 1 Structure of the MH_4 forwarding table

$SNNN_MH_i$, where MH_i specifies the computer that created the sequence number and $SNNN$ is a sequence number value. Also suppose that there are entries for all other Mobile Hosts, with sequence numbers $SNNN_MH_i$, before MH_1 moves away from MH_2 . The install time field helps determine when to delete stale routes. With our protocol, the deletion of stale routes should rarely occur, since the detection of link breakages should propagate through the ad-hoc network immediately. Nevertheless, we expect to continue to monitor for the existence of stale routes and take appropriate action.

From table 1, one could surmise, for instance, that all the computers became available to MH_4 at about the same time, since its install_time for most of them is about the same. $Ptr1_MH_i$ would all be pointers to null structures, because there are not any routes in Figure 1 which are likely to be superseded or compete with other possible routes to any particular destination.

Table 2 shows the structure of the advertised route table of MH_4 .

Now suppose that MH_1 moves into the general vicinity of MH_8 and MH_7 , and away from the others (especially MH_2). The new internal forwarding tables at MH_4 might then appear as shown in table 3.

Only the entry for MH_1 shows a new metric, but in the intervening time, many new sequence number entries have been received. The first entry thus must be advertised in subsequent incremental routing information updates until the next full dump occurs. When MH_1 moved into the vicinity of MH_8 and MH_7 , it triggered an immediate incremental routing information update which was then broadcast to MH_6 . MH_6 , having, determined that significant new routing information had been received, also triggered an immediate update

Destination	Metric	Sequence number
MH_1	2	S406_ MH_1
MH_2	1	S128_ MH_2
MH_3	2	S564_ MH_3
MH_4	0	S710_ MH_4
MH_5	2	S392_ MH_5
MH_6	1	S076_ MH_6
MH_7	2	S128_ MH_7
MH_8	3	S050_ MH_8

Table 2 Advertised route table by MH_4

Destination	NextHop	Metric	Sequence number	Install	Stable_data
MH₁	MH₆	3	S516_ MH_1	T810_ MH_4	Ptr1_ MH_1
MH_2	MH_2	1	S238_ MH_2	T001_ MH_4	Ptr1_ MH_2
MH_3	MH_2	2	S674_ MH_3	T001_ MH_4	Ptr1_ MH_3
MH_4	MH_4	0	S820_ MH_4	T001_ MH_4	Ptr1_ MH_4
MH_5	MH_6	2	S502_ MH_5	T002_ MH_4	Ptr1_ MH_5
MH_6	MH_6	1	S186_ MH_6	T001_ MH_4	Ptr1_ MH_6
MH_7	MH_6	2	S238_ MH_7	T002_ MH_4	Ptr1_ MH_7
MH_8	MH_6	3	S160_ MH_8	T002_ MH_4	Ptr1_ MH_8

Table 3 MH_4 forwarding table (updated)

which carried along the new routing information for MH_1 . MH_4 , upon receiving this information, would then broadcast it at every interval until the next full routing information dump. At MH_4 , the incremental advertised routing update would have the form as shown in table 4.

In this advertisement, the information for MH_4 comes first, since it is doing the advertisement. The information for MH_1 comes next, not because it has a lower address, but because MH_1 is the only one which has any significant route changes affecting it. As a general rule, routes with changed metrics are first included in each incremental packet. The remaining space is used to include those routes whose sequence numbers have changed.

In this example, one node has changed its routing information, since it is in a new location. All nodes have transmitted new sequence numbers recently.

If there were too many updated sequence numbers to fit in a single packet, only the ones which fit would be transmitted. These would be selected with a view to fairly transmitting them in their turn over several incremental update intervals. There is no such required format for the transmission of full routing information packets. As many packets are used as are needed, and all available information is transmitted. The frequency of transmitting full updates would be reduced if the volume of data began to consume a significant fraction of the available capacity of the medium.

Destination	Metric	Sequence number
MH_4	0	S820_ MH_4
MH_1	3	S516_ MH_1
MH_2	1	S238_ MH_2
MH_3	2	S674_ MH_3
MH_5	2	S502_ MH_5
MH_6	1	S186_ MH_6
MH_7	2	S238_ MH_7
MH_8	3	S160_ MH_8

Table 4 MH_4 advertised table (updated)

Damping Fluctuations

The following describes how the settling time table is used to prevent fluctuations of routing table entry advertisements. The general problem arises because route updates are selected according to the following criteria:

- Routes are always preferred if the sequence numbers are newer;
- Otherwise, routes are preferred if the sequence numbers are the same and yet the metric is better (lower).

To see the problem, suppose that two routes with identical sequence numbers are received by a Mobile Host, but in the wrong order. In other words, suppose that MH_4 receives the higher metric next hop first, and soon after gets another next hop with a lower metric but the same sequence number. This could happen when there are a lot of Mobile Hosts, transmitting their updates not quite regularly. Alternatively, if the Mobile Hosts are acting independently and

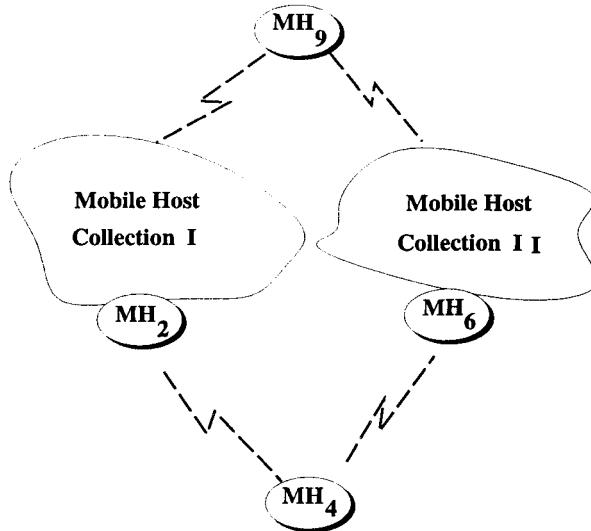


Figure 2 Receiving fluctuating routes

with markedly different transmission intervals, the situation could occur with correspondingly fewer hosts. Suppose, in any event, in Figure 2 that there are enough Mobile Hosts to cause the problem, in two separate collections of Mobile Hosts both connected to a common destination MH_9 , but with no other Mobile Hosts in common. Suppose further that all Mobile Hosts are transmitting updates approximately every 15 seconds, that Mobile Host MH_2 has a route to MH_9 with 12 hops, and Mobile Host MH_6 has a route to MH_9 with 11 hops. Moreover, suppose that the routing information update from MH_2 arrives at MH_4 approximately 10 seconds before the routing information update from MH_6 . This might occur every time that a new sequence number is issued from Mobile Host MH_9 . In fact, the time differential can be drastic if any Mobile Host in collection II begins to issue its sequence number updates in multiple incremental update intervals, as would happen, for instance, when there are too many hosts with new sequence number updates for them all to fit within a single incremental packet update. In general, the larger the number of hops, the more drastic differentials between delivery of the updates can be expected in Figure 2.

The settling time data is stored in a table with the following fields, keyed by the first field:

- Destination address
- Last settling time
- Average settling time

The settling time is calculated by maintaining a running, weighted average over the most recent updates of the routes, for each destination.

Suppose a new routing information update arrives at MH_4 , and the sequence number in the new entry is newer than the sequence number in the currently used entry but has a worse (i.e., higher) metric. Then MH_4 must use the new entry in making subsequent forwarding decisions. However, MH_4 does not have to advertise the new route immediately and can consult its route settling time table to decide how long to wait before advertising it. The average settling time is used for this determination. For instance, MH_4 may decide to delay (average_settling_time x 2) before advertising a route.

This can be quite beneficial, because if the possibly unstable route were advertised immediately, the effects would ripple through the network, and this bad effect would probably be repeated every time Mobile Host MH_9 's sequence number updates rippled through the ad-hoc network. On the other hand, if a link via Mobile Host MH_6 truly does break, the advertisement of a route via MH_2 should proceed immediately. To achieve this when there is a history of fluctuations at Mobile Host MH_4 , the link breakage should be detected fast enough so that an intermediate host in Collection II finds out the problem and begins a triggered incremental update showing an ∞ metric for the path along the way to Mobile Host MH_9 . Routes with an ∞ metric are required by this protocol to be advertised immediately, without delay.

In order to bias the damping mechanism in favor of recent events, the most recent measurement of the settling time of a particular route must be counted with a higher weighting factor than are less recent measurements. And, importantly, a parameter must be selected which indicates how long a route has to remain stable before it is counted as truly stable. This amounts to specifying a maximum value for the settling time for the destination in the settling time table. Any route more stable than this maximum value will cause a triggered update if it is ever replaced by another route with a different next hop or metric.

When a new routing update is received from a neighbor, during the same time that the updates are applied to the table, processing also occurs to delete stale entries. Stale entries are defined to be those for which no update has

been applied within the last few update periods. Each neighbor is expected to send regular updates; when no updates are received for a while, the receiver may make the determination that the corresponding computer is no longer a neighbor. When that occurs, any route using that computer as a next hop should be deleted, including the route indicating that computer as the actual (formerly neighboring) destination. Increasing the number of update periods that may transpire before entries are determined would result in more stale routing entries, but would also allow for more transmission errors. Transmission errors are likely to occur when a CSMA-type broadcast medium is used, as may well be the case for many wireless implementations. When the link breaks, an ∞ metric route should be advertised for it, as well as for the routes that depend on it.

The new routing algorithm was particularly developed for enabling the creation of *ad-hoc* networks, which are most specifically targeted for the operation of mobile computers. However, the routing algorithm itself, and the operation of an ad-hoc network, can be beneficially used in situations which do not include mobile computers. For instance, the routing algorithm could be applied in any situation where reduced memory requirements are desired (compared to link-state routing algorithms). The operation of an *ad-hoc* network could be applied to wired as well as wireless mobile computers. In general, then, we provide a new destination-sequenced routing algorithm, and this algorithm is supplemented by a technique for damping fluctuations.

5 PROPERTIES OF THE DSDV PROTOCOL

At all instants, the DSDV protocol guarantees loop-free paths to each destination. To see why this property holds, consider a collection of N mobile hosts forming an instance of an ad-hoc style network. Further assume that the system is in steady-state, i.e. routing tables of all nodes have already converged to the actual shortest paths. At this instant, the next node indicators to each destination induce a tree rooted at that destination. Thus, routing tables of all nodes in the network can be collectively visualized as forming N trees, one rooted at each destination. In the following discussion, we'll focus our attention on one specific destination x and follow the changes occurring on the directed graph $G(x)$ defined by nodes i and arcs (i, p_i^x) where p_i^x denotes the next-hop for destination x at node i . Operation of DSDV algorithm ensures that at every instant $G(x)$ is loop-free, or rather, it is a set of disjoint directed trees. Each

such tree is rooted either at x or at a node whose next-hop is *nil*. Since this property holds with respect to each destination x , all paths induced by routing tables of DSDV algorithm are indeed loop free at all instants.

Potentially a loop may form each time node i changes its next-hop. There are two cases which should be considered. First, when node i detects that the link to its next-hop is broken, the node resets p_i^x to *nil*. Clearly, this action cannot form a loop involving i . The second scenario occurs when node i receives, from one of its neighbors k , a route to x , with sequence number s_k^x and metric m , which is selected to replace the current route it has through p_i^x . Let s_i^x denote the value of the sequence number stored at node i and d_i^x denote the distance estimate from i to x just prior to receiving route from k . i will change its next-hop from p_i^x to k only if either of the following two happens.

1. the new route contains a newer sequence number, i.e., $s_k^x > s_i^x$
2. the sequence number s_k^x is same as s_i^x , but the new route offers a shorter path to x , i.e., $m < d_{ix}$

In the first case, by choosing k as its new next-hop node i cannot close a loop. This can be easily deduced from the following observation. A node i propagates sequence number s_i^x to its neighbors only after receiving it from its current next-hop. Therefore, at all times the sequence number value stored at the next-hop is always greater or equal to the value stored at i . Starting from node i , if we follow the chain of next-hop pointers, the sequence number values stored at visited nodes would form a nondecreasing sequence. Now suppose node i forms a loop by choosing k as its next-hop. This would imply that $s_k^x \leq s_i^x$. But this contradicts our initial assumption that $s_k^x > s_i^x$. Hence, loop-formation cannot occur if nodes use newer sequence numbers to pick routes.

The loop-free property holds in the second scenario due to a theorem proved by Jaffe and Moss [7], which states that in presence of static or decreasing link weights distance-vector algorithms always maintain loop-free paths.

6 COMPARISON WITH OTHER METHODS

The table 5 presents a quick summary of some of the main features of a few chosen routing protocols. The chosen set, although small, is representative of

Routing Method	Looping	Internodal Coordination	Space Complexity
Bellman Ford [5]	s/l	-	$O(nd)$
Link State [3]	s	-	$O(n + e)$
Loop-free BF [6]	s	-	$O(nd)$
RIP [4]	s/l	-	$O(n)$
Merlin Segall [9]	loop free	Required	$O(nd)$
Jaffe Moss [7]	loop free	Required	$O(nd)$
DSDV	loop free	-	$O(n)$

s - short term loop, l - long term loop

n - number of nodes, d - maximum degree of a node

Table 5 Comparison of various routing methods

a variety of routing techniques most commonly employed in operational data networks. Except for the link-state approach, all routing methods shown in the table are a variant of the basic distance-vector approach. The comparison criteria reflects some of the most desirable features that a routing algorithm should possess for it to be useful in a dynamic ad-hoc style environment. In wireless media, communication bandwidth is the most precious and scarce resource. The formation of any kind of routing loops, therefore, is highly undesirable. In the case of infrared LANS which employ a pure CSMA protocol, looping packets not only consume the communication bandwidth but they can further degrade the performance by causing more collisions in the medium. A common technique employed for loop prevention is what we call *internodal-coordination* whereby strong constraints on the ordering of the updates among nodes is imposed. The resulting internode protocols tend to be complex. Furthermore, their update coordination may restrict a node's ability to obtain alternate paths quickly in an environment where topology changes are relatively frequent. The last criteria used for comparison is the space requirement of the routing method. Nodes in an ad-hoc network may be battery powered lap-tops, or even hand-held notebooks, which do not have the kind of memory that NSFNET dedicated routers are expected to have. Therefore, economy of space is of importance.

The primary concern with using a Distributed Bellman Ford algorithm in ad-hoc environment is its susceptibility towards forming routing loops and counting-to-infinity problem. RIP [4], which is very similar to DBF algorithm, also suffers from the same problem. Unlike DBF, RIP only keeps track of the best route to each destination, which results in some space saving at no ex-

tra performance hit. RIP also employs techniques known as *split-horizon* and *poisoned-reverse* to avoid a ping-pong style of looping, but these techniques are not powerful enough to avoid loops involving more than two hops. The primary cause of loop formation in BF style algorithms is that nodes make uncoordinated modifications to their routing tables based on some information which could be incorrect. This problem is alleviated by employing an internodal coordination mechanism as proposed by Merlin and Segall in [9]. A similar technique, but with better convergence results, is developed by Jaffe and Moss in [7]. However, we do not know of any operational routing protocols which employ these complex coordination methods to achieve loop-freedom, which leads us to the conclusion that from a practical point of view the usefulness of such complex methods is diminished.

Link-state [3] algorithms are also free of the *counting-to-infinity* problem. However, they need to maintain the up-to-date version of the entire network topology at every node, which may constitute excessive storage and communication overhead in a highly dynamic network. Besides, link-state algorithms proposed or implemented to date do not eliminate the creation of temporary routing-loops.

It is evident that within ad-hoc environment design tradeoffs and the constraints under which a routing method has to operate are quite different. The proposed DSDV approach offers a very attractive combination of desirable features. Its memory requirement is very moderate $O(n)$. It guarantees loop-free paths at all instants, and it does so without requiring nodes to participate in any complex update coordination protocol. The worst case convergence behavior of the DSDV protocol is certainly non-optimal but, in the average case, it is expected that convergence will be quite rapid.

7 FUTURE WORK

There are many parameters of interest which control the behavior of DSDV, for instance the frequency of broadcast, the frequency of full routing table dumps versus incremental notifications, and the percentage change in the routing metric which triggers an immediate broadcast of new routing information. We hope to discover optimal values for many of these parameters for large populations of mobile computers by performing simulations.

Our original goals did not include making any changes to the existing idea of using the number of hops as the sole metric for making route table selections. We ended up designing a method to combat fluctuations in route tables at the mobile nodes which can be caused by information arriving faster over a path which has more hops. However, it may well be the case that such paths are preferable just because they are faster, even if more mobile computers are involved in the creation of the path. We would like to consider how to improve the routing metric by taking into account a more sophisticated cost function which includes the effects of time and cost as well as merely the number of hops.

DSDV approach relies on periodic exchange of routing information among all participating stations. An alternative is to design a system that performs route discovery on a need-to-know basis. For devices operating on limited battery power this could be an important design consideration.

A pure on-demand system operates in two phases: *route-discovery* and *route-maintenance*. A source starts the first phase by broadcasting a route discovery (RD) packet in the network. These packets are relayed by all participating nodes to their respective neighbors. As an RD packet travels from a source to various destinations, it automatically causes formation of *reverse paths* from visited nodes to the source. In order to setup a *reverse path*, a node is only required to record the address of the neighbor from which it receives the first copy of the RD packet. Any duplicates received thereafter are discarded. When the RD packet arrives at the destination, a reply is generated and forwarded along the reverse path. By a mechanism similar to reverse-path set-up, *forward route* entries get initialized as the reply packet travels towards the source. Nodes not lying on the path between source/destination pairs eventually timeout their "reverse path" routing entries. Once the path set up is complete, the route maintenance phase takes over. The second phase is responsible for maintaining paths between active source/destination pairs in face of topological changes.

8 SUMMARY

Providing convenient connectivity for mobile computers in ad-hoc networks is a challenge that is only now being met. DSDV models the mobile computers as routers cooperating to forward packets as needed to each other. We believe this innovative approach makes good use of the properties of the wireless broadcast medium. Our approach can be utilized at either the network layer (layer 3), or

below the network layer but still above the MAC layer software in layer 2. In the latter case certain additional information should be included along with the routing tables for the most convenient and efficient operation. The information in the routing tables is similar to what is found in routing tables with today's distance vector (Bellman-Ford) algorithms, but includes a sequence number, as well as settling-time data useful for damping out fluctuations in route table updates.

All sequence numbers are generated by the destination computer in each route table entry, except for the cases when a link has been broken; the latter case is described by an ∞ metric. This case is easily distinguishable, since no infinite metric will ever be generated along the tree of intermediate nodes receiving updates originating from the destination. By the natural operation of the protocol, the metric chosen to represent broken links will be superseded by real routes propagated from the newly located destination as soon as possible. Any newly propagated routes will necessarily use a metric less than what was used to indicate the broken link. This allows real route data to quickly supersede temporary link outages when a mobile computer moves from one place to another.

We have borrowed the existing mechanism of triggered updates to make sure that pertinent route table changes can be propagated throughout the population of mobile hosts as quickly as possible whenever any topology changes are noticed. This includes movement from place to place, as well as the disappearance of a mobile host from the interconnect topology (perhaps as a result of turning off its power).

In order to combat problems arising with large populations of mobile hosts, which can cause route updates to be received in an order delaying the best metrics until after poorer metric routes are received, we have separated the route tables into two distinct structures. The actual routing is done according to information kept in the internal route table, but this information is not always advertised immediately upon receipt. We have defined a mechanism whereby routes are not advertised until it is likely, based upon past history, that they are stable. This measurement of the settling time for each route is biased towards the most recent measurements for the purposes of computing an average.

We have found that mobile computers, modeled as routers, can effectively cooperate to build ad-hoc networks. We hope to explore further the necessary application-level support needed to automatically enable use of the network-

layer route capabilities to provide simple access to conferencing and workplace tools for collaboration and information sharing.

REFERENCES

- [1] W. Diepstraten, G. Ennis, and P. Belanger, "Dfwmac - distributed foundation wireless medium access control." IEEE Document P802.11-93/190, Nov 1993.
- [2] M. Schwartz and T. Stern, "Routing techniques used in computer communication networks," *IEEE Transactions on Communications*, vol. COM-28, pp. 539–552, Apr. 1980.
- [3] J. M. McQuillan, I. Richer, and E. C. Rosen, "The new routing algorithm for the ARPANET," *IEEE Transactions on Communications*, vol. COM-28, pp. 711–719, May 1980.
- [4] C. Hedrick, "Routing Information Protocol." RFC 1058, June 1988.
- [5] D. Bertsekas and R. Gallager, *Data Networks*, pp. 297–333. Prentice-Hall, Inc., 1987.
- [6] C. Cheng, R. Riley, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves, "A loop-free Bellman-Ford routing protocol without bouncing effect," in *ACM SIGCOMM '89*, pp. 224–237, Sept. 1989.
- [7] J. M. Jaffe and F. Moss, "A responsive distributed routing algorithm for computer networks," *IEEE Transactions on Communications*, vol. COM-30, pp. 1758–1762, July 1982.
- [8] J. J. G. Luna-Aceves, "A unified approach to loop-free routing using distance vectors or link states," in *ACM SIGCOMM*, pp. 212–223, 1989.
- [9] P. M. Merlin and A. Segall, "A failsafe distributed routing protocol," *IEEE Transactions on Communications*, vol. COM-27, pp. 1280–1287, Sept. 1979.

IMPROVING THE PERFORMANCE OF RELIABLE TRANSPORT PROTOCOLS IN MOBILE COMPUTING ENVIRONMENTS

Ramón Cáceres* and Liviu Iftode**

* AT&T Bell Laboratories
101 Crawfords Corner Road
Holmdel, NJ 07733, USA

** Princeton University
Department of Computer Science
Princeton, NJ 08544, USA

ABSTRACT

We explore the performance of reliable data communication in mobile computing environments. Motion across wireless cell boundaries causes increased delays and packet losses while the network learns how to route data to a host's new location. Reliable transport protocols like TCP interpret these delays and losses as signs of network congestion. They consequently throttle their transmissions, further degrading performance. We quantify this degradation through measurements of protocol behavior in a wireless networking testbed. We show how current TCP implementations introduce unacceptably long pauses in communication during cellular handoffs (800 milliseconds and longer), and propose an end-to-end fast retransmission scheme that can reduce these pauses to levels more suitable for human interaction (200 milliseconds). Our work makes clear the need for reliable transport protocols to differentiate between motion-related and congestion-related packet losses, and suggests how to adapt these protocols to perform better in mobile computing environments.

Copyright ©1995 IEEE. Reprinted, with permission, from IEEE Transactions on Selected Areas in Communications, pages 850–857, June 1995

1 INTRODUCTION

Reliable transport protocols have been tuned for networks composed of wired links and stationary hosts. They adapt to prevailing end-to-end delay conditions throughout the life of a connection, and interpret unexpected increases in delay as packet losses caused by congestion. In response to perceived losses, protocols like the Transmission Control Protocol (TCP) [9] aggressively slow their transmissions to allow the network to recover. These congestion control policies have proven beneficial in improving the overall performance of networks like the Internet. The premise underlying these policies, that packet losses are largely due to congestion, is correct for existing networks.

Future networks, however, will include wireless links and mobile hosts. In particular, there will be local-area networks composed of wireless cells of a few meters in diameter. Such *microcellular* networks are desirable for three important reasons: they offer high aggregate bandwidth, they require low power from mobile transceivers, and they provide accurate location information. Users in microcellular environments will often carry hosts across cell boundaries without warning and in the midst of data transfers.

Transport-level connections will thus encounter types of delay and loss that are unrelated to congestion. First, communication may pause while the handoff between cells completes and packets can again be routed to and from the mobile host. Second, packets may be lost due to futile transmissions over the wireless network when a mobile host moves out of reach of other transceivers, especially in networks with little or no overlap between cells. Third, packets may be lost due to the relatively frequent transmission errors suffered by wireless links. Some performance degradation due to these delays and losses is unavoidable.

These events also trigger congestion control procedures that further degrade performance. In particular, TCP implementations continually measure how long acknowledgements take to return. They maintain a running average of this delay and an estimate of the expected deviation in delay from the average. If the current delay is longer than the average by more than twice the expected deviation, TCP assumes that the packet was lost. In response, TCP retransmits the lost packet and initiates congestion control procedures to give the network a chance to recover [7]. First, TCP drops the transmission window size to reduce the amount of data in transit through the network. Second, it activates the slow-start algorithm to restrict the rate at which the window grows to previous levels. Third, it resets the retransmission timer to a backoff interval that doubles with each consecutive timeout.

When motion is mistaken for congestion, these procedures result in significant reductions in throughput and unacceptable interactive delays for active connections. The degradation is readily apparent, for example, to users of emerging ubiquitous computing environments [3].

This paper quantifies the effects of motion on throughput and delay, identifies the factors that contribute to the loss of performance, and suggests an end-to-end approach for alleviating the problem. It shows how waits for TCP's retransmission timeouts cause pauses in communication that last 0.8 seconds and longer after each cell crossing. Other researchers have called attention to the long pauses caused by TCP's exponential backoff policy [2][6][8], but to our knowledge this is the first systematic treatment of this problem. This paper also describes how using TCP's fast retransmission procedure can reduce these pauses to 0.2 seconds. We focus on TCP because it is the most widely used reliable transport protocol and will be used in at least the first generation of mobile computing environments. Furthermore, lessons learned from TCP apply to other reliable transport protocols that must deal with both mobility and congestion.

The remainder of this paper is organized as follows. Section 2 describes the wireless networking testbed used to obtain our results. Section 3 presents the measured effects of host motion on the performance of reliable transport protocols. Section 4 proposes and evaluates an end-to-end approach to alleviating the negative effects of motion. Section 5 discusses wireless transmission errors as an area for future work, and Section 6 concludes the paper.

2 WIRELESS NETWORKING TESTBED

We explore the effects of mobility through measurements of transport protocol behavior in a wireless networking testbed. The testbed consists of mobile hosts (MH), mobility support stations (MSS), and stationary hosts (SH) deployed in an ordinary office environment. Mobile hosts connect to a 2-Mbit/second WaveLAN local-area wireless network. WaveLAN is a direct-sequence spread spectrum radio product from AT&T. Stationary hosts connect to a 10-Mbit/second Ethernet local-area wired network. Mobility support stations connect to both networks. Figure 1 shows the minimum testbed configuration.

All hosts and support stations are equipped with 50-MHz i486 processors, 330-Mbyte hard disks, 16 Mbytes of memory, and the necessary network interface

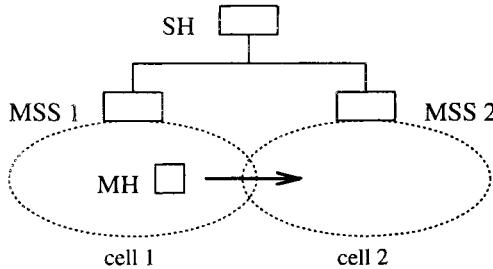


Figure 1 Wireless networking testbed

hardware. They run the 4.3BSD-Tahoe version of TCP from the University of California at Berkeley, Mobile IP software from Columbia University [6], and the Mach 3.0 microkernel and Unix server (MK77/UX37) from Carnegie Mellon University [1]. 4.3BSD-Tahoe TCP is widely used throughout the Internet and implements exponential retransmission backoffs and the slow-start algorithm.

2.1 Cellular Handoff Procedures

Each MSS defines one cell and is responsible for the MHs in its cell. It acts as the default gateway for those MHs, routing packets that originate in an MH from the wireless to the wired part of the network. Similarly, it forwards packets destined to an MH from the wired to the wireless part of the network.

MHs and MSSs collaborate to perform handoffs between cells. MSSs make their presence known by broadcasting periodic beacons over the wireless network. An MH decides to switch cells when it receives a beacon from a new MSS with a stronger wireless signal than the beacon from the old MSS, or when it receives the first beacon from a new MSS after failing to receive beacons from the old MSS.

To switch cells the MH sends a greeting packet to the new MSS, and changes its own routing tables to make the new MSS its default gateway. It also notifies the new MSS of the identity of the old MSS. The new MSS acknowledges the greeting to the MH, adds the MH to the list of MHs for which the new MSS is responsible, and begins to route the MH's packets accordingly. The new MSS also informs the old MSS that the host has moved and can be reached through the new MSS. The old MSS then adjusts its routing tables in order to forward to the new MSS any packets that arrive for the MH, and acknowledges the

handoff to the new MSS. Finally, the new MSS acknowledges the completion of the handoff to the MH. Further details of this protocol are found in [6].

2.2 Methodology

In our experiments, we initiate a reliable data transfer over a TCP connection between an MH and an SH, we cause the MH to cross cell boundaries while the connection is active, and we measure the performance of the connection.

We simulate motion across cell boundaries in software. The MH in our testbed is always in range of both MSSs, but we modified the Mobile IP software on the MH to ignore beacons from all but one MSS. After the MH spends a specified number of beaconing periods in that MSS's cell, the modified software listens for a beacon from the other MSS in order to initiate handoff procedures with the new MSS.

An important benefit of simulating motion in software is that it lets us study networks with overlapping cells as well as networks with non-overlapping cells. When adjacent cells overlap and an MH is in the region of overlap, packets can continue to flow between the MH and the old MSS while the handoff to the new MSS is in progress. When cells do not overlap, there is an unavoidable pause in network-level communication while the MH is out of reach from the old MSS and the handoff to the new MSS has not yet completed. The testbed allows us to explore the full range of handoff scenarios, from the case when the MH is in contact with both MSSs throughout the handoff, to the case when the MH cannot communicate with any MSS for an arbitrary interval of time after it leaves the old cell.

Another benefit of simulating motion in software is that it gives us precise control over the instant when handoffs begin. Under normal circumstances, handoffs begin at indeterminate times based on the time remaining in a cell's beaconing period when a host enters the cell, or on the relative strengths of two wireless signals. Our testbed makes this process deterministic and therefore allows us to reliably reproduce test conditions. Finally, simulating motion in software eliminates the need to physically move test equipment during experiments.

3 THE EFFECTS OF MOTION

We ran a number of experiments in the manner described above. We found that throughput dropped significantly in the presence of motion. We then analyzed the problem in more detail to determine the causes of the performance loss. We tracked the TCP sequence number and window size over the lifetime of a connection to determine how TCP behaved during handoffs. We also traced TCP and Mobile IP packets during the course of each handoff to determine if any packets were lost and why. This section presents our results.

Due to space limitations, we only present results for the case where data packets flow from the MH to the SH and acknowledgement packets flow from the SH to the MH. However, we also ran our experiments for the opposite case, with very similar results. We summarize our results for both cases in Section 4.4.

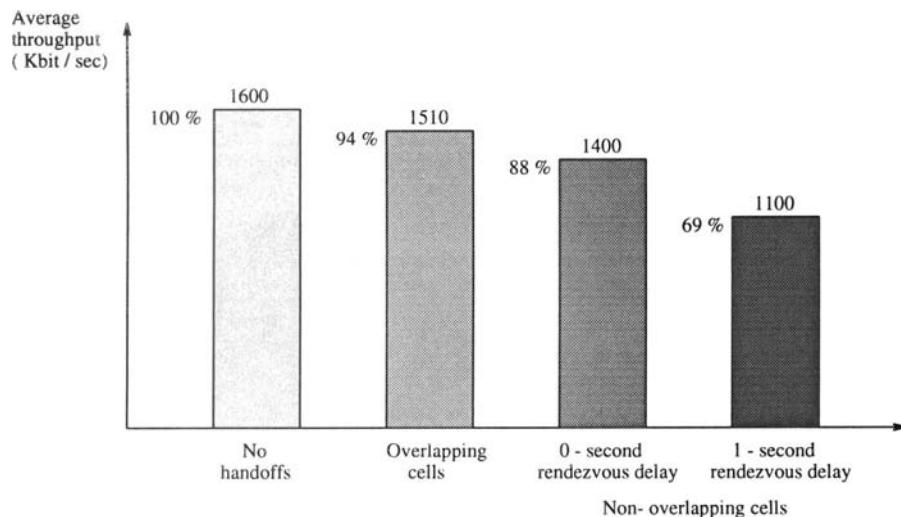


Figure 2 Loss of throughput due to host motion

3.1 Loss of Throughput

Figure 2 shows the average application-level throughput achieved when transferring 4 Mbytes of data between an MH and an SH. From left to right, the vertical bars represent the throughput obtained under four scenarios:

- The MH does not move.
- The MH moves between overlapping cells.
- The MH moves between non-overlapping cells and receives a beacon from the new MSS at the instant it leaves the old cell (0-second rendezvous delay).
- The MH moves between non-overlapping cells and receives a beacon from the new MSS one second after leaving the old cell (1-second rendezvous delay).

In the scenarios that involve motion, the beaconing period is 1 second and the MH switches cells every 8 beaconing periods. These parameters were chosen to allow TCP connections to attain maximum throughput between handoffs while also allowing us to observe multiple handoffs during a single data transfer.

We believe these four scenarios show a complete and fair picture of the problems introduced by host motion. We use the no-motion scenario as a base for comparison. The motion scenario with overlapping cells represents the best handoff performance possible with our hardware and software. It is realizable in a real network only if overlap regions are large enough, and hosts move slowly enough, for handoff operations to complete while a moving host is still in the overlap region. The scenario with zero rendezvous delay represents the minimum network-level interruption introduced by non-overlapping cell handoffs. It is realizable only if the MH does not have to wait for a beacon before it can communicate with the new MSS, for example in a network where MSSs announce their presence by means of a continuous signal. Finally, the scenario with a 1-second rendezvous delay shows what happens as the length of network-level interruptions increases. It is a realistic scenario when a periodic beaconing scheme is used, since an MH may have to wait up to a full beaconing period before it receives a beacon from the new MSS.

As shown in Figure 2, throughput degrades substantially in the presence of motion across non-overlapping cells. In the overlapping cell scenario, throughput degrades only slightly, by 6%. In the non-overlapping cell scenario with zero rendezvous delay, throughput drops by 12% even though only 3 handoffs occur in the roughly 24-second lifetime of the connection. Throughput drops much further with a 1-second rendezvous delay, by 31% with 3 handoffs in roughly 29 seconds.

In the rest of this section we study the causes of this performance degradation in increasing detail. We concentrate on single handoffs to eliminate from our

results any dependencies on the parameters of the throughput test discussed above (4 Mbytes of data with handoffs every 8 seconds). Our results will thus apply to all cell handoffs in each motion scenario.

3.2 Pauses in Communication

Figure 3 shows how the TCP sequence number behaves over the life of a connection. In this example, the MH moves between non-overlapping cells with a 1-second rendezvous delay. As shown, the sequence number ceases to advance for roughly 3 seconds after the first two cell crossings, and for roughly 1 second after the last crossing. A 3-second pause is typical of a 1-second rendezvous delay, while a 1-second pause is more typical of a 0-second rendezvous delay. During these pauses, TCP transmits no new data and transport-level communication comes to a halt.

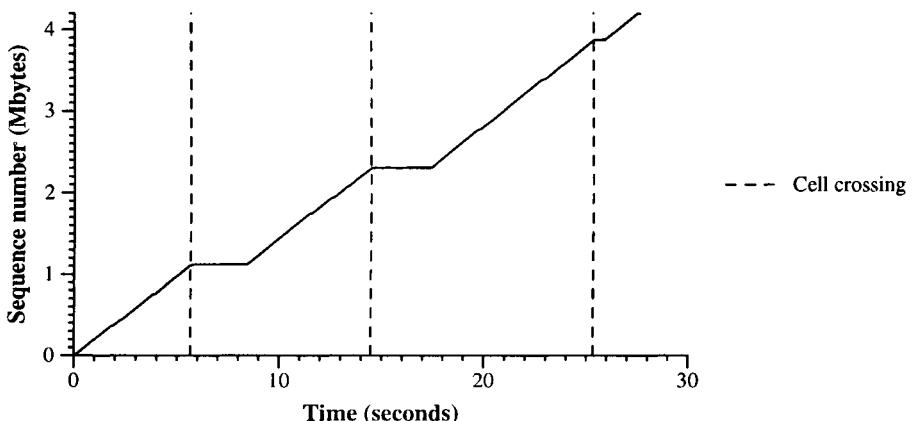


Figure 3 Behavior of TCP sequence number in response to cell boundary crossings

The effect is also visible in Figure 4, which graphs the TCP congestion window over the life of the same connection. The congestion window is an upper bound on the transmission window, which in turn controls how much unacknowledged data a TCP connection can have in transit over the network. As shown, the congestion window stops growing with every cell crossing. Some time after the crossing, the window shrinks to its minimum value and eventually begins to grow again. The intervals between when the window stops growing and when

it begins to grow again correspond to the 3-second and 1-second pauses in communication noted above.

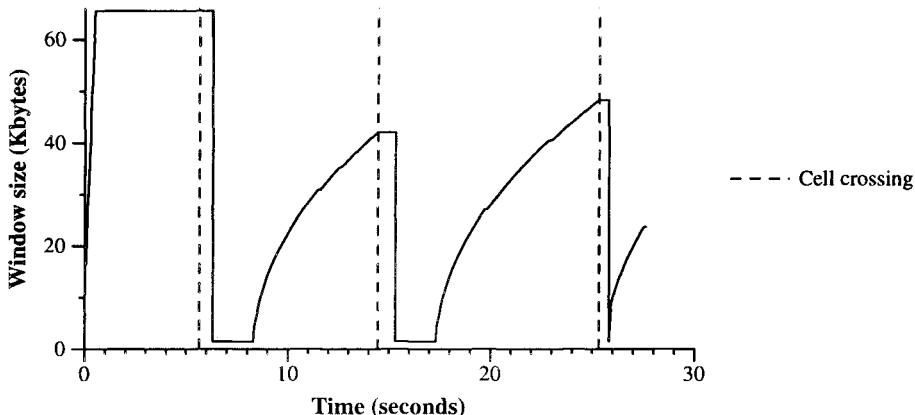


Figure 4 Behavior of TCP congestion window in response to cell boundary crossings

3.3 Packet Losses

The long pauses in communication are caused by TCP's response to packet losses. Losses occur due to routing inconsistencies during non-overlapping cell handoffs. Consider the route from the MH to the SH. When the MH leaves a cell without warning, its routing tables continue to point to the old MSS as the default gateway. The MH does not know it has moved and therefore does not change its routing tables until a beacon arrives from the new MSS. Until then, the MH continues to send packets destined for the SH directly to the old MSS. These packets are lost because the MH can no longer reach the old MSS through the wireless interface.

Inconsistencies persist longer with the route from the SH to the MH. The old MSS does not know that the MH has left the cell until an explicit notification arrives from the new MSS, which cannot send the notification before it receives a greeting from the MH. Until the old MSS learns of the MH's motion, it continues to route packets directly to the MH. These packets are also lost because the old MSS can no longer reach the MH. Any other parts of the network involved in the handoff must also wait for explicit notification that the MH has moved before they can change their routing tables to point away from the old MSS to the new MSS.

Figure 5 shows what happens during one handoff in the case of zero rendezvous delay. Although the beacon from the new MSS arrives concurrently with the cell crossing, the MH's routing tables do not point to the new MSS until 0.05 seconds after the cell crossing. Similarly, the old MSS's routing tables do not point to the new MSS until 0.15 seconds after the cell crossing. Although the system overhead implicit in these figures can be reduced through careful implementation, handoff latency cannot be altogether eliminated because at least two packet exchanges are needed to notify both the new MSS and the old MSS that the MH has changed cells. Because these packets incur unavoidable propagation delays, there will always be a window of opportunity during which both data and acknowledgement packets can be routed to unreachable wireless transceivers.

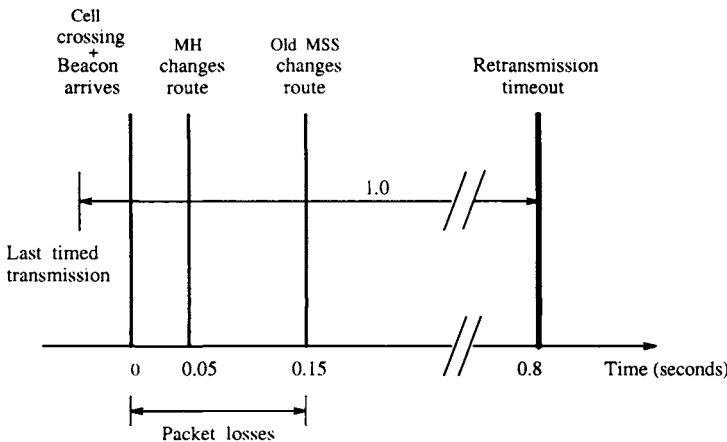


Figure 5 Handoff latency and related packet losses with a 0-second rendezvous delay

An active TCP connection thus loses up to a full transmission window's worth of packets and related acknowledgements during each handoff. Once the transmission window fills, communication stops until the retransmission timer expires. When a timeout occurs, TCP retransmits the earliest unacknowledged packet, doubles the retransmission interval, and resets the timer. If the handoff is not yet complete when the timeout occurs, the retransmitted packet is also lost and TCP waits for yet another timeout before retransmitting. A single timeout is typical of zero rendezvous delay, as shown on Figure 5. Two consecutive timeouts are typical of a 1-second rendezvous delay, as shown on Figure 6.

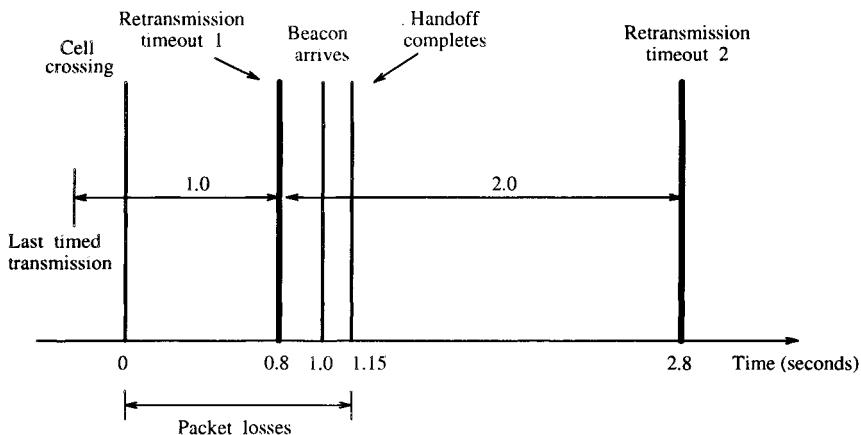


Figure 6 Handoff latency and related packet losses with a 1-second rendezvous delay

It is evident how waits for retransmission timeouts freeze transport-level communication for 0.8 seconds or more with each cell crossing across non-overlapping cells, and are responsible for a large part of the throughput losses reported earlier. In contrast, handoffs between overlapping cells do not cause the same long pauses in communication because the implementation of overlapping cells in our testbed insures that no packets are lost during those handoffs. The slight throughput losses reported earlier for the overlapping cell scenario are due to encapsulation and forwarding delays during handoffs.

3.4 Slow Recovery

As shown in Figure 4, the congestion window drops abruptly after a cell crossing when the retransmission timer goes off, but returns only gradually to its previous level once transport-level communication resumes. TCP's slow-start algorithm [7] is responsible for this behavior. As acknowledgements reach the TCP transmitter, slow-start first grows the congestion window exponentially until it reaches a threshold, then grows it linearly. The threshold is set to one half of the window size at the time of the retransmission timeout. The slow-start threshold thus decays exponentially with consecutive timeouts.

The slow recovery after each handoff contributes to the loss of throughput discussed earlier, but only moderately. Our measurements show that the algo-

rithm throttles transmissions for approximately 1 second after communication resumes. At that point the connection again reaches its maximum throughput (1.6 Mbit/second), and the congestion window ceases to affect performance.

3.5 Unacceptable Interactive Response

Interactive delays are a concern in addition to throughput. Studies of human factors indicate that people perceive interactive response to be “bad” if it takes longer than 100 to 200 milliseconds [11]. As discussed above and shown in Figures 3, 4, 5, and 6, transport-level communication comes to a halt for 800 milliseconds or longer after non-overlapping cell crossings. Furthermore, these pauses grow exponentially with growing rendezvous delays due to TCP’s exponential retransmission backoff policy. In interactive applications that use TCP for reliable data transport, user inputs and their responses will be unable to travel between mobile hosts and remote servers during these pauses.

Although users may not always interact with their computers while moving, there will certainly be times when they will do so soon after stopping. Our results show that pauses will persist from 650 milliseconds to several seconds after a host enters a new cell and the handoff completes. Motion will thus lead to unacceptable interactive response unless we solve the problems presented in this section.

4 IMPROVING PERFORMANCE

Our results demonstrate that we must improve the performance of reliable transport communication in mobile computing environments. Two approaches are possible: hiding motion from the transport level, and adapting the transport level to react better to motion.

4.1 Smooth Handoffs

Cellular networks should strive to provide smooth handoffs in order to eliminate packet losses during cell crossings and thus hide motion from the transport level. As we have shown with our testbed, one way to achieve this goal is to implement “make then break” handoffs and to engineer enough overlap between cells to insure that handoffs complete before an MH loses contact with the old

MSS. However, there are compelling reasons to build networks with little or no overlap between small cells:

- They offer high aggregate bandwidth because they can use the same portion of the electromagnetic spectrum in nearby cells. Bandwidth is scarce in wireless networks.
- They support low-powered mobile transceivers because signals need only reach short distances. Mobile computers have stringent power consumption requirements.
- They provide accurate location information because cells are small and sharply defined. Location information adds important functionality to distributed systems.

It is possible to provide smooth handoffs in spite of packet losses due to motion between non-overlapping cells. For example, MSSs could buffer packets they have recently sent to MHs. When an MSS is notified that an MH has moved out of the MSS's cell, the MSS can send the buffered packets for that MH to the MSS now responsible for the MH. The new MSS can in turn forward the packets to the MH. This technique increases the memory requirements of the MSSs, but may prove feasible because the amount of data that an MSS needs to buffer is bounded by the maximum handoff latency between adjacent cells.

However, it is unlikely that all cellular networks will provide perfectly smooth handoffs in the near future. It is therefore worthwhile to investigate transport-level techniques for alleviating the effects of packet losses during handoffs.

4.2 More Accurate Retransmission Timers

The long pauses in communication presented in Section 3 are due partly to inaccurate retransmission timers. TCP implementations historically have used coarse timers with a 300- to 500-millisecond resolution. For example, the 4.3BSD-Tahoe implementation in our testbed uses a 500-millisecond resolution timer. The resulting minimum timeout value is twice the timer resolution, or 1 second (this 1-second value is evident in Figures 5 and 6). The retransmission timer is intended to track the round-trip delay experienced by a TCP connection, but actual round-trip delays are much smaller than 500 milliseconds. For example, connections in our testbed experience well under 1

millisecond of round-trip delay. It may appear that changing TCP implementations to use higher-resolution timers would result in more accurate round-trip time estimates and would thus reduce pauses in communication during cellular handoffs.

However, more accurate timers will not solve the problems introduced by motion across wireless cell boundaries. A timer that successfully tracks the round-trip delay will lead to timeout values on the order of 1 millisecond or less. These small timeout values will result in multiple timeouts while a handoff completes, which in turn will lead to the following three problems:

- Multiple reductions of the slow-start threshold. The threshold decays exponentially with consecutive timeouts and can quickly reach the minimum window size of one packet. When communication resumes after a handoff, connections will find themselves in the linear region of window growth dictated by the slow-start algorithm, and will take many round-trip times before they reach maximum throughput. Our testbed avoided this problem because of its coarse timers.
- Multiple backoffs of the retransmission timer. Backoffs grow exponentially with consecutive timeouts and can quickly lead to the long pauses in communication we are trying to avoid.
- Multiple retransmissions before the routes become consistent. These futile retransmissions waste bandwidth in the slow wireless medium.

In general, it is difficult for a timer-based scheme to adapt to the abrupt changes in round-trip delay introduced by cellular handoffs.

4.3 Fast Retransmissions

An attractive end-to-end solution [10] to the problems presented in Section 3 is for the transport protocol to resume communication immediately after handoffs complete, without waiting for a retransmission timeout. Modern TCP implementations, including the 4.3BSD-Tahoe implementation in our testbed, already perform similar *fast retransmissions* when a transmitter receives triplicate acknowledgements from a remote receiver. When activated, the fast retransmission procedure immediately retransmits the earliest unacknowledged packet, drops the transmission window, and initiates the slow-start algorithm.

The rationale behind current fast retransmissions is that triplicate acknowledgements clearly indicate that packet loss has occurred, and thus there is no need to wait for a timeout before retransmitting.

We made modest changes to the TCP and Mobile IP software in our testbed to invoke the existing fast retransmission procedure as soon as routes become consistent following a cell crossing. First, the Mobile IP software on the MH signals the TCP software on the MH when a greeting acknowledgement arrives from the new MSS. Second, the TCP transmitter on the MH invokes the fast retransmission procedure when it receives such a signal. The signal is delivered through shared memory between TCP and IP software in the same host.

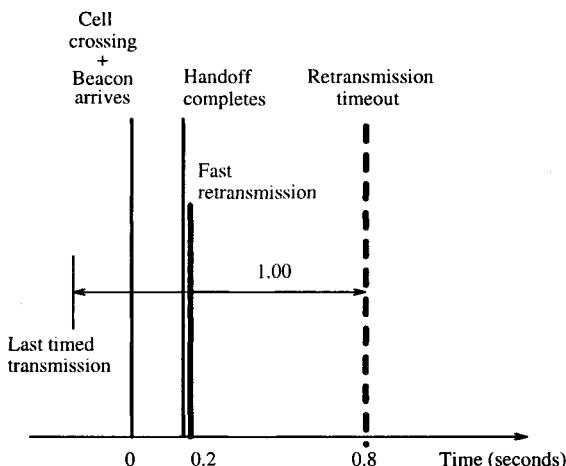


Figure 7 Fast retransmission after a handoff with a 0-second rendezvous delay

Figure 7 shows the measured effect of fast retransmissions after a non-overlapping cell handoff with a 0-second rendezvous delay. As shown, fast retransmissions cause a TCP connection to resume communication 50 milliseconds after the handoff completes. In contrast, the retransmission timeout would not have occurred until 650 milliseconds after the handoff completed.

An additional communication step is necessary to inform the TCP software on the SH of the events occurring at the other end of the connection. First, the Mobile IP software on the MH signals the TCP software on the MH of the completion of the handoff, as described above. Second, the TCP software on the MH forwards the signal over the network to the SH. Third, the TCP software on the SH invokes the fast retransmission procedure when it receives

such a signal. The signal travels from the MH to the SH through normal IP routes and can take either of two forms: It can be a specially marked TCP acknowledgement packet containing the sequence number of the last data packet successfully received by the MH, or it can be three identical but ordinary TCP acknowledgement packets. The triplicate acknowledgement approach consumes more resources but does not require modifications to TCP implementations on stationary hosts.

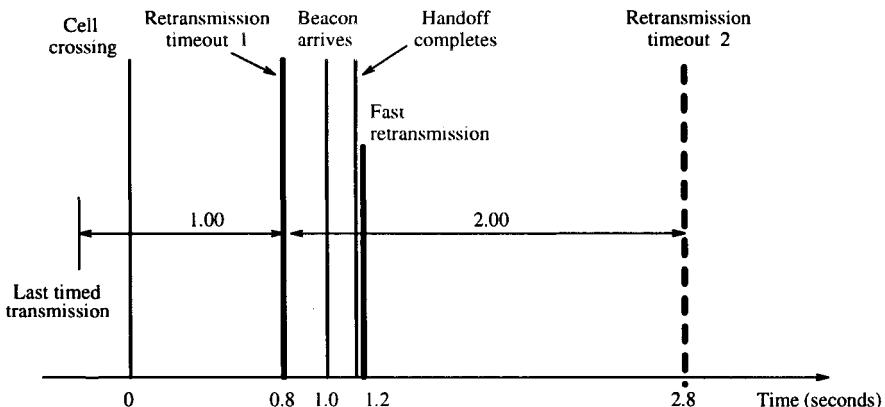


Figure 8 Fast retransmission after a handoff with a 1-second rendezvous delay

Figure 8 shows the measured effect of fast retransmissions after a non-overlapping cell handoff with a 1-second rendezvous delay. As shown, fast retransmissions again causes a TCP connection to resume communication 50 milliseconds seconds after the handoff completes. In contrast, the retransmission timeout would not have occurred until 1,650 milliseconds after the handoff completed.

The fast retransmission approach has three desirable features:

- It requires minimal changes to software on the end hosts. It changes Mobile IP only to propagate an end-of-handoff signal one layer up in the protocol stack. It changes TCP only to invoke the existing fast retransmission procedure when the end-of-handoff signal arrives. It need not change TCP on stationary hosts if triplicate acknowledgements are used.
- It does not depend on special support from the network, including mobility support stations or other intermediate routers. It therefore does not

depend on any one mobile networking environment and will work over an internetwork.

- It follows established congestion avoidance policies by closing the transmission window and using the slow-start algorithm after the initial retransmission. It thus avoids congesting the cell the MH has just entered. Gently probing the congestion state of a new route, such as the route to a new cell, is one of the principal motivations behind the slow-start algorithm.
- It preserves end-to-end reliability semantics.

It is important to note that there is no need to initiate fast retransmissions in networks that guarantee smooth handoffs, that is, in networks that never lose packets during handoffs. In that case, the MH software involved in the handoff need not signal the transport level when handoffs complete. The fast retransmission scheme therefore coexists with any handoff scheme. The software that implements the scheme resides in the transport level and is exercised only when needed.

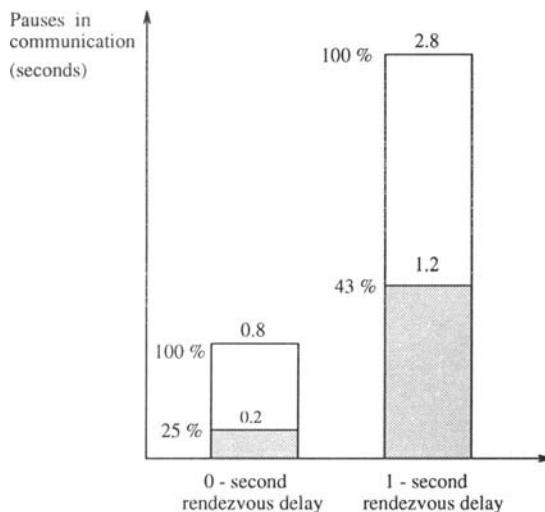


Figure 9 Improvements in latency due to fast retransmissions with the transmitter on the MH

4.4 Improvements in Latency

Figure 9 shows the pauses in transport-level communication caused by motion across non-overlapping cell boundaries, together with the improvements gained by applying the fast retransmission procedure. As shown, when the transmitter resides on the MH, fast retransmissions reduce these pauses from 0.8 to 0.2 seconds for a 0-second rendezvous delay, and from 2.8 to 1.2 seconds for a 1-second rendezvous delay.

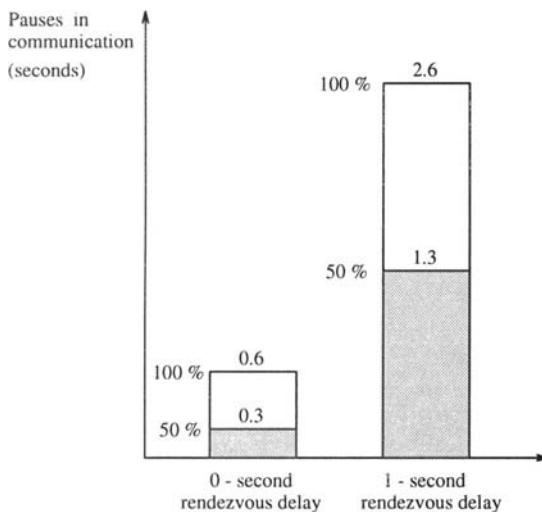


Figure 10 Improvements in latency due to fast retransmissions with the transmitter on the SH

Figure 10 shows our results for the case when the TCP transmitter resides on the SH, where pauses drop from 0.6 to 0.3 seconds for 0-second rendezvous delays, and from 2.6 to 1.3 seconds for 1-second rendezvous delays. Pauses before the improvements are shorter when the transmitter is on the SH (e.g., 0.6 vs. 0.8 seconds for 0-second rendezvous delays) because data packets incur added propagation delay before they are lost. Effectively, lost packets are sent earlier before the cell crossing, and thus retransmission timeouts occur earlier after the crossing. Pauses after the improvements are longer when the transmitter is on the SH (e.g., 0.3 vs. 0.2 seconds for 0-second rendezvous delays) because the fast retransmission must wait for an acknowledgement packet to travel between the MH and the SH after the handoff completes.

The fast retransmission scheme thus succeeds in reducing interactive delays to 200-300 milliseconds beyond the rendezvous. Reducing handoff latency through careful implementation would further reduce this remaining delay. The Mobile IP software in our testbed is an early example of support for mobile networking and was not written with fast handoffs in mind. For example, it incurs substantial system overhead by employing application-level processes to process beacons, change routes, and perform other handoff-related functions. A more efficient implementation of handoffs combined with fast retransmissions should in all cases bring pauses in communication to 100 milliseconds or less after the rendezvous. If users do not attempt to interact with their mobile computers until they stop moving across cell boundaries, interactive delays will then drop to acceptable levels.

4.5 Improvements in Throughput

We also measured significant improvements in throughput due to the fast retransmission scheme. As shown in Figure 11 for the test described in Section 3.1, throughput improves from 1400 to 1490 Kbit/second for 0-second rendezvous delays, and from 1100 to 1380 Kbit/second for 1-second rendezvous delays. Some throughput losses remain because a transport-level scheme like fast retransmissions does not reduce network-level delays and packet losses, and because the slow-start algorithm throttles connections for some time after transport-level communication resumes.

5 WIRELESS TRANSMISSION ERRORS

Even in the absence of motion, the WaveLAN network in our testbed suffers from relatively frequent packet losses due to physical transmission errors. A separate measurement study found that WaveLAN exhibited excellent packet capture rates (over 99%) in an indoor environment [5]. However, in our environment, packet loss frequency varies widely even across short distances and depends on such factors as the positions of antennas in a room. Such problems are common in wireless communication because wireless media are vulnerable to ambient noise and multipath interference. Commonly cited bit error rates for radio and infrared links are 10^{-6} or worse, compared to 10^{-12} or better for fiber optic links.

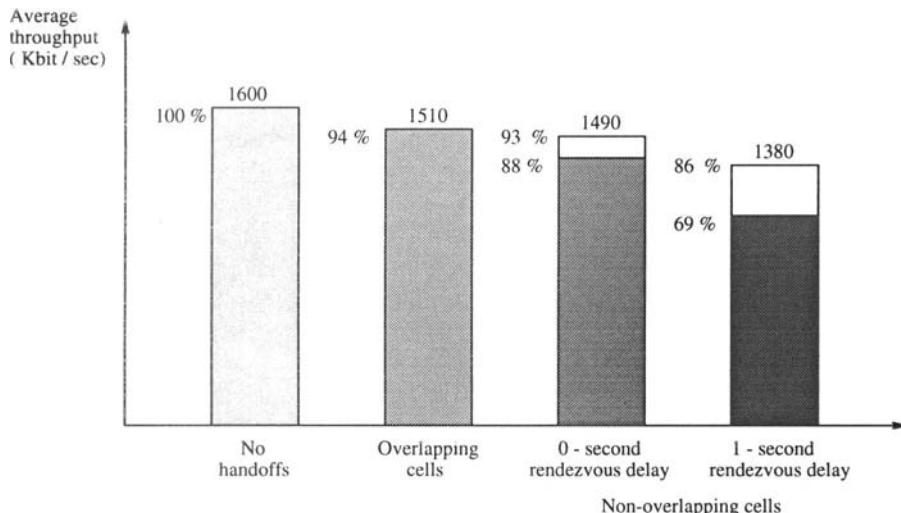


Figure 11 Improvements in throughput due to fast retransmissions

Wireless transmission errors will also trigger the transport-level problems described in Section 3. One possible solution is for the link-layer protocol that controls wireless links to retransmit packets lost on those links and thus hide the losses from higher layers. However, recent research shows that, under certain packet loss conditions, competing retransmission strategies in the link and transport layers can interact to reduce end-to-end throughput while increasing link utilization [4]. Alternative techniques such as selective retransmissions at the transport layer may prove more effective than link-layer retransmissions.

We wanted to isolate the effects of motion across cell boundaries from the effects of wireless transmission errors. We solved the problem by positioning the WaveLAN antennas physically close together in an area relatively free from ambient radiation and multipath problems. Packet losses in the absence of cell crossings then dropped to negligible levels. We also repeated all our handoff experiments using a wired network to emulate a wireless network; we substituted a second Ethernet for the WaveLAN in our testbed and found no fundamental differences in our results. We did not treat transmission errors any further in order to concentrate on handoffs. Nevertheless, the impact of wireless transmission errors on reliable transport protocols warrants further study.

6 CONCLUSIONS

Mobility changes important assumptions on which existing systems operate. In particular, networks that include wireless links and mobile hosts suffer from delays and packet losses that are unrelated to congestion. Current reliable transport protocols react to these delays and losses by abruptly slowing their transmissions, a response that further degrades the performance of active connections. We have identified the factors that contribute to this performance degradation and have quantified their effects in detail. We have shown how waits for retransmission timeouts cause pauses in communication at least 650 milliseconds longer than the underlying network-level interruption. These pauses are readily noticed by interactive users and significantly reduce throughput.

We have also described a fast retransmission scheme that can reduce the pauses in communication to 50 milliseconds past the moment when transport-level communication resumes. Fast retransmissions thus reduce interactive delays to acceptable levels and regain much of the lost throughput. The fast retransmission approach is attractive because it calls for only minimal changes to end systems, relies on no special support from the underlying network or intermediate routers, follows established congestion avoidance procedures, and preserves end-to-end reliability semantics. The approach is thus applicable to a large and varied internetwork like the Internet.

Our work makes clear the need for reliable transport protocols to differentiate between motion-related and congestion-related packet losses. Our results can be used to adapt TCP to mobile computing environments. They also apply to other reliable transport protocols that must cope with both mobility and congestion.

Acknowledgements

This work was performed at Matsushita Information Technology Laboratory. Dan Duchamp and John Ioannidis provided the Mach 2.5 version of the Mobile IP software. Chuck Lewis helped to set up and maintain the testbed. Greg Minshall provided useful comments on an earlier draft of this paper.

REFERENCES

- [1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: A new kernel foundation for UNIX development. *Proc. of the USENIX 1986 Summer Conference*, July 1986.
- [2] B. R. Badrinath, A. Bakre, T. Imielinski, and R. Marantz. Handling mobile clients: A case for indirect interaction. *Proc. of IEEE WWOS-IV*, October 1993.
- [3] S. Deering and M. Weiser. Private communication. Xerox PARC, October 1993.
- [4] A. DeSimone, M. C. Chuah, and O. C. Yue. Throughput performance of transport-layer protocols over wireless LANs. In *Proc. of Globecom '93*, December 1993.
- [5] D. Duchamp and N. F. Reynolds. Measured performance of a wireless LAN. In *Proc. of the 17th IEEE Conf. on Local Computer Networks*, September 1992.
- [6] J. Ioannidis and G. Q. Maquire Jr. The design and implementation of a mobile internetworking architecture. *Proc. of the USENIX 1993 Winter Conference*, January 1993.
- [7] V. Jacobson. Congestion avoidance and control. *Proc. of ACM SIGCOMM '88*, August 1988.
- [8] A. Myles and D. Skellern. Comparison of mobile host protocols for IP. *Journal of Internetworking Research and Experience*, 4(4), December 1993.
- [9] J. Postel. Transmission Control Protocol. *Request for Comments 793*, 1981.
- [10] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *Proc. of the 2nd International Conference on Distributed Computing Systems*, April 1981.
- [11] B. Shneiderman. *Designing the User Interface*. Addison-Wesley, 1987.

INDIRECT TRANSPORT LAYER PROTOCOLS FOR MOBILE WIRELESS ENVIRONMENT

Ajay V. Bakre and B.R. Badrinath

*Department of Computer Science
Rutgers University, New Brunswick, NJ 08903
USA*

ABSTRACT

Internetworking protocols for mobile hosts have hitherto treated host mobility as a routing problem to be handled entirely within the network (IP) layer. Such an approach however, ignores the distinctive features of wireless mobile computing. IP-based transport protocols thus suffer from poor performance when used for communication between a mobile host and hosts on the wired network. This is caused by frequent disruptions in network layer connectivity due to mobility and wireless losses. We describe indirect transport layer protocols for mobile hosts which can tackle mobility and wireless related performance problems without compromising backward compatibility with the transport protocols used over the wired network. Indirect protocols utilize the resources of Mobility Support Routers (MSRs) to provide transport layer communication between mobile hosts and those on the fixed network. We also present performance figures for I-TCP, an indirect transport protocol for mobile computers that is compatible with TCP, showing substantial improvement in throughput over regular TCP.

1 INTRODUCTION

Integration of mobile hosts into the existing internetwork consisting mostly of stationary hosts gives rise to some peculiar problems because of the special requirements of low power mobile hosts and also because of the special characteristics of the wireless link. Several Mobile-IP proposals [11, 21, 23] have addressed the problem of delivering IP packets to mobile hosts regardless of their location. In theory one can use existing fixed network transport protocols

such as UDP [19] and TCP [20] with one of the Mobile-IP proposals for communication between mobile hosts and the fixed network. This naive approach however, gives rise to performance problems, especially when a mobile host switches cells or is temporarily disconnected [7]. More seriously, all Mobile-IP proposals attempt to hide mobility, disconnection and other features of mobile wireless computing from transport and higher layers thus ruling out any specialized handling of such features. On the other hand, use of a new protocol stack for mobile hosts causes interoperability problems with the fixed network protocols. An *indirect* model for mobile hosts [3] allows the development and use of specialized transport protocols that address the performance issues on the comparatively low bandwidth and unreliable wireless link. Protocols developed using this model can also mitigate the effects of disconnections and moves while maintaining interoperability with fixed network protocols.

This paper describes indirect transport layer protocols that we have developed for a mobile wireless networking environment. Indirect protocols can interoperate with existing IP-based transport protocols used by a large number of stationary hosts over the Internet and yet allow the tuning of the transport layer for the special requirements of mobile wireless computing. The interoperability is achieved via mediation from one or more *mobility support routers (MSRs)*. We present performance figures for one such protocol implementation known as I-TCP, under simulated conditions of mobility and loss over wireless link. Experiments with I-TCP on our testbed show substantial throughput improvement over regular TCP.

The remaining of the paper is organized as follows. We first describe the system model for mobile internetworking which includes a description of our wireless testbed. Next, we outline the advantages of an indirect transport layer for mobile computers. Performance analysis of I-TCP, in comparison to regular TCP is presented. Some of the alternatives to indirect protocols including related work is described next which is followed by concluding remarks.

2 SYSTEM MODEL

Our system model for mobile internetworking consists of two separate network components:

1. The wired or fixed network which consists of local area networks interconnected by high speed links.

2. Wireless network which consists of separate wireless cells each of which is supported by a wireless base station and may be populated by a few mobile wireless hosts (MHs). We assume that each base station is directly attached to the fixed network and can route IP datagrams to and from MHs in cooperation with other base stations and thus we call it a *mobility support router* or MSR.

In our model, an MH communicates with hosts on the wired network via the MSR serving its current wireless cell. Although direct MH to MH communication is possible when they are in the same cell, we assume that in general such communication involves (at least) two MSRs currently serving the two MHs (in different cells). The MHs can freely move into other wireless cells and it is the responsibility of the MSRs to correctly route data packets to the MHs regardless of where they are located. Each MSR may maintain some state information about the MHs that are currently in its cell. Such information is handed over to the next MSR in case the MH switches cells.

2.1 Columbia Mobile IP

We use Columbia Mobile IP [11] for routing and location management of mobile hosts within a group of cells known as a *campus*. Any other mobile IP scheme that supports explicit intimation of a move from the new MSR to the old MSR can potentially be used as well. In the Columbia scheme, all the MHs in a campus and the wireless interfaces of all the MSRs are assigned IP addresses from a single subnet and the MHs can move from one cell to another in the same campus without changing their IP addresses.

In the Columbia scheme, user level processes *mhmicp* and *msrmicp* running at an MH and its MSR respectively participate in a registration protocol when an MH enters a wireless cell. The MSR periodically broadcasts a beacon in its wireless cell that is used by the mobile hosts to discover the MSR. When a mobile host moves into a cell, it responds to the MSR beacon with a greeting message that also contains the address of its previous MSR. This enables the new MSR to register the mobile host and to send a forwarding pointer to the old MSR for the MH which switched cells. All the MSRs in a campus cooperate to maintain location information about the registered MHs in their respective cells.

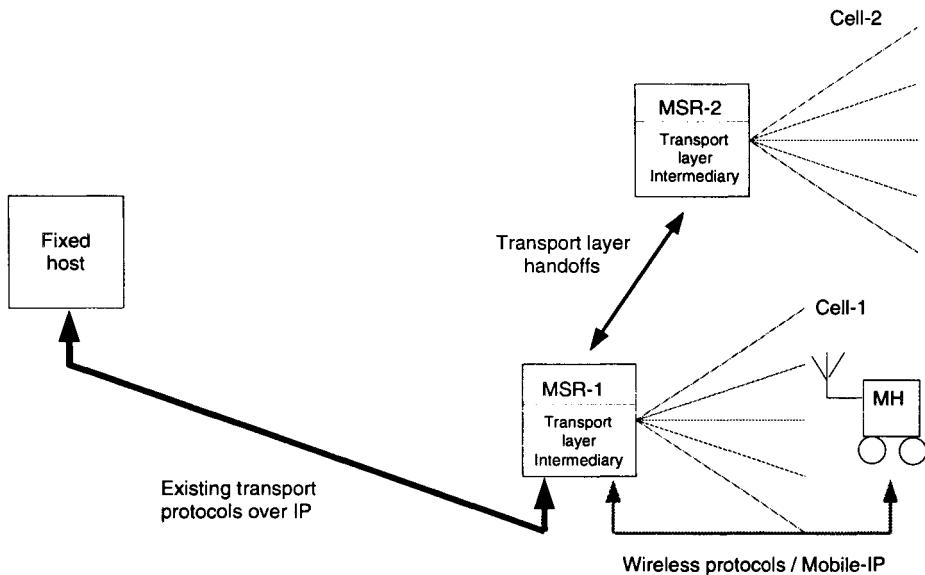


Figure 1 Indirect transport layer

2.2 Experimental wireless testbed

Our experimental wireless testbed consists of three MSRs, all of them 33 MHz 386 PC-ATs with 16 MB memory and 400 MB disk drives. These three MSRs support three wireless cells which overlap with each other. The mobile hosts used in our experiments are 66 MHz 486 PC-ATs. All the mobile hosts and MSRs are equipped with 2Mbps NCR WaveLan cards for wireless communication. The MSRs are also connected to 10 Mbps ethernet segments which are part of a single administrative domain. The MSRs run Mach 3.0 micro kernel from CMU with Unix server (MK84/UX40) [1] modified for Columbia Mobile-IP and indirect transport. Special purpose daemon processes running on the MSRs provided support for indirect transport layer connections. The MHs have similar configurations but run a slightly different version of the Unix server in addition to user level programs and libraries which together form the MH side of the indirect transport protocols.

3 INDIRECT TRANSPORT LAYER

This section gives an overview of indirect protocols [3] and describes the benefits of using indirection at the transport layer. We begin with a brief description of how mobility and the wireless medium affect transport layer performance.

3.1 Effects of wireless links and mobility

The characteristics of wireless links are very different from the wired or fixed links. The wired links (ethernet or long-haul links and ATM in the future) are becoming faster and more reliable every day whereas the wireless links (especially the outdoor links) are still much slower in comparison. The wireless links are also vulnerable to noise and loss of signal due to fading which result in much higher bit error rates than the wired links. Host mobility causes temporary disconnections and loss of packets if an MH moves to an area where signal strength is low or if it crosses cell boundaries.

For the transport layer protocols the above properties translate into longer and often unpredictable round trip delays and increased loss of data packets. The throughput of reliable transport protocols such as TCP typically depends upon accurate estimation of round trip time between the two end points. The throughput of such protocols is thus adversely affected if one of the links between the end points is a wireless link. The increased packet loss over wireless also triggers congestion control at the transmitting host which further degrades the throughput [7]. This problem is made worse by the assumption of reliable links underlying the congestion control policies [12, 18] which have evolved for the wired network.

Unreliable protocols such as UDP are mainly used by two kinds of applications – *i*) unreliable stream applications such as audio or video streams and *ii*) applications which build their own reliability mechanisms on top of the unreliable (and simpler) service provided by the transport protocol. The second class of applications includes *request-response* style of communication as in Sun's Network File System built on top of SunRPC which uses UDP for transport. The mobility and wireless related problems mentioned above cause degraded stream quality in the first kind of applications. The second kind of applications can fail in unexpected ways if an MH experiences excessive loss of data packets or disconnections because such applications are typically built for local area networks where losses and disconnections are rare.

3.2 Indirect model for mobile hosts

The indirect protocol model for mobile hosts suggests that any interaction from a mobile host (MH) to a machine on the fixed network should be split into two separate interactions – one between the MH and its mobility support router (MSR) over the wireless medium and another between the MSR and the fixed host (FH) over the wired network. This provides an elegant method for accommodating the special requirements of mobile hosts in a way that is backward compatible with the existing fixed network. All the specialized support that is needed for mobile applications as well as for the low speed and unreliable wireless medium can be built into the wireless side of the interaction while the wired side is left unchanged.

At the transport layer, use of indirection results in the following benefits:

1. Indirection separates the flow control and congestion control functionality on the wireless link from that on the fixed network.
2. A separate transport protocol for the wireless link can support notification of events such as disconnections, moves and other features of the wireless link such as the available bandwidth etc. to the higher layers which can be used by *link aware* and *location aware* mobile applications.
3. An indirect transport protocol can provide some measure of reliability only where it is needed i.e. over the wireless link for those applications which prefer to use unreliable transport over the fixed network.
4. Indirect transport protocols provide backward compatibility with the existing wired network protocols. Thus no modifications at unrelated fixed hosts are needed for accommodating mobile hosts.
5. Indirection allows the base station (mobile support router or MSR) to manage much of the communication overhead for a mobile host. Thus, a mobile host (e.g. a small palmtop) which only runs a very simple wireless protocol to communicate with the MSR can still access fixed network services such as *WWW* which may otherwise require a full TCP/IP stack running on the mobile.
6. Indirection at the MSR allows faster reaction to mobility and wireless related events compared to a scheme where the remote communicating host tries to react to such events. This is because the MSR is the router closest to the wireless link.

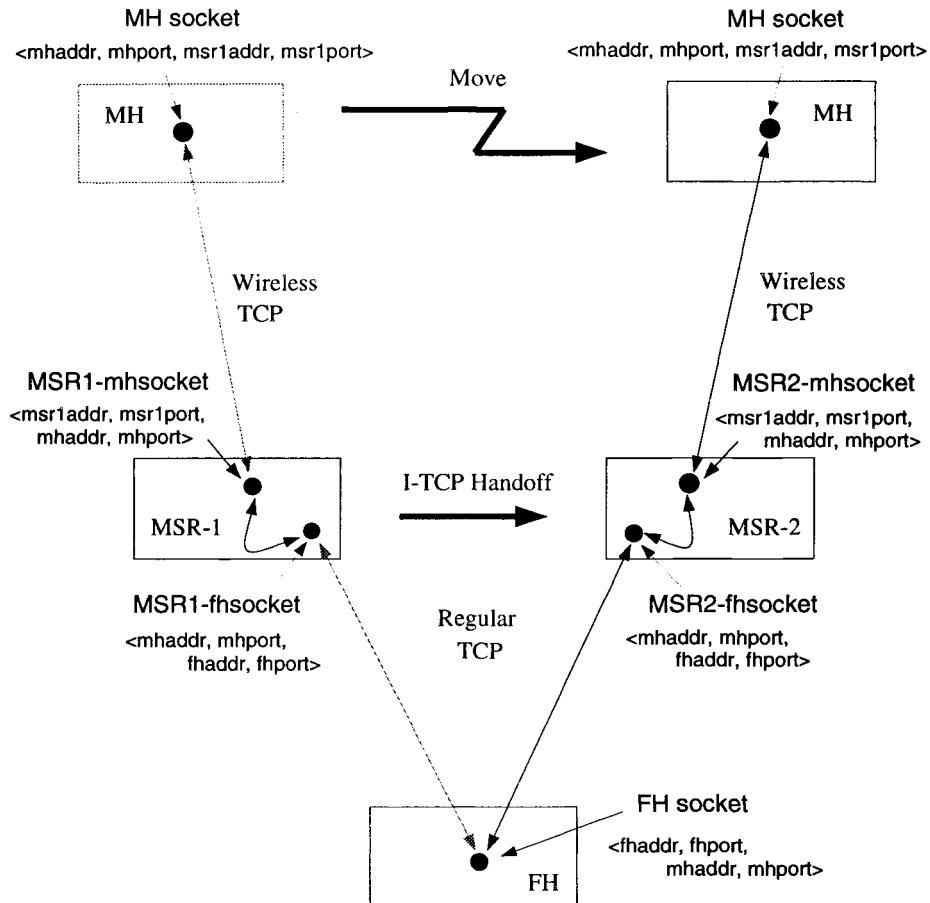


Figure 2 I-TCP connection setup

3.3 I-TCP: A Reliable Stream Protocol

I-TCP is a reliable transport layer protocol for mobile hosts which is based on the Indirect Protocol model. I-TCP is fully compatible with TCP/IP on the fixed network and is built around the following simple concepts:

1. A transport layer connection between a mobile host (MH) and a fixed host (FH) is established as two separate connections – one over the wireless

medium and another over the fixed network with the current MSR being the center point.

2. If the MH switches cells during the lifetime of an I-TCP connection, the center point of the connection moves to the new MSR.
3. The FH is completely unaware of the indirection and is not affected even when the MH switches cells i.e. when the center point of the I-TCP connection moves from one MSR to another.

When an MH wishes to communicate with some FH using I-TCP, a request is sent to the current MSR (which is also attached to the fixed network) to open a TCP connection with the FH *on behalf of* the MH. The MH communicates with its MSR on a separate connection using a variation of TCP that is tuned for wireless links and is also aware of mobility. The FH only sees an image of its peer MH that in fact resides on the MSR. It is this image which is handed over to the new MSR in case the MH moves to another cell.

As an example, figure 2 shows the setup for an I-TCP connection. The mobile host (MH) first establishes a connection with a fixed host (FH) through MSR-1 and then moves to another cell under MSR-2. When the MH requests an I-TCP connection with the FH while located in the cell of MSR-1, MSR-1 establishes a socket with the MH address and MH port number to handle the connection with the fixed host. It also opens another socket with its own address and some suitable port number for the wireless side of the I-TCP connection to communicate with the MH.

When the MH switches cells, the state associated with the two sockets for the I-TCP connection at MSR-1 is handed over to the new MSR (MSR-2). MSR-2 then creates the two sockets corresponding to the I-TCP connection with the *same endpoint parameters* that the sockets at MSR-1 had associated with them. Since the connection endpoints for both wireless and the fixed parts of the I-TCP connection do not change after a move, there is no need to *reestablish* the connection at the new MSR. This also ensures that the indirection in the transport layer connection is completely hidden from the FH. We currently use a modified version of TCP itself for the wireless part of the I-TCP connection, although in a future version we plan to use a transport protocol that is optimized for the one hop wireless link.

3.4 I-TCP semantics

Since I-TCP uses separate transport layer (TCP) connections for the wired and the wireless links, applications using I-TCP cannot rely on end-to-end transport layer acknowledgments. Many TCP-based applications such as **ftp** however, use application layer acknowledgments in addition to the end-to-end acknowledgments provided by TCP. This is at least partly because TCP does not provide a mechanism to notify a sending application when data is actually removed by the receiving application from its socket buffers. *Thus, if we assume that there are no MSR failures and that an MH does not stay disconnected from the fixed network indefinitely, using I-TCP instead of regular TCP does not compromise end-to-end reliability.* An MSR failure and subsequent reboot however, results in the loss of I-TCP connections established via that MSR whereas an end-to-end TCP connection can typically survive temporary failures of intermediate routers. I-TCP is therefore well suited for applications such as **ftp** and **Mosaic**, in which a higher layer protocol (or the user) can retry failed connections in case of (hopefully rare) MSR failures. On the other hand, those applications which depend on the end-to-end transport layer acknowledgments, e.g. **telnet** are better off using regular TCP. We expect the former kind of applications to predominate in a mobile computing environment where mobile hosts will need to access information services from the fixed network. Such applications are also typically throughput intensive which we believe will benefit the most from the use of I-TCP as opposed to interactive applications.

3.5 RDP/UDP: A Datagram Protocol

RDP/UDP is a semi-reliable datagram protocol suitable for request-response style of communication that is characteristic of RPC-based applications. This indirect protocol provides a UDP interface to the fixed hosts while using a reliable data protocol (RDP) [22] on the wireless link. The translation from RDP to UDP is performed by a daemon process running at the MSR. Client applications on mobile hosts, such as NFS clients, that wish to access UDP services on the fixed network can utilize RDP/UDP to protect themselves against excessive wireless losses. One particular example of a higher layer protocol that uses RDP/UDP can be found in M-RPC [6], which is an RPC service for mobile clients. If an MH switches cells during some interaction with a server on the fixed network, the state information related to the MH which was resident at the old MSR's RDP/UDP daemon is handed over to a similar daemon process at the new MSR which takes over all the active interactions on behalf of the

MH. As with I-TCP, the correspondent host (server) on the fixed network sees an image of the MH as its peer, communicating over plain UDP in this case.

4 IMPLEMENTATION AND HANDOFFS

We give a brief outline of various software components that are needed to support indirect transport protocols. Such support consists of the following three parts:

1. Library support at the MHs which forms the Application Programmer's Interface (API) for indirect transport layer.
2. Kernel support at the MSRs needed to perform transport layer handoffs.
3. User level daemons at the MSRs to handle the indirect connections on behalf of the MHs.

4.1 MH transport library

The transport layer library provides an API for the MH applications that is similar to the socket interface in BSD Unix [16]. Library calls are provided which emulate the functionality of **connect**, **listen**, **accept** and **close** system calls. A library call only serves as a wrapper around the corresponding system call to hide the communication with the MSR needed to establish (or close) an indirect transport connection. Once such a connection is established, regular socket calls can be used to send or receive data on the connection. The transport library also notifies the local *mhmicp* process about the endpoint parameters of indirect connections - this information is needed at the time of handoffs as described later. Currently, our scheme provides different linkable libraries for I-TCP and RDP/UDP connections.

4.2 MSR kernel support

We modified the networking module of the Unix server, which runs on top of Mach 3.0, to allow binding of the address of the mobile host requesting an indirect connection to the socket that is used by the MSR to communicate with the correspondent (fixed) host. Our scheme thus keeps *indirection* hidden

from hosts on the wired network. If an MH moves from one cell (say, under MSR-1) to another (under MSR-2), the state information related to all the indirect transport connections active at MSR-1 on behalf of the MH must be moved to MSR-2. The indirect transport connections must then be *restarted* at MSR-2. In addition, this migration of connections needs to be accomplished without any help from the fixed host at the other end of the connection. We implemented kernel calls in the form of socket **iocctl**s to support transport layer handoffs. These calls are used by MSR daemons to transfer state information from one machine to another when a mobile host switches cells. Details of the implementation describing specialized mechanisms developed for transport layer handoff can be found in [5].

4.3 MSR transport layer daemons

Transport layer daemons running on every MSR in our mobile internetworking environment perform the translation between wireless and wired parts of an indirect connection between an MH and its correspondent (fixed) host. Currently we use separate daemons for I-TCP and RDP/UDP but it is possible to integrate them into one transport layer *agent* that can provide all reasonable combinations of wireless and wired side protocols for use by MHs. It is also possible to link the daemon code with a higher layer agent process such as in the M-RPC system [6]. A transport layer daemon employs the kernel based mechanisms described above to pose as the MH process to the hosts on the fixed network by using the address and port number identifying the MH process which requested an indirect connection. The socket **iocctl**s mentioned above are used by such a daemon to exchange state information about an MH socket with a similar daemon at another MSR.

4.4 Handoff considerations

We have thus far described the various components at the MH and MSRs that support indirect transport layer connections. These components must follow a transport layer handoff protocol if an MH switches cells while it has open indirect connections with other hosts on the Internet. To minimize retransmission of lost packets, the handoff protocol must be closely tied with the mobile IP scheme used by the MSRs for routing IP datagrams to the mobile hosts within a domain. In addition, the handoff protocol must also be well integrated with similar protocols at higher layers. The handoff algorithm for I-TCP is described in [4]. We will only give a general outline of a transport layer handoff here.

1. When an MH moves to a new cell, it sends a greeting to the new MSR with any authentication information, the end point parameters for all active indirect connections and the address of its previous MSR.
2. The new MSR creates skeleton sockets for the indirect connections and requests the previous MSR to transfer state information related to those connections.
3. The previous MSR freezes the indirect connections for the MH which moved out of its cell and sends the related state information to the new MSR.
4. The new MSR restarts the indirect connections for the MH from the state information sent by the previous MSR.

The transport layer segments belonging to the indirect connections that are in transit during the handoff period are buffered (without processing) at the new MSR and are acknowledged as soon as complete state information is available for those connections at the new MSR. Measurements of handoff delay for I-TCP connections using different socket buffer sizes can be found in [5].

5 PERFORMANCE RESULTS

We present performance figures for experiments conducted using the **ttcp** benchmark which measures TCP throughput between two hosts. The throughput experiments were conducted on our wireless testbed which was described earlier.

We experimented with two distinct cases to study the performance of I-TCP for connections spanning over local area and wide area networks i.e. – *i*) when the FH to MH communication involved only a few hops within the Rutgers LCSR administrative domain and *ii*) when the FH to MH communication involved a long-haul link over the Internet.

5.1 Mobility experiments

Our experiments were inspired by similar experiments reported by Caceres and Iftode [7] to study the effect of mobility on reliable transport protocols. Figures 3 and 4 compare the end-to-end throughput of an I-TCP connection

between an MH and a fixed host (FH) with that of a direct TCP connection for local area and wide area connections respectively. In all our experiments, the FH sent a few megabytes of data (4 MB in case of local area and 2 MB in case of wide area) to the MH using a window size of 16 KB. We chose to make the MH to be the receiving host, since we expect it to be a typical situation with most mobile applications that will download more data from the fixed network rather than sending data over the uplink. The end-to-end throughput was measured at the MH under four different mobility patterns:

- i) **No Moves** – The MH stayed in one wireless cell during the lifetime of a connection.
- ii) **Moves between overlapped cells** – MH kept switching between two overlapped cells every 8 seconds such that it stayed in contact with the previous MSR during handoff. For a brief period after switching cells, the MH continued to receive packets from the previous MSR before the Mobile-IP routing adjustments took effect.
- iii) **Moves between non-overlapped cells with 0 second between cells** – In case of non-overlapped cells, the cell boundaries were sharply defined and therefore no communication was possible with the previous MSR after a move to another MSR. The MH started looking for a beacon from the new MSR immediately after a move and thus in the worst case the link layer connectivity was lost for one full interval between successive MSR beacons which was 1 second in our testbed. The cell switching again occurred every 8 seconds.
- iv) **Moves between non-overlapped cells with 1 second between cells** – Same as in iii) above but in this case the MH started looking for a beacon *1 second after* moving out of the previous cell. As in the previous case, an additional 1 second could elapse before a beacon is received by the MH and the link layer connectivity is reestablished.

In terms of coverage area, the two wireless cells used in our experiments completely overlapped. Non-overlapped cells were simulated with the two MSRs controlling the two cells transmitting using different Wavelan (MAC layer) network IDs. Cell switching was implemented in software to allow better control on the timing of cell crossovers.

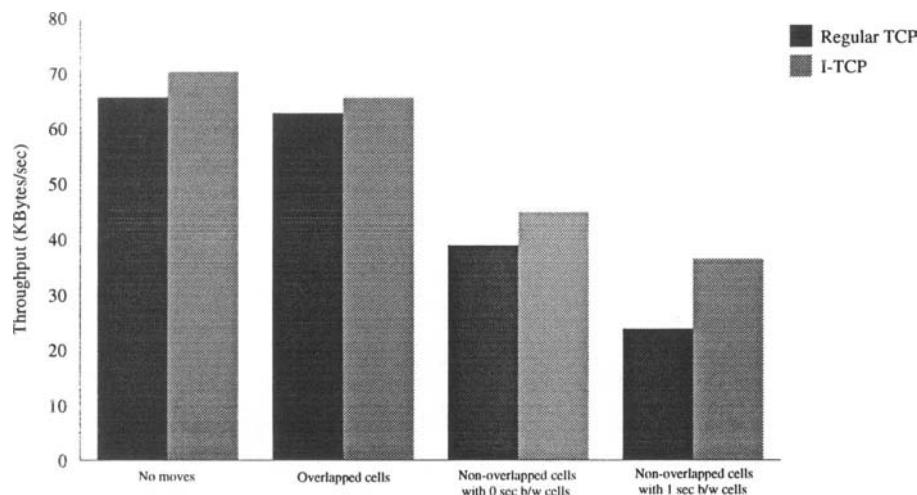


Figure 3 Local area performance comparison with mobility

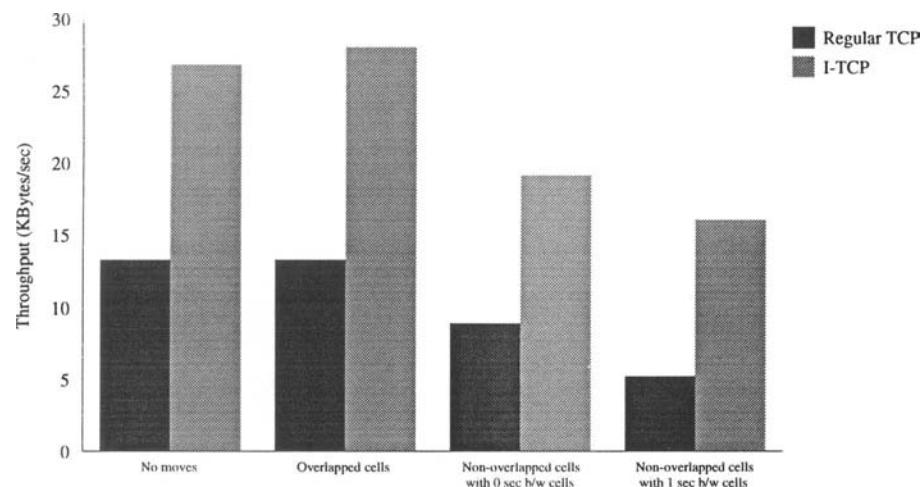


Figure 4 Wide area performance comparison with mobility

Performance over local area

With local-area experiments, we observed that I-TCP performed slightly better compared to regular TCP when the MH stayed within one cell. In the

second case when the MH switches between two completely overlapped cells, the link-layer connectivity is maintained at all times since the MH is in contact with both the new MSR and its previous MSR during handoff. There is still some degradation in TCP throughput since the TCP segments that are in transit during handoff are delayed because of IP layer routing adjustments by the MSRs. I-TCP performance suffers only marginally in this case despite the additional overhead of I-TCP state handoff between the two MSRs on every move. We believe that the main reason for the slight improvement in performance with I-TCP in the first two test cases is that the sending host (FH) sees more uniform round-trip delays for TCP segments as compared to the regular TCP. Loss of TCP segments over the wireless link, although infrequent, was also responsible for the difference in performance since I-TCP seemed to recover faster from a lost packet than regular TCP.

The two cases of non-overlapped cells where the MH temporarily lost contact with the fixed network (for 0 and 1 second respectively) before such contact was reestablished at the new MSR, affected the end-to-end throughput more severely. With regular TCP, congestion control kicked in at the FH on every handoff because of packet loss and it took some time after a cell crossover before the FH was able to send data again at full speed. In addition, the exponential back off policy of TCP resulted in the FH going into long pauses that continued even after the MH was ready to communicate in its new cell. In case of I-TCP however, a cell crossover by the MH manifested itself in the form of shrinking receive window size at the MSR which forced the FH to stop sending data when the MSR buffers were full. After a handoff, the new MSR could accept more data from the FH and the data rate on the connection quickly came back to normal. Congestion control did kick-in on the wireless link between the MSR and the MH however and so did exponential back off. We found that a simple reset of the TCP retransmission timer at the new MSR immediately after an I-TCP handoff forced the MSR to initiate a slow-start on the wireless link, and was enough to quickly get the wireless part of I-TCP out of the congestion recovery phase. In the worst case when the MH lost connectivity with the fixed network for 1 second, I-TCP showed an improvement by a factor of about 1.5 over regular TCP.

Performance over wide area

Our wide area experiments highlight the benefits of I-TCP even more clearly. Because of relatively long round-trip delays with wide area connections, any packet loss over the wireless link severely limits the end-to-end throughput of regular TCP. This is because *the time needed to recover from falsely triggered*

congestion control increases with the round-trip delay. Similarly any perturbations (such as cell crossovers or transient changes in the observed round-trip delay) have a more drastic effect over wide area connections than over local area connections. For the first two test cases i.e. when the MH stayed within once cell and when it switched between overlapped cells, the observed performance of I-TCP was about 2 times better than that of regular TCP. Since there was no packet loss because of mobility in these two cases, the performance improvement with I-TCP comes entirely from separating the TCP connections over wired and wireless links. This separation is beneficial in two respects. First, since the retransmissions due to lost segments over wireless (even though such losses are infrequent) are restricted to the wireless link, the recovery from such losses is much faster compared to the end-to-end retransmission and recovery done by regular TCP. Second factor for improvement in the throughput was the aggregating effect at the MSR which received TCP segments of size 512 bytes¹ from the FH and sent segments of 1440 bytes² over the wireless link to the MH. This points to another parameter namely the segment size, which can be tuned to suit a particular wireless link independently of the segment size chosen by the TCP implementations on the wired network. We did not observe any significant degradation in performance with the MH switching between overlapped cells either with I-TCP or with regular TCP which suggests that the effect of variation in round trip delay because of IP level routing changes was negligible for wide area connections.

In case of moves between non-overlapped cells, the throughput with regular TCP dropped to almost a third (61% degradation) of the no-moves throughput in the worst case when the MH lost contact with the fixed network for 1 second. With I-TCP, the corresponding degradation in throughput was only 40%. The net effect was that I-TCP throughput in the worst case was 3 times better than that of regular TCP. The main reason for this improved performance with I-TCP is that the retransmissions due to packets lost on the wireless link (due to moves and due to wireless errors) were confined only to the wireless part of I-TCP which can recover much faster from the congestion control phase because of the following two factors – *i*) much shorter round-trip delay between the MH and the MSR as compared to the delay between the MH and the FH and *ii*) we reset the retransmission timer at the MSR immediately after a handoff.

¹512 bytes is the default maximum segment size for wide area TCP connections.

²1460 bytes is the default maximum segment size for local area TCP connections, but we use a size of 1440 bytes in our wireless environment to avoid IP layer fragmentation in the presence of IPIP encapsulation used by Columbia Mobile IP.

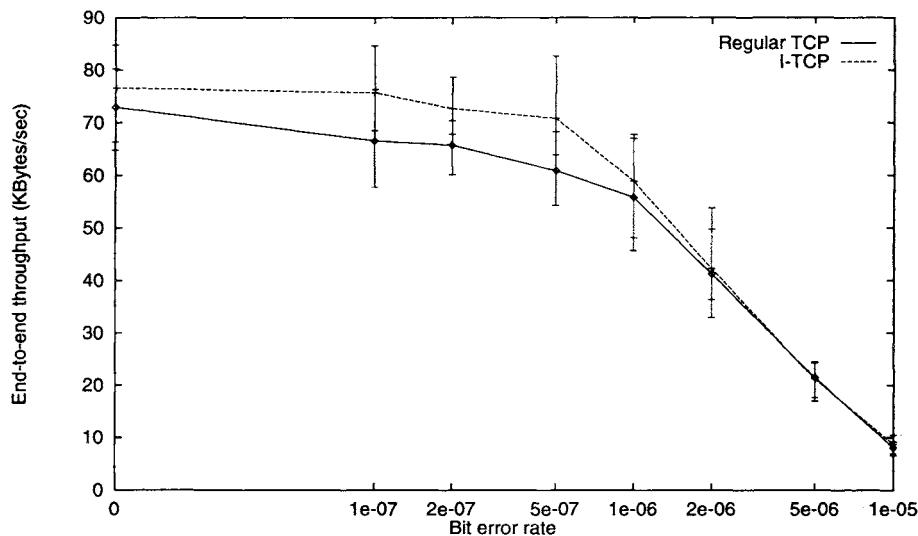


Figure 5 Local area performance comparison with wireless losses

5.2 Loss experiments

We compared the throughput of I-TCP with that of regular TCP for different bit error rates over the wireless link. The bit errors were simulated by introducing a packet dropping routine in the input processing code of the ethernet (IF) layer in Unix both at the MH and at the MSR. The packet dropping routine was based on a pseudo random number generator that produced uniformly distributed numbers in a specified range. The probability of finding a packet in error was determined from the length of the packet and the configured bit error rate. Such a simple model provides reasonable error characteristics for low error rates but becomes inaccurate as the reciprocal of the bit error rate approaches the length of the packet. Since the error simulation was used both at the MH and at the MSR, it affected the data traffic (traveling in one direction) as well as the acknowledgments (traveling in the other).

Local area experiments

Figure 5 shows the comparison of I-TCP throughput with that of regular TCP for data transfer between a fixed host (FH) and a mobile host (MH) for various

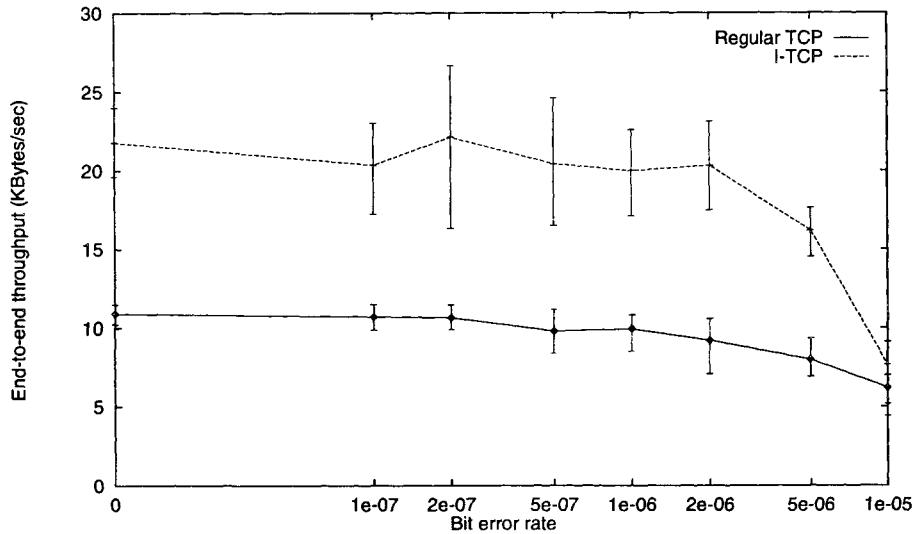


Figure 6 Wide area performance comparison with wireless losses

bit error rates (BER) when the FH is only a couple of ethernet hops away from the wireless subnet. The comparison shows that I-TCP performs better than regular TCP for error rates of up to 2×10^{-6} . For excessive wireless losses that characterize even higher error rates, the throughput of I-TCP and regular TCP is about the same. The TCP implementations used by the FH, as well as those used by the MSR and the MH in our experiments did not have the fast retransmit and fast recovery mechanisms recently proposed [14]. Also, the minimum TCP retransmission timeout was 500 msec. even for the wireless link, which does not allow early detection of lost segments. In such circumstances, there is not much difference between retransmitting lost segments from the MSR and doing the same from the FH if the FH is only a couple of ethernet hops away. Employing a TCP implementation (or a different transport protocol) for the wireless part of I-TCP that uses smaller timeouts and selective acknowledgments, should further improve I-TCP performance over regular TCP.

Wide area experiments

Figure 6 shows the comparison of I-TCP throughput with that of regular TCP when the communication between the FH and MH involved a long haul link. It

can be seen that I-TCP throughput is about twice as much as that of regular TCP for error rates of up to 5×10^{-6} . Even for a BER as high as 10^{-5} , I-TCP performance is significantly better than regular TCP. The reasons for this improvement are the same as we mentioned earlier in the context of wide area mobility experiments viz. – *i*) faster recovery from wireless losses and *ii*) aggregating effect at the MSR. The wide area experiments clearly show the potential of performance improvement with I-TCP. Just like in the local area experiments, further improvement in throughput should be possible if a better version of TCP (or a separate protocol) is used for the wireless part of the I-TCP connection.

6 ALTERNATIVES TO INDIRECT PROTOCOLS

We first list some of the approaches that other researchers have taken to solve mobility and wireless related problems that affect transport layer performance in a mobile environment. We also describe some of the desirable features that a transport protocol developed specially for mobile hosts should have.

6.1 Related Work

Thinwire protocols [10] and TCP header compression [13] can help in improving the response time of interactive applications such as **telnet** on low speed links. However, these solutions do not deal with host mobility. Link layer retransmission (LLR) can be used on error-prone wireless links to bring their error rate on par with that on the wired networks but such an approach interferes with the end-to-end retransmissions of TCP and does not always result in improved performance [8]. Fast retransmission [14] coupled with modification of the TCP software on the mobile hosts [7] solves only part of the problem because the transmitting host still performs a *slow start* if more than one segment is lost per window, thus limiting the effective throughput. A somewhat better approach that uses retransmissions on the (last) wireless link was suggested in [2] which falls somewhere between LLR and I-TCP. This approach has the advantage that end-to-end semantics of TCP are unchanged. Such an approach however, can help very little if the transmitting host on the fixed network times out while the retransmission mechanism on the last link is trying to get a data packet to the mobile host. This approach is also not very useful if the data is being sent in the other direction i.e. from the mobile to a fixed host.

Experiments with split TCP [24] have shown performance improvement over regular TCP where smaller MTU is used over the wireless part of the split connection. A similar scheme to connect mobile hosts to the Internet using digital cellular network has been described in [15]. This scheme has the advantage that a transport layer handoff is not required *every time* a mobile switches cells since the intermediate point of the split TCP connection is on a host within the cellular telephone network that is connected to multiple cellular base stations. Finally, modifications to TCP have been suggested [17], which act on mobility information from the network layer to quickly recover from TCP congestion recovery phase in case the congestion control was triggered by a move. This approach has the drawback that the proposed modifications must be applied to fixed hosts as well.

6.2 New Transport Protocols

One element that is almost universal in all the above approaches as well as in the indirect protocols is backward compatibility with the existing transport protocols such as TCP and UDP. Although desirable in an environment where mobile hosts need to communicate with hosts on the wired network, backward compatibility imposes restrictions on the extent to which a transport protocol can be modified to take mobility and wireless effects into account. In this subsection we explore how a transport protocol designed for internetworking between mobile and fixed hosts would look like if backward compatibility was not needed.

From our experience with transport protocols in a mobile wireless environment, we believe that such protocols should support the following features:

1. Selective acknowledgments to deal with losses on the wireless link and to minimize unnecessary retransmissions resulting from such losses. Reliable data protocol (RDP) supports such ACKs and there is a proposal to include selective ACKs in TCP as well.
2. Means to distinguish between losses on the wireless link and those on the wired network or better still, to distinguish between losses due to errors and those due to network congestion. This may be impossible to do if the protocol relies entirely on the communicating endpoints for such notifications; but can be accomplished easily by sending appropriate indications from an intermediate router (e.g. MSR) to the transmitting host.

3. Fast reaction to wireless losses and moves which may necessitate separate retransmission timers for the wireless link. Applications should be allowed to provide their own handlers for events such as loss of signal or moves.
4. Separate flow control on wired and wireless links since the loss characteristics of the two kinds of links are very different. It has been shown that such a separation is desirable for high packet error probabilities [9].
5. Ability to adapt to wireless links with different characteristics that may be found in different cells or by switching from an indoor wireless LAN (e.g. WaveLan) to an outdoor cellular environment (e.g. CDPD).
6. For unreliable protocols that rely on the wired links being largely error free, some kind of configurable reliability can be provided on the wireless link.

The above list indicates that even a new transport protocol without the restrictions of backward compatibility will need more support from the underlying networks (and intermediate routers) than is available from a network layer based on IP. It is also clear that accurate indications of events such as wireless errors and moves can only be provided by one or more routers in the part of the internetwork that supports mobility, i.e. by the MSRs. Thus we can safely predict that transport layer protocols built in future for mobile wireless computers will be *indirect* in some sense rather than end-to-end. This still leaves us with many different choices for the point of indirection or the intermediate point, which can either participate in the protocol as a fully empowered *agent* of a mobile host as in I-TCP or simply as an advisory entity that provides hints for improved performance. On one extreme, such an intermediary could reside on every wireless base station (MSR); on the other, it could reside on a designated router that is connected to the wired network as well as to multiple base stations in a geographic region (e.g. an MDIS in CDPD network). In the former case, a transport layer handoff or a transfer of state is potentially needed on every move whereas in the latter a handoff may be needed only if the mobile host crosses the boundary of a region. On the contrary, each MSR can make the transport connections in its cell adapt to the prevailing conditions in that cell in the former scheme which is more difficult to do in the latter.

7 CONCLUSION AND FUTURE WORK

We have described *indirection* or mediation by mobility support routers (MSRs), as a robust approach to improve transport layer performance in a mobile wire-

less environment. Our approach first confines the mobility related performance problems to the wireless link and then attempts to alleviate such problems by adapting transport layer protocols for the wireless link in a way that requires no modifications to the hosts on the fixed network.

I-TCP [4], which is a TCP compatible indirect protocol, is particularly suited for applications which are throughput intensive. Experiments with I-TCP on our testbed showed greatly improved throughput in comparison to regular TCP under simulated mobility conditions and wireless losses. The performance improvement for wide-area connections was higher than for local-area connections. We have ported some throughput intensive applications namely **ftp** and **chimera** WWW browser to use I-TCP for data transfer and observed performance improvements similar to those reported in section 5 for the **ttcp** benchmark. We have also employed RDP/UDP indirect protocol which is suitable for request-response style of communication, for our M-RPC system [6]. This protocol uses RDP for increased reliability on error-prone wireless link. We believe that *indirection* will provide the bridge between the very different worlds of mobile wireless computing and wired networks.

We are planning to build a flexible and lightweight transport protocol for the wireless side of I-TCP which can adapt to changes in the wireless environment and can support voluntary disconnections. I-TCP library at the MH can also be used to place transport layer filters [25] for indirect connections at the MSR. I-TCP handoffs allow such filters to move to the new MSR if the MH switches cells thus obviating the need for the MH to reestablish the filters in the new cell. Presentation layer services can be built on top of indirect transport layer which will allow mobile applications to dynamically choose a format for data transmitted over the wireless medium.

Acknowledgments

We are thankful to Darrell Long (UC, Santa Cruz) for allowing us to use one of the UCSC machines for our experiments with wide area connections. We would also like to thank Dan Duchamp and John Ioannidis of Columbia University for providing us with the source code of Columbia's Mobile-IP implementation and Craig Partridge of BBN Inc. for providing us with RDP sources. The Wave-Lan driver for Mach used in our experiments was originally written by Anders Klemets and was adapted for our network configuration by Girish Welling.

REFERENCES

- [1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian and M. Young, "Mach: a new kernel foundation for UNIX development", *Proc. of the USENIX 1986 Summer Conference*, July 1986.
- [2] E. Amir, H. Balakrishnan, S. Seshan and R. Katz, "Efficient TCP over networks with wireless links", *Proc. of Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, May 1995.
- [3] B.R. Badrinath, A. Bakre, T. Imielinski and R. Marantz, "Handling Mobile Clients: A Case for Indirect Interaction", *4th Workshop on Workstation Operating Systems (WWOS-IV)*, Oct. 1993.
- [4] A. Bakre and B.R. Badrinath, "I-TCP: Indirect TCP for mobile hosts", *Proc. of the 15th Intl. Conf. on Distributed Computing Systems*, May 1995.
- [5] A. Bakre and B.R. Badrinath, "Handoff and system support for indirect TCP/IP", *Proc. of the 2nd USENIX Symposium on Mobile and Location-Independent Computing*, April 1995.
- [6] A. Bakre and B.R. Badrinath, "M-RPC: A remote procedure call service for mobile clients" *To appear in Proc. of the 1st ACM Conf. on Mobile Computing and Networking*, Nov. 1995.
- [7] R. Caceres and L. Iftode, "The Effects of Mobility on Reliable Transport Protocols", *Proc. of the 14th Intl. Conf. on Distributed Computing Systems*, pp. 12–20, June 1994.
- [8] A. DeSimone, M.C. Chuah and O.C. Yue, "Throughput performance of transport-layer protocols over wireless LANs", *Proc. of Globecom '93*, Dec. 1993.
- [9] S.W. Edge, "Comparison of the hop-by-hop and endpoint approaches to network interconnection", In *Flow Control in Computer Networks*, J.-L. Grange and M. Gien eds., pp. 359–377, North Holland, 1979.
- [10] D.J. Farber, G.S. Delp and T.M. Conte, "A Thinwire protocol for connecting personal computers to the Internet", *RFC 914*, Sept. 1984.
- [11] J. Ioannidis, D. Duchamp and G.Q. Maguire, "IP-based protocols for mobile internetworking", *Proc. of ACM SIGCOMM*, pp. 235–245, Sept. 1991.
- [12] V. Jacobson, "Congestion avoidance and control", *Proc. of ACM SIGCOMM*, pp. 314–329, August 1988.

- [13] V. Jacobson, "Compressing TCP/IP headers for low-speed serial links", *RFC 1144*, Feb. 1990.
- [14] V. Jacobson, R. Braden and D. Borman, "TCP extensions for high performance", *RFC 1323*, May 1992.
- [15] M. Kojo, K. Raatikainen and T. Alanko, "Connecting mobile workstations to the Internet over a digital cellular telephone network", *Mobidata Workshop on Mobile and Wireless Information Systems*, Nov. 1994.
- [16] S.J. Leffler, M.K. McKusick, M.J. Karels and J.S. Quarterman, "The design and implementation of the 4.3BSD UNIX Operating System", Addison Wesley, 1989.
- [17] P. Manzoni, D. Ghosal and G. Serazzi, "Impact of mobility on TCP/IP: an integrated performance study", *To appear in the IEEE Journal on Selected Areas in Communications*, 1995.
- [18] J. Nagle, "Congestion control in IP/TCP Internetworks", *RFC 896*, Jan. 1984.
- [19] J. Postel, "User datagram protocol", *RFC 768*, August 1980.
- [20] J. Postel, "Transmission control protocol", *RFC 793*, Sept. 1981.
- [21] Y. Rekhter and C. Perkins, "Optimal routing for mobile hosts using IP's loose source route option", *Internet Draft*, Oct. 1992.
- [22] D. Velten, R. Hinden and J. Sax, "Reliable data protocol", *RFC 908*, July 1984.
- [23] H. Wada, T. Yozawa, T. Ohnishi and Y. Tanaka, "Mobile computing environment based on internet packet forwarding", *Proc. of the USENIX Winter Technical Conference*, Jan. 1993.
- [24] R. Yavatkar and N. Bhagwat, "Improving end-to-end performance of TCP over mobile internetworks", *IEEE Workshop on Mobile Computing*, Dec. 1994.
- [25] B. Zenel and D. Duchamp, "Intelligent communication filtering for limited bandwidth environments", *Proc. of Fifth Workshop on Hot Topics in Operating Systems (HoTOS-V)*, May 1995.

CONNECTING MOBILE WORKSTATIONS TO THE INTERNET OVER A DIGITAL CELLULAR TELEPHONE NETWORK

Markku Kojo, Kimmo Raatikainen
and Timo Alanko

*Department of Computer Science, University of Helsinki
P. O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland*

ABSTRACT

Modern portable computers and wireless connections over a cellular telephone network have created a new platform for distributed information processing. We present a communication architecture framework which makes it possible to exploit the existing TCP/IP communication architecture but which also takes into account the specific features of wireless links. Our communication architecture is based on the principle of indirect interaction. The mediating interceptor, *Mobile-Connection Host*, is the bridge between the worlds of wireless and wireline communication. The interceptor also provides enhanced functionality that improves fault-tolerance and performance for applications aware of mobility. Prototypes of the architecture are implemented both for the Unix (Linux) and for the Windows (3.11) platform.

1 INTRODUCTION

Recent developments in mobile communication and personal computer technology have created a new platform for information processing. A modern portable computer gives remarkable processing power always at hand for a nomadic user. A wireless connection gives access to information stores for the user — independent of his or her present location.

The computing environment we are interested in consists of a fixed data communication network, of portable laptop computers, and of a wireless access to that network through a cellular telephone network. Examples of such cellular networks include the Global System for Mobile Communications (GSM) [1] and the Nordic Mobile Telephone (NMT) — a digital system and an analogue system, respectively. Even in this new kind of environment it is still expected that old applications are available. From the software engineering point of view it would be desirable if all software which is sensitive to mobility could be embedded in the data communication subsystem; in other words within the physical, data link, and network layers. Such expectations are not completely unjustified. Under favorable conditions a wireless telephone link behaves like a link in a traditional Public Switched Telecommunications Network (PSTN). In addition, the cellular telephone network can — if the user so wishes — hide the user mobility. However, there are situations in which the differences between wireline and wireless links should be taken into account. It is quite possible or even probable that the existing software will work in this environment. However, the existing software may behave in a way that the user finds disrupting or even in a way that creates sheer catastrophes.

The realization of a mobile computing environment requires a communication architecture which is not only compatible with the current architectures but which also takes into account the specific features of mobility and wirelessness. Current communication architectures, such as the TCP/IP Internet protocols, are suitable for fast and reliable networks like the megabits per second LAN's. In contrast, the wireless telephone links have a low throughput, only kilobits per second, and a highly variable quality of transmission.

The cellular GSM telephone system offers a communication medium which is rather reliable when used in the so called non-transparent mode: the bit error rate is required to be less than 10^{-8} [2]. The price paid for the low bit error rate is the occurrence of highly variable transmission delays. Our performance results indicate that in favorable conditions the round-trip time is about one second and bandwidth about 900 bytes per second but in unfavorable conditions the delays can extend even to tens of seconds [3]. It should also be remembered that the conditions may become so unfavorable that a mobile telephone transmission path will be temporarily broken.

The main goals in the Mowgli¹ project are 1) to examine the behavior of data communication over cellular telephone links and 2) to develop architectural solutions to overcome difficulties typical for this kind of mobile wireless envi-

¹Mobile Office Workstations using GSM Links.

ronment. In this paper we introduce a communication architecture framework which builds a bridge between the wireline and wireless worlds. The main design goals in our approach are the following:

1. to retain the TCP/IP-based communication architecture unmodified at existing hosts of the fixed network so that neither the existing network applications nor the protocol software on fixed hosts need to be modified,
2. to provide an application programming interface which allows an easy development of new mobile client applications in a way that they are able to cooperate with existing services on fixed hosts,
3. to minimize the traffic over the wireless link and to improve fault tolerance and performance, and
4. to be able to conform with the work in progress on the Mobile IP standard [4].

These goals cannot be achieved at any single architectural level. Instead, we must consider several design issues at all architectural levels — from the data link layer up to the application layer. The most vital point of interest is the fixed-network node to which the mobile node is connected during a (GSM-) session. This mediating node has a “Janus-like” functionality. On one side it is vis-a-vis with a fragile and low-performing wireless network. On the other side it is part of a reliable and high-performance world-wide network which is robust and well-established in its functionality.

In a distributed application it would be reasonable to implement the communication between the mobile part and the fixed part using two distinct subsystems. The mediating node is a natural place for splitting the end-to-end communication into two halves. For example, this node can act as a gateway which connects two different communication protocols, each of which is suited for the underlying communication medium. Similar ideas of splitting the end-to-end TCP connection are recently presented in [5, 6, 7]. In addition, the mediating node can be used to offer more sophisticated services to the higher layers of the communication system.

As an application programming interface (API) we provide two socket interfaces depending on the platform used on the mobile node. The interfaces are downwards compatible with the Berkeley sockets [8, pp. 335–363] and the Windows Sockets [9], respectively. Hence, our API provides the functionality of reliable stream connections and of connectionless datagram delivery. In addition, our

approach provides to application programmers a possibility to implement enhanced functionality needed by users who are aware of the impacts of mobility and wireless communication on their computation.

The rest of the paper is organized as follows. In Section 2 we discuss the work in progress on mobility support for TCP/IP networking. We also relate the TCP/IP protocols to some problems of wireless links. In Section 3 we outline a socket-level solution based on the use of a mediating node between the wireline world and the wireless world. We also summarize the current (July 1995) status of implementation. In Section 4 we describe enhanced functionality in our architecture.

2 MOBILE NODES AND TCP/IP PROTOCOLS

The general structure of a distributed system with Mobile Nodes using GSM links is the basis for Figure 1. When a *Mobile Node* (MN) wants to be attached to a fixed TCP/IP network, it makes a GSM call to a fixed node which is able to provide that connection. In our model the mediating dial-in server is called the *Mobile-Connection Host* (MCH). The task of the MCH is to deliver packets to and from the Mobile Node.

2.1 Mobile IP Routing

The cellular telephone network is a straightforward way to solve the mobility problem. Using a mobile telephone an end user can always make a direct call to an MCH at the home network. This MCH acts as a dial-up IP router. IP datagrams are transmitted over the wireless link by encapsulating them within the Serial Line Internet Protocol (SLIP) [10] or the Point-to-Point Protocol (PPP) [11].

However, if the user happens to be far away from the home network, it may be economically more attractive to use an alternative solution. In this case the user chooses the closest available MCH in the fixed data communication network and dials in to this MCH. Through the MCH the Mobile Node then has a connection to any desired node within the internet. This variant has to make use of the Mobile IP techniques to manage the routing of the IP datagrams to the Mobile Node.

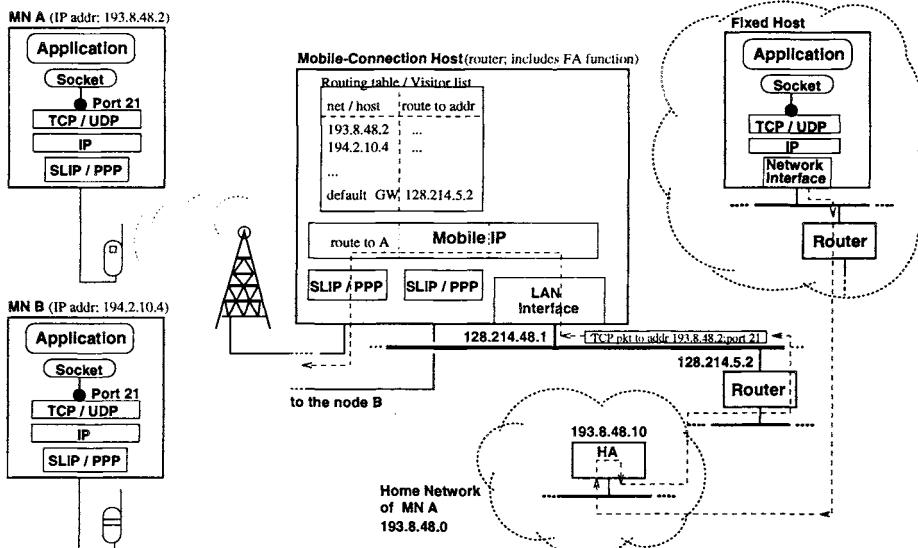


Figure 1 Routing IP datagrams to a Mobile Node

The proposed Mobile IP standard [4] specifies protocol enhancements which allow transparent routing of IP datagrams to Mobile Nodes in the Internet. The key components in the proposed standard are the *Home Agent* (HA) located at the home network of the Mobile Node and the *Foreign Agent* (FA) located at the network to which the Mobile Node is connected. When a Mobile Node attaches to a foreign network, it first finds a locally reachable Foreign Agent. Then the Mobile Node registers its current location with the Home Agent. The registration occurs through the Foreign Agent, which forwards the registration to the Home Agent. After a successful registration the Home Agent intercepts all packets addressed to the Mobile Node, encapsulates them, and tunnels them to the Foreign Agent. The Foreign Agent decapsulates the packets and delivers them to the Mobile Node. Outbound IP datagrams from the Mobile Node are delivered to their destination by using the standard IP routing. Thus, applications on the Mobile Node can communicate with the rest of the Internet as if the Mobile Node were connected to its home network. In our architecture the Foreign Agent function is provided by the MCH to which the Mobile Node dials in. After the dial-up connection to the MCH is set up, the cellular telephone system hides all further mobility of the Mobile Node.

As described above, the Mobile IP routing and the cellular telephone system can be used to solve the mobility problem. However, neither of them is able to solve the problems related to the behavior of the wireless data link: high variability in the transmission delays, which can sometimes be very long, and occasional temporary disruptions of the transmission service.

2.2 Problems in Using TCP/IP over Wireless Links

Due to the different nature of the wireline and wireless behavior, there are several drawbacks to a straightforward adoption of a cellular telephone system and the Mobile IP routing support. Firstly, there is network traffic which is unnecessary for the Mobile Node but which will be delivered over the slow wireless link if not explicitly filtered by the MCH. There is also unnecessary TCP/IP protocol header information exchanged over this link. Of course, the header information of TCP connections can be reduced by using the technique proposed by Van Jacobson [12]. This technique can be used with the SLIP framing protocol called CSLIP. The PPP protocol also supports compression of TCP and IP headers. However, reducing the header information of UDP datagrams is not supported.

Secondly, most of the traditional network applications such as FTP and mail exchange (e.g., SMTP [13]) use TCP, which is an end-to-end protocol. Therefore, all retransmissions, also those due to packets lost on the path between the MCH and the peer host in the fixed net, will cross the slow wireless link. Furthermore, link-level recovery from bursty errors may cause delays that are long enough for the TCP timers to trigger a packet retransmission. This can result in two kinds of undesirable behavior: 1) The slow wireless link has to transmit all the resent packets even though no packets were actually lost but only delayed. 2) The fixed network has the extra burden of transporting all the packets from the fixed host via the Home Agent to the MCH. It should also be noted that the reasonable packet sizes on wireline and wireless links are different.

In addition to excess data delivery, there are performance problems due to the congestion control policy of TCP [14]. If the retransmission timer goes off indicating a packet loss, the TCP sender reacts as if the reason were congestion somewhere on the route. Hence, after each timeout the TCP sender slows down its transmission rate through doubling the retransmission timeout value and through decreasing the transmission window. Thus, a temporary disruption on

the wireless link rapidly leads to long response times. Furthermore, the slow-start algorithm used by TCP is rather conservative in restoring the efficient operational transmission rate [15]. However, after the disappearance of the distortion, the wireless link is immediately capable of working at full speed.

Finally, the wireless link can cause serious problems for existing applications which depend on TCP. These problems arise when the behavior of the wireless link causes a TCP connection to break down: few applications are sophisticated enough to continue the interrupted activity. For example, a file transfer may have to be restarted from the beginning. An unnecessary break-down of the TCP connection can occur in a couple of cases. When the wireless link between the Mobile Node and the MCH breaks, many TCP/IP implementations react by terminating the end-to-end TCP connection once and for all. Even if the break-down of the wireless link would not directly cause termination of the TCP connection, the re-establishment of the dial-up connection can take a very long time causing the TCP sender to terminate the TCP connection after several retries. Exceptionally long delays on wireless links can have the same effect: after several retransmissions the TCP sender believes that a data link is broken and terminates the TCP connection. In principle, it is possible to reconfigure or dynamically adjust the TCP timeout or counter that is used to decide how long to retransmit an unacknowledged packet before terminating the connection. However, hardly any implementation of TCP allows a reconfiguration of the TCP timeout for maximum retransmits, not to mention a dynamic adjustment.

Even if it were possible to readjust the retransmission timeout, it is well known that finding a suitable timeout value is difficult. If the wireless link is broken and the timeout period is long, all pending packets from the fixed host to the Mobile Node will be retransmitted for a long time. This will inevitably cause excess load in the fixed network and in the MCH. However, the most serious aspect is that timeouts should be adjusted at both ends of the connection. This implies that the fixed node would need to know that it is dealing with a Mobile Node. Hence, the TCP protocol software on any fixed node willing to communicate with Mobile Nodes would have to be modified.

3 THE MOWGLI COMMUNICATION ARCHITECTURE

The Mowgli communication architecture is designed to increase the usability, reliability, and efficiency of client-server communication between a Mobile Node

and a fixed host. In the Mowgli architecture the basic idea is to split the channel with an end-to-end control into two parts with a store-and-forward -type interceptor. This interceptor, the Mobile-Connection Host, allows us to implement two separate but flexibly cooperating data communication subsystems: one wireline oriented, the other wireless oriented. Therefore, we can retain the existing TCP/IP-based communication infrastructure of the fixed network but we can also implement special purpose protocols for the wireless link. The interceptor also makes it possible to implement enhanced functionality on higher levels.

3.1 The Socket Abstraction and TCP/IP Protocols

A socket interface is widely used as the interface between application programs and communication protocols. Nowadays, most of the traditional TCP/IP Internet applications use TCP/IP protocols through some version of socket API.

When a socket abstraction is used for interprocess communication, data is exchanged between a pair of sockets bound to the underlying protocol. A socket interface defines functions needed to address a peer socket, create and close connections, as well as to send and receive data according the scheme defined by the underlying protocol. TCP/IP protocols define a communication endpoint, a half association, to consist of a 3-tuple: 1) an IP address, 2) a port number, and 3) a protocol (TCP or UDP). In order to accept incoming TCP connection requests or to receive UDP datagrams from a socket the application program must bind a local communication endpoint to the socket. In order to create a TCP connection or to send a UDP datagram from the socket to a foreign destination the program has to supply the communication endpoint of the desired destination so that a full association can be assigned to the socket. The full association is defined as a 5-tuple which consists of the half associations of the involved ends (see Figure 2).

3.2 Mowgli Sockets

The application programming interface offered in our communication architecture is based on the socket abstraction. The socket interface called Mowgli sockets enables application programs on a Mobile Node to use TCP/IP-based

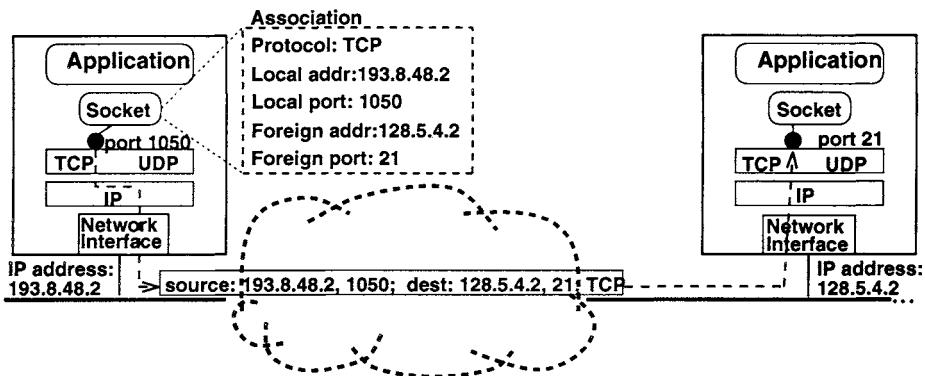


Figure 2 An association assigned to a TCP socket

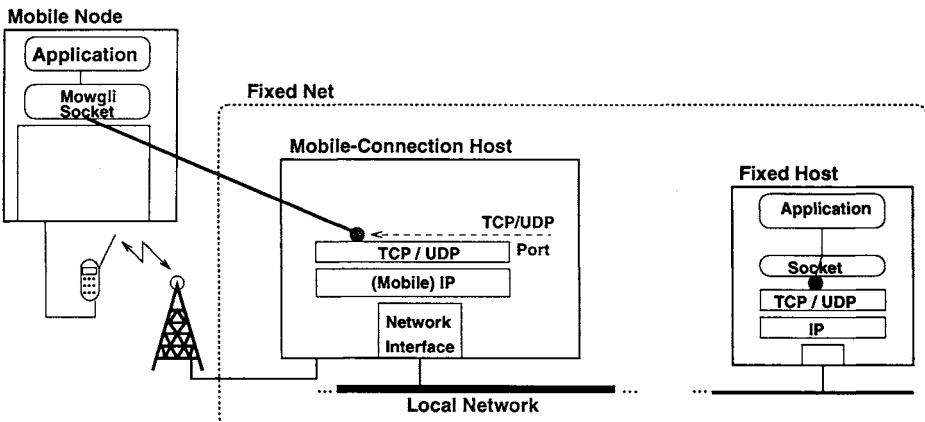


Figure 3 General view of the Mowgli socket approach

services which are available on fixed hosts although the TCP/IP protocols are not used on the wireless link between the Mobile Node and the MCH. The Mowgli socket interface is implemented in such a way that it provides the same functionality as the original stream sockets and datagram sockets, i.e. sockets for TCP and UDP protocols, respectively. Thus, applications on a Mobile Node can use TCP/IP sockets for communication but the TCP/IP stack is actually running on the MCH as exemplified in Figure 3. This is achieved by using the MCH as a socket-level gateway.

As a result, an application program on a Mobile Node can communicate through the Mowgli socket interface with the rest of the TCP/IP Internet by sending and receiving data exactly in the same way as it would do when connected directly to the Internet. The application program at the other end is unaware of the modifications that have been made to transfer data over the wireless link. This approach works well when all traffic to and from a Mobile Node traverses through an MCH as it does when wireless point-to-point link is used to connect the Mobile Node to the Internet.

When TCP is used as the transport level service, a reliable byte stream between the fixed host and the MCH is achieved as a result of standard TCP functionality. The implementation of Mowgli sockets preserves the reliability between the MCH and the Mobile Node.

3.3 Implementation Overview

An implementation outline of the Mowgli sockets is given in Figure 4. When a Mobile Node dials in to an MCH, the MCH creates a new virtual network interface and assigns the IP address of the Mobile Node to that interface. Then it creates a *proxy* that will act as the mediating agent for the Mobile Node. If the Mobile IP routing is used, the MCH registers with the Home Agent of the Mobile Node. The Home Agent tunnels IP datagrams destined for the Mobile Node to the Foreign Agent on the MCH as proposed in the Mobile IP draft. In this way a mobile user can dial in to an MCH which is not located at the user's home network.

For each Mowgli socket, created by an application program on the Mobile Node, the proxy creates a corresponding socket that will be bound to the virtual network interface having the IP address of the Mobile Node. When an IP datagram addressed to the Mobile Node arrives at the MCH, the Foreign Agent does not deliver it directly to the Mobile Node as it does in a straightforward Mobile IP implementation (see Figure 1). Instead, all IP datagrams addressed to the Mobile Node which arrive at the MCH are passed all the way up through the TCP/IP protocol stack to the corresponding socket in our approach.

The proxy is responsible for receiving data from the socket and transmitting the data over the wireless link to the Mowgli socket on the Mobile Node. Special purpose protocols are used to transmit the data over the wireless medium. The application on the Mobile Node receives the data from the Mowgli socket. Similarly, data sent from the Mobile Node through the Mowgli socket is first

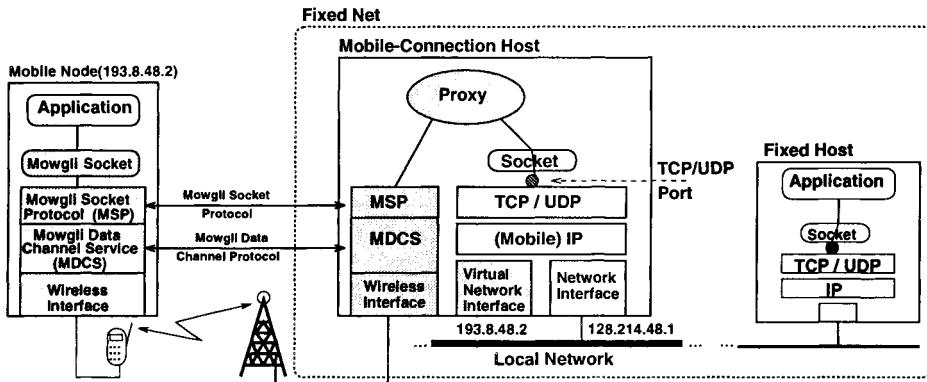


Figure 4 Implementation outline of the Mowgli sockets

transferred to the proxy on the MCH. The proxy then sends the data through the “traditional” socket on behalf of the application program on the Mobile Node. The outbound packets which carry user data originating from the Mobile Node will contain the IP address of the Mobile Node as their source address, i.e. the address assigned to the corresponding virtual network interface. Thus, the peer somewhere on the fixed network receives the data as if it were directly sent by the application program on the Mobile Node. Although the data traverses through the higher layers at the MCH, there is no significant performance degradation because the slow wireless link still remains the bottleneck.

The proxy has two basic responsibilities. The first one is to redirect data between the sockets used by applications on a Mobile Node and on a fixed host. The second one is to perform the necessary socket operations on behalf of the application on the Mobile Node. The *Mowgli Socket Protocol* (MSP) is designed to take care of these responsibilities. The MSP carries the control information between the Mobile Node and the MCH so that socket operations requested by the application on the Mobile Node can be invoked in the MCH. The MSP also carries the user data associated with the Mowgli sockets.

In order to support concurrent TCP connections several logical connections must be multiplexed over the wireless link. The *Mowgli Data Channel Service* (MDCS) provides logically independent communication channels over the wireless link. Each of these channels has its own flow control and priority. Furthermore, the MDCS provides two types of data channels: stream channels and message channels. Thus, the functionality of the TCP and UDP protocols is easy to implement.

The *Mowgli Data Channel Protocol* (MDCP) controls the delivery of data over the channels. The MDCP protocol is specially designed for data transmission over a slow wireless point-to-point link. It is a light-weight protocol: only minimal amount of protocol overhead is involved and special attention is paid to take into account the long latency of the cellular telephone links.

As the wireless medium itself is rather unreliable, a reliable link-level protocol is typically needed. The high reliability of the GSM data transmission in the non-transparent mode is achieved by running the Radio Link Protocol over the wireless link. Thus, the primary objective of the MDCP protocol in GSM environment is to take care of the recovery from link level disconnections. Although the MDCP protocol is designed for a GSM link in the first place, any existing link level protocol or a protocol specially designed for a wireless link can be used. More details about Mowgli protocols and their implementation can be found in [16].

We have implemented a prototype of our architecture on a Unix² (Linux) platform. The software for Mobile Nodes is also implemented on a Windows³ (3.11) platform. The basic functionality of the architecture was relatively straightforward to implement. In many Unix versions no kernel-level modifications should be needed on implementing the MCH software. Yet, a more convenient solution can be implemented with minor kernel-level modifications in conjunction with a Mobile IP implementation. However, the Mobile IP routing need not to be supported if the Mobile Nodes always dial in to MCHs located at their home networks.

Since we have not integrated the Mobile IP routing in our prototype, all the functions needed on the MCH are implemented as user-level software. The only exception is the virtual network interface which is based on an existing extension of the loopback network driver for Linux. The MSP protocol is embedded in the code of the proxy. The MDCP protocol is implemented as a user-level process. Similarly, on the Linux platform most software for the Mobile Node is implemented as user-level software. The operations of the MSP protocol are implemented in a user-level agent, which cooperates with the socket layer. About 500 lines of C were inserted into the Linux kernel so that the existing socket layer can interact with the MSP agent. On the Windows platform software for the Mobile Node is implemented as a dynamic-link library (DLL).

²UNIX is a registered trademark in the U.S. and other countries licensed exclusively through X/Open Company, Ltd.

³Windows is trademark of Microsoft Corporation.

4 ENHANCED FUNCTIONALITY FOR MOBILITY

In addition to the implementation of basic functionality in the Mowgli sockets, the proxy (and the agent on the Mobile Node) can perform enhanced operations for the Mobile Node. The enhanced functionality provided by the proxy and the agent can be used to reduce the amount of traffic and to control the operations crossing the slow and vulnerable wireless link. Typical examples include fault-tolerant transfer of long files, remote control of complicated data-base operations, and processing of replicated information. However, any action taken by the proxy must be either explicitly or implicitly agreed with the Mobile Node. The control over the enhanced functionality is achieved by using application-level parameters for quality of service and by adding new operations to the Mowgli socket interface. A more detailed description of enhanced functionality in the Mowgli communication architecture can be found in [16].

4.1 Improving Quality of Service

In a mobile environment the most important quality-of-service attributes at the application level are those related to reliability and performance. The functions intended for improving the quality of service can be roughly classified into two groups. The first group belongs to the general infrastructure of platforms aware of mobility. They exist to hide the effects of mobility. In terms of the ODP Reference Model [17] these functions implement essential distribution transparencies such as the location transparency and the failure transparency. The second group is a collection of functions for supporting mobility. They can be used to design enhanced application-level functionality, to help the user, either explicitly or implicitly, and to cope with problems of mobility. Using the basic functions the application programmer can create compound functions needed for the specific application.

The first group includes basic functions needed especially when old client-server applications are used in a mobile environment without any modifications. Enhanced functions which are general enough to help any type of communication are needed. For example, the wireless dial-up connection can be automatically established when needed or can be re-established after an unexpected disconnection. Similarly, the dial-up connection can on an idle wireless link can be automatically torn down after a predefined timeout and re-established when new data arrives to be delivered across the link. Furthermore, if the wire-

less connection is temporarily broken, the proxy can receive and store data for delayed transmission to the Mobile Node.

The second group provides functions which are more application oriented. Typical operations may include name-service functions modified to take mobility into account, fault-tolerant transfer of data, and control functions for remote and group operations. These operations can have attributes related to the application-level quality of service. For example, the generic MSP agent and the proxy can be replaced with a customized agent-proxy pair to improve the performance of some application-level protocols over the wireless link. The agent and the proxy can use a more sophisticated version of the application protocol to reduce the amount of data exchanged between the Mobile Node and the MCH. As an example case, we have implemented an application specific agent-proxy pair for the World-Wide Web (HTTP) [18].

4.2 Implementation of the Enhanced Functionality in the Mowgli Sockets

The Mowgli communication architecture in Mobile Nodes is shown in Figure 5. One of the basic ideas is to use a virtual layer to select between different socket implementations. Our virtual layer is called the *Mowgli Socket Layer* (MSL).

The objective of the Mowgli Socket Layer is to bind a socket to the appropriate protocol stack according to the communication media in use and to hide the slight differences between socket interfaces. The Mowgli sockets are used when the Mobile Node is operating in a “mobile” mode. A “traditional” socket interface is used when the Mobile Node is operating in a “wired” mode. The user data sent and received through Mowgli sockets is delivered either by a generic agent or by a customized agent depending on the values of the attributes.

Mobile users can use a graphical user interface to set the values of quality-of-service parameters. By setting suitable parameter values end users can adapt proxies for existing applications which use the traditional BSD or Windows sockets. Particularly, parameter values can be prespecified for the well-known ports used by common applications. In this way the user can, for example, choose an appropriate agent-proxy pair for the application, and can request a service that establishes the dial-up connection without user involvement.

Sometimes new (client) application programs specially designed for Mobile Nodes are needed to improve the usability of applications in a mobile com-

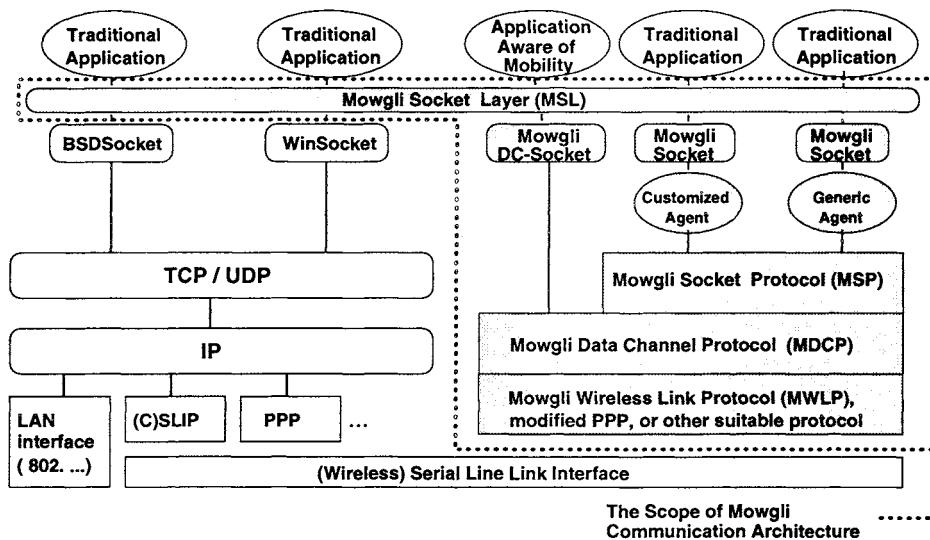


Figure 5 An overview of the implementation on a Mobile Node

munication environment. When new client programs for Mobile Nodes are implemented, the functionality of customized agent can be integrated into the client software. These applications can directly use the channels provided by the Mowgli Data Channel Service. Application programmers have a new type of socket called *Mowgli DC-socket* as an application programming interface to the MDCS. The same interface is also used when customized agents and proxies for existing applications are implemented.

Enhanced operations specified in the MDCS are especially intended for accommodating the indirect interaction model of communication. The operations can be used when new kinds of “compound” client programs are implemented. In such a “compound” program the client on the Mobile Node and the specialized proxy on the MCH are closely combined together. Together the client and proxy act as a client of an existing fixed server. Therefore, most of the communication with the server can be done by the proxy. Furthermore, the application on the Mobile Node can directly control the operations over the wireless link.

As an example we can take distributed transaction processing with the client on a Mobile Node. The mobile-node resident part of the application specifies a particular transaction. This specification is then sent to the application-specific

proxy in the MCH. The proxy takes over the responsibility for the transaction, and the connection between the Mobile Node and the MCH can be terminated. The client on the Mobile Node may later ask the proxy to give the results of the transaction, or the proxy can call back to inform the client.

5 DISCUSSION

We have presented a communication architecture which builds a bridge between the worlds of wireless and wireline communication. Our architecture retains the TCP/IP-based communication architecture unmodified at existing hosts of the fixed network so that neither the existing network applications nor the protocol software on fixed hosts need to be modified. In addition, our approach makes it possible to accommodate existing application protocols to the mobile environment with help of application specific mediators. The application programming interface provides not only the functionality of the standard TCP/IP socket interface but it also allows an easy development of new mobile (client) applications which cooperate with existing services on fixed hosts.

We have paid particular attention to fault-tolerance and performance. Our implementation of the architecture removes the unnecessary traffic due to TCP/IP from the wireless link. It also improves the quality of service at application level. The bottom line in our architecture is to split the channel with an end-to-end control into two parts with a store-and-forward -type interceptor. Communication between a mobile client and a fixed server is based on a higher level communication model of indirect client-server interaction. The model of indirect client-server interaction involves two separate interactions, which can be handled in different ways, and a mediator between these two hops.

Acknowledgements

The authors are grateful for the fruitful discussions with Heimo Laamanen, Marko Moilanen, and Henry Tirri. The Mowgli research project is supervised by professor Martti Tienari and funded by Digital Equipment Corporation, Nokia Mobile Phones, Nokia Telecommunications, Telecom Finland, and the Finnish Ministry of Education.

REFERENCES

- [1] Rahnema, M., "Overview of the GSM System and Protocol Architecture," IEEE Communication Magazine 31(4), April 1993, pp. 92–100.
- [2] "Quality of Service," GSM specification 02.08, version 3.0.0., ETSI/TC GSM, March 1990.
- [3] Alanko, T., Kojo, M., Laamanen, H., Liljeberg, M., Moilanen, M., and Raatikainen, K., "Measured Performance of Data Transmission Over Cellular Telephone Networks," Computer Communication Review 24(5), October 1994, pp. 24–44.
- [4] Perkins, C. (ed.), "IP Mobility Support," Internet Draft, IETF, March 1995.
- [5] Badrinath, B. R., Bakre, A., Imielinski, T. and Marantz, R., "Handling Mobile Clients: A Case for Indirect Interaction," In Proc. of the 4th Workshop on Workstation Operating Systems (WWOS-IV), Napa, Calif., 1993.
- [6] Bakre, A. and Badrinath, B.R., "I-TCP: Indirect TCP for Mobile Hosts," In Proc. IEEE 15th International Conference on Distributed Computer Systems, Vancouver, Canada, May 1995.
- [7] Yavatkar, R. and Bhagawat, N., "Improving End-to-End Performance of TCP over Mobile Internetworks," In Proc. IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, Calif., December 1994.
- [8] Comer, D.E., "Internetworking With TCP/IP: Principles, Protocols, and Architecture, 2nd Ed.," Prentice-Hall, 1991.
- [9] Hall, M., Towfig, M., Arnold, G., Treawell, D., and Sanders, H., "Windows Sockets: An Open Interface for Network Programming under Microsoft Windows," Version 1.1, January 1993.
- [10] Romkey, J., "A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP," Request for Comments 1055, Network Information Center, June 1988.
- [11] Simpson, W., "The Point-to-Point Protocol (PPP)," Request for Comments 1661, Network Information Center, July 1994.
- [12] Jacobson, V., "Compressing TCP/IP Headers for Low-Speed Serial Links," Request for Comments 1144, Network Information Center, February 1990.

- [13] Postel, J., "Simple Mail Transfer Protocol," Request for Comments 821, Network Information Center, August 1982.
- [14] Jacobson, V., "Congestion Avoidance and Control," In ACM SIGCOMM'88 Symposium on Communications Architectures and Protocols, Stanford, Calif., August 1988.
- [15] Cáceres, R. and Iftode, L., "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments," IEEE Journal on Selected Areas in Communications, 1995.
- [16] Kojo, M., Alanko, T., Liljeberg, M., and Raatikainen, K., "Enhanced Communication Services for Mobile TCP/IP Networking," Technical Report C-1995-15, Univ. of Helsinki, Dept. of Computer Science, April 1995.
- [17] "Draft Recommendation X.903: Basic Reference Model of Open Distributed Processing - Part 3: Architecture," Draft ITU-T Recommendation, ISO/IEC JTC1/SC21/WG7, March 1995.
- [18] Liljeberg, M., Alanko, T., Kojo, M., Laamanen, H., and Raatikainen, K., "Optimizing World-Wide Web for Weakly Connected Mobile Workstations: An Indirect Approach," In Proc. of 2nd Int. Workshop on Services in Distributed and Networked Environments, Whistler, Canada, June 1995.

ASYNCHRONOUS VIDEO: COORDINATED VIDEO CODING AND TRANSPORT FOR HETEROGENEOUS NETWORKS WITH WIRELESS ACCESS

Johnathan M. Reason, Louis C. Yun,
Allen Y. Lao, and David G. Messerschmitt

*Electrical Engineering Division, Department of EECS
University of California, Berkeley, California 94720
USA*

ABSTRACT

Wireless access to continuous-media services such as video, voice, and audio is becoming increasingly prevalent. Interactive video services such as video conferencing and multimedia editing are two such services of particular interest. We discuss some of the problems with using the MPEG standard (which was designed for wired, circuit-switched services) for wireless packet-video transport in a mobile environment. We propose a novel strategy for video transport using a layered source coder in conjunction with a variable QOS, multiple-substream abstraction for the transport. This abstraction addresses specifically the need to obtain simultaneously high spectral efficiency, good subjective quality, and low perceptual delay on a wireless channel. It also addresses the heterogeneous transport resulting from the concatenation of a wireless access link with a broad-band backbone network.

We use asynchronous video (ASV) reconstruction, running counter to current techniques, which use strictly synchronous (frame-by-frame) video processing. By doing so, we hope to achieve a perceptual delay that is much lower than the worst-case transport delay. By *perceptual delay*, we refer to the effective end-to-end latency observed by the user, for example as represented by the audio delay required to maintain lip synchronization. By identifying packets to the transport with relaxed reliability and/or delay requirements, the transport (particularly wireless) can achieve high traf-

fic capacity. Reasonable and promising simulation results are achieved, although much work remains on achieving significant video compression in this environment.

1 INTRODUCTION

Wireless access to wired backbone networks will be a common model for the future, resulting in a heterogeneous network infrastructure. Typically the wireless access will be the bottleneck for such a heterogeneous network, due to its scant bandwidth allocation, propagation impairments (e.g., multipath fading), and interference. Thus, our focus in this research is the provision of video services over this heterogeneous network, with emphasis on the issues raised by the wireless access.

In continuous-media (CM) services such as video, the only meaningful criterion for evaluation of the service quality is the subjective quality. The primary subjective impairments will be time jitter, latency or delay, and artifacts introduced by loss mechanisms in the compression and the transport. Another important consideration will be cost, as represented in part by the traffic capacity consumed for the service, particularly on the wireless access link.

In this mix of issues, we believe that delay has been largely overlooked. For interactive applications such as video conferencing and multimedia editing, low delay is crucial to user acceptance, and yet current trends run directly counter to achieving low delay. Existing examples of wireless audio/voice access, such as digital cellular, use transcoders (conversions from one audio compression standard to another) in the basestation, which add significant delay. A similar approach for video would have the same problem. On variable-delay transport, typical of packet/cell networks, existing video compression algorithms synchronously reconstruct video frames, implying a delay that includes the *worst-case* transport delay plus any compression/decompression delay.

A primary goal of this research is to minimize the *perceptual delay* for video transport. By perceptual delay, we mean the delay perceived by the user, for example as would be indicated by the audio delay required to achieve lip synchronization in a video conferencing application. We reduce this perceptual delay by two related techniques. First, we code the video in such a way that transcoders are not required within the network; that is, we transport the video transparently from network edge to edge. Second, we do this in such a

fashion that the perceptual delay is *less* than the transport worst-case delay plus compression/decompression delay.

The second objective is achieved by reconstructing the video presentation at the receiver asynchronously. That is, we identify information within the video that can be delayed from its original frame to later frames, without significant artifacts being introduced. For example, information elements in areas of the screen with low motion are more tolerant to delay than areas of high motion. As another example, the high-resolution information in areas with little motion can be delayed relative to the low-resolution information. (This is closely related to the well-known technique of progressive image transmission.) When the video is asynchronously reconstructed, there is no longer a clear *objective* definition of delay, but rather we have to rely on the vague notion of perceptual delay, a quantity that can be determined only by subjective testing.

Decoupling perceptual delay from worst-case transport delay leads to another opportunity. By allowing the worst-case transport delay to increase (without affecting perceptual delay), we can actually increase the network traffic capacity. This will be particularly advantageous on wireless access links.

2 MPEG AND MOBILE CHANNELS

Given the tremendous volume of activity by the MPEG committee in establishing a video compression standard, the burning question may be, why not use MPEG? In fact, with a compression ratio of about 100:1, MPEG may at first glance seem ideal for networks with bandlimited wireless links. In a wireless link, however, bandwidth is not the only scarce resource: high reliability is also costly to provide. MPEG was originally designed with storage applications in mind, and as a consequence is relatively intolerant of errors. It is this error sensitivity, coupled with the need for synchronous reconstruction at the decoder, that creates difficulties when we attempt to use MPEG for wireless access.

To quantify these effects, we consider how a heterogeneous network with wireless access may attempt to support MPEG. MPEG bit error rate (BER) requirements are in the range of 10^{-12} to 10^{-9} [12] [11]. Since the BER for a mobile wireless channel is many orders of magnitude worse, we would need to apply forward error correction (FEC), an automatic-repeat-request (ARQ) protocol, or a combination of both.

Let us address pure FEC first. We consider the Rayleigh fading channel, a widely accepted model for mobile radio [5] [6]. From [5], a sophisticated concatenated convolutional and Hadamard code requires a bandwidth expansion of approximately twice the number of orders of magnitude decrease in BER. The BER for a mobile wireless channel is typically 10^{-2} to 10^{-3} [13]. As a result:

- to lower the BER of a mobile wireless channel from 10^{-3} to 10^{-12} , the MPEG BER requirement, requires a bandwidth expansion factor of 18. If MPEG has a compression ratio of 100:1, then after adding channel coding redundancy, the effective compression ratio drops to a modest 5:1;
- the FEC assumes soft Viterbi decoding, which is complex to implement in hardware;
- a complex channel decoder at the mobile receiver will adversely impact the power consumption of the portable unit.

We can try to ameliorate the situation by applying a combination of ARQ and FEC. We consider the most bandwidth efficient of all ARQ protocols, namely Selective Repeat (SR). Further, let us disregard its buffering requirements for now and suppose that ideal SR is possible. Since MPEG decoding requires synchronous reconstruction, if a packet is delayed beyond the tolerable bound for interactivity, the packet becomes useless and is lost. Hence, a packet is useful only if received within B transmissions, where

$$B \times \text{roundtrip delay} \leq \text{delay bound} \quad (10.1)$$

Let X be the number of transmissions of a packet, and P_e be the probability of packet error. From [10], for ideal SR, we can express the probability that a packet is successfully received on the i^{th} transmission as,

$$\Pr[X = i] = (1 - P_e) P_e^{i-1}. \quad (10.2)$$

Summing the geometric distribution, we can determine the probability that a packet is *stale*, that is, the packet is lost because its total delay exceeds the delay bound for interactivity.

$$Pr[X > B] = P_e^B. \quad (10.3)$$

If we assume independent bit errors,

$$P_e = 1 - (1 - BER)^N, \quad (10.4)$$

where N is the number of bits in a packet. Combining (10.3) and (10.4), we have the result that to meet the MPEG bit-error rate requirement, the channel bit-error rate must satisfy

$$BER \leq 1 - \left(1 - BER_{MPEG}^{1/B}\right)^{1/N}. \quad (10.5)$$

If the BER is small, we can approximate (10.5) by

$$BER \leq \frac{BER_{MPEG}^{1/B}}{N}. \quad (10.6)$$

For $BER_{MPEG} = 10^{-12}$, a round-trip delay of 30 ms, an interactive delay bound of 100 ms, and a packet size of 188 bytes (the size of an MPEG2 transport packet [14]), the ideal SR protocol would require a BER of approximately 10^{-7} or lower. The BER requirement would have to be achieved by applying FEC to the unconditioned channel. This hybrid ARQ/FEC scheme has the following implications:

- By use of ARQ, we have lightened the burden on FEC considerably. Nonetheless, lowering the channel BER from 10^{-3} to 10^{-7} still requires a bandwidth expansion factor of 8, which means the effective compression ratio of MPEG is approximately 12:1, almost an order of magnitude lower!
- ARQ is ineffective if the round-trip delay exceeds the interactive delay bound. This may happen, for example, if the network route contains a satellite link.

- ARQ incurs additional bandwidth overhead for retransmissions as well as for error detection.
- Finally, the system becomes even more complex to implement. In particular, the selective repeat protocol requires extensive buffering at the receiver, whereas, other protocols that do not require buffering at the receiver (e.g., Go-Back-N) are less bandwidth efficient.

In summary, the analysis above suggest that to ameliorate MPEG's intolerance to errors, we must use a more complex receiver in the portable unit and severely compromise MPEG's compression ratio. We believe added complexity is a major problem for portable, personal communications systems (PCS) since more complexity usually leads to more power consumption, and PCS must operate under low-power constraints. Further, an MPEG decoder is also a fairly complex device that can consume substantial power. In fact, to date we know of no low-power implementation of an MPEG decoder. We point out these difficulties with MPEG not to discount MPEG, but to motivate the need for video coding algorithms that are better coordinated with the transport of the network, particular for wireless access. We explore this concept further below.

3 SYSTEM LEVEL CONSIDERATIONS

To realize techniques such as end-to-end coding and asynchronous reconstruction of video we must have certain functionality provided by the network. For example, asynchronous reconstruction of video requires the video coder to identify information within the video stream that can be delayed from its original frame to later frames. However, for this technique to be useful, the network must have the intelligence to exploit the different delay requirements of the different information elements. That is, how does the network recognize which information elements have relaxed delay requirements? Furthermore, once the network has identified these elements, how does it use this information to reduce perceptual delay and increase traffic capacity? These system-level issues and more are discussed in detail in [1], which proposes a basic architecture for heterogeneous CM networks. In this section, we briefly summarize the salient features and concepts of [1] that are relevant to the current discussion.

3.1 Syntactical constraints

In the realization of CM services, as distinct from other services, there are three critical signal processing technologies: *compression*, *forward error-correction coding (FEC)*, and *encryption*. These signal processing technologies modify or hide basic syntactical and semantic components of a bit stream. Subjective quality is important for CM services, and is affected by both signal processing and transmission impairments. Signal-processing considerations should thus play a major role in decisions about network architecture.

Figure 1 illustrates some fundamental syntactical constraints that we should keep in mind while designing a network architecture for CM services:

- Compression must precede encryption and decryption must precede decompression. Encryption would hide basic statistical characteristics of an uncompressed audio or video signal, such as spatial and temporal correlations, that are heavily exploited by compression algorithms.
- Compression must precede FEC and decompression must follow FEC, since there is no point to "correcting" the benign and desired changes in a bit stream due to compression and decompression.
- The relationship between encryption and error-correction coding is more complicated. We divide error-correction coding into two classes: *binary* (such as algebraic and convolutional coding) that transform a bit stream into another bit stream, and *signal space* (such as trellis and lattice coding) that are integrated into a modulation system and involve Euclidean-space manipulations [6]. There are hybrid cases, such as convolutional coding with soft decoding, which we lump into the signal-space category. Since encryption, like binary coding, transforms one bit stream into another, it can precede or follow binary error-correction coding. However, since a signal-space code generates an output in the real-number field, it cannot precede encryption and signal space decoding cannot follow decryption. The purpose of binary coding before encryption is to attempt to correct post-decryption errors. The purpose of binary or signal-space coding after encryption is to prevent errors in the transport of the encrypted bit stream, which will indirectly prevent post-decryption errors.

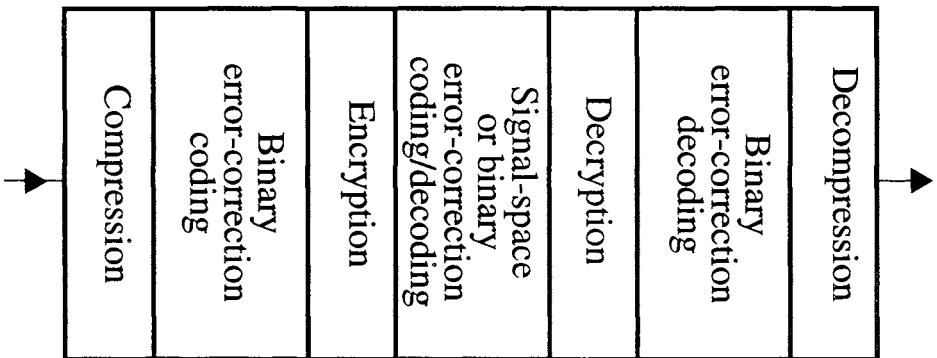


Figure 1 Syntactical Constraints on Signal Processing Functions

3.2 Edge vs. link architecture

Another crucial decision is the partitioning and mapping of the signal processing functions. For example, do we perform compression, encryption, and forward error-correction on a per-link or an end-to-end basis? There is a strong tendency to perform compression on a per-link basis, placing transcoders (converters from one compression standard to another) at the boundaries of the subnetworks. For example, in the telephone network, in a call from a wired to a digital cellular telephone, one voice coding technique (8 kHz sampled PCM) is used on the wired network and another (VSELP in the case of the North American IS-54 standard) is used on the digital cellular subnet [7]. This is for valid and important technical reasons; namely, the desire for spectral efficiency on the digital cellular subnet, resulting in more aggressive compression (traded off against implementation cost and reduced subjective quality) and the need for joint source/channel coding. An unfortunate side effect is that introducing a new or improved service, such as wideband voice, can only be done by the service providers, not by the user. Another side effect is that privacy cannot be guaranteed, since transcoding requires decryption and hence the user must provide encryption keys to the service provider (or leave encryption entirely to the service provider). A third side effect is a substantial accumulation of added subjective impairment and delay. For example, the digital cellular transcoder introduces on the order of 80 milliseconds of one-way delay, so that two digital cellular phones experience a round-trip delay on the order of 320 milliseconds due to the tandem transcoders. The extension of this approach to more complicated heterogeneous networking scenarios could easily result in unacceptably large (multi-second) delays, which is an even more severe problem when video and audio are combined.

In contrast, performing some of these signal processing functions (if not all) on an end-to-end basis offers some significant advantages:

- *Privacy and security.* The link architecture is incapable of providing privacy by end-to-end encryption under user control, since an encrypted signal cannot be transcoded. The best that can done is encryption on a link basis by the service provider(s).
- *Open to change.* The edge architecture is open to substitution of different transport service layers at the network edge (user terminal or access point). This leads to an economically viable method to upgrade transport services over time, as well as introduce new ones.
- *Performance.* The link architecture suffers from the accumulation of delay and subjective impairment through tandem compressions and decompression of the CM signal. As mentioned above, this problem has already become serious in digital cellular telephony. In more complicated heterogeneous scenarios, delay could become unacceptable for delay-sensitive interactive applications.
- *Mobility.* The link architecture embeds considerably more state within the network associated with the realization of a CM service, creating additional requirements for migration of state when terminals are mobile.

Thus, we believe the edge architecture is superior to the link architecture and it should be adopted for the future.

3.3 Substream abstraction of transport

One of the key features of [1] that embodies many of the ideas discussed above is its *substream abstraction* for the transport network. Figure 2 illustrates this abstraction.

For a particular audio or video *stream*, the transport provisions *substreams* (through a link-layer protocol). Each substream has a quality of service (QOS) specification of loss, corruption, and delay characteristics. The QOS of the substreams are different, as negotiated at session establishment. In addition, the collection of substreams comprising the stream has a *joint* specification of rate parameters (substreams have correlated rates since they originate from the same CM source). The substreams provide a mechanism by which a CM

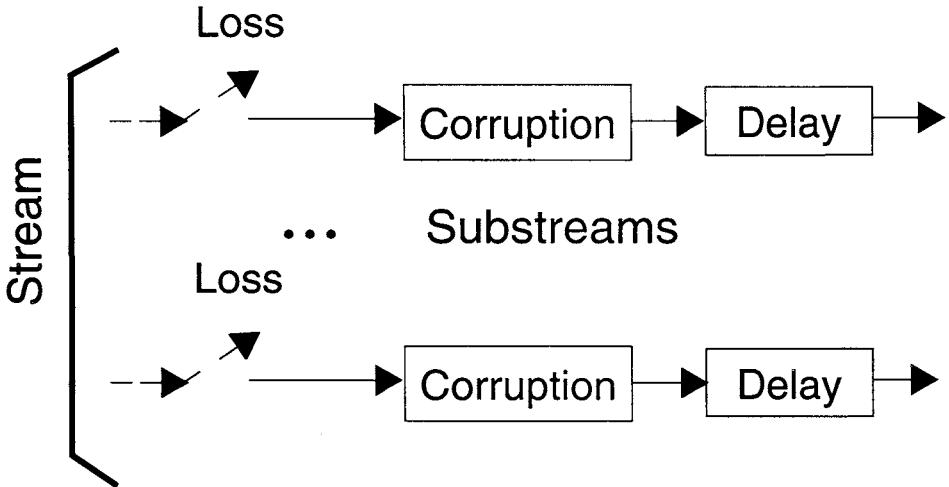


Figure 2 Substream Abstraction of Transport

service can specify different QOS characteristics for different information elements. This in turn becomes the mechanism by which the benefits of *joint source/channel coding* are achieved without a close coupling of source and channel coding design. Since substreams can be transported intact across heterogeneous subnetwork segments, joint source/channel coding on downstream links does not require transcoders. Further, encryption does not interfere with joint source/channel coding, as long as the substreams are independently encrypted.

In the absence of substreams, the delay characteristics of all data in a CM stream is the same. With substreams, data can be segregated into parts with different delay characteristics, and the network can exploit the relaxed delay characteristics of some substreams to achieve higher overall traffic capacity (this is one manifestation of joint source/channel coding, albeit one not previously emphasized). From the perspective of the CM coder, there is control over which data arrives earlier than the worst-case delay, and thus this low-delay data can be exploited.

While constructs similar to substreams (typically called *flows*) have been proposed for future Internet protocols, we argue that substreams are crucial to obtaining high traffic efficiency in networks with wireless access links. Since the wireless access link is where the greatest gains in capacity can be realized in a heterogeneous network, we will examine it more closely.

4 QOS AND TRAFFIC CAPACITY

The key components of QOS are bandwidth, delay and reliability. Let us consider reliability first. For packet networks, we may distinguish between corruption (errors in the packet payload) versus loss (errors in the packet header, causing the entire packet to be lost). On a wireless channel, the reliability of a substream is proportional to the power at which it is transmitted over the air: the greater the power of one's own signal relative to the aggregate power of other users, the lower the probability of error. Moreover, the traffic capacity of wireless multiple access schemes such as code-division multiple access (CDMA), and wireless cellular systems in general, are not just bandlimited, but interference-limited as well [4]. It is therefore beneficial to transmit the minimum power necessary to support a given reliability for a substream, as this creates the least interference to other users. In the absence of substreams, an entire video or audio stream would have to be transmitted at the reliability level needed by its most error-sensitive component. This squanders power and increases interference. With the application of substreams, power allocation can be fine-tuned, thereby maximizing the interference-limited capacity of a wireless subnet [4].

We have seen that providing variable reliability via the substream abstraction can lead to network capacity gain. A natural question is, can we gain an additional increase in traffic capacity by making the different delay requirements among substreams visible to the network? That is, will the network capacity be greater if each stream is specified by a single (tightest) delay requirement, or if each substream can specify its own delay requirement? We address this issue by considering the impact of variable substream delays in two ways: on the capacity of the network to accommodate delay contracts, and on the capacity of a wireless access link limited by interference.

Consider a work conserving queue¹ with inputs from streams for the following two cases:

- the server knows that each stream is further partitioned into substreams. Hence, the server knows the average delay requirement for each substream and achieves just those delays;

¹A queue server is work-conserving if it never idles when there are packets waiting to be transmitted; a simple example is first-come-first-served. Work conserving queuing disciplines are widely used because they are bandwidth efficient — an important consideration for wireless links.

- the server just knows a single (tightest) average delay requirement for each stream, and achieves that delay for each stream.

Now suppose in each case the system is operating at capacity, in the sense that the bounds on average delay are just being met. What is the maximum utilization in each case?

For *any* work-conserving queuing discipline and *any* set of substream arrival processes, the GI/G/1 conservation law states [2]

$$\sum_{\text{stream } i} \sum_{\text{substream } j} \rho_{i,j} W_{i,j} = \text{constant}, \quad (10.7)$$

where $\rho_{i,j}$ is the offered load (or utilization factor) and $W_{i,j}$ is the average waiting time of substream j , stream i .

Consider the case where we do not partition stream m into substreams. The overall delay requirement of stream m then equals that of its most delay-sensitive component. It follows that the average delay of stream m satisfies

$$\tilde{W}_m = \min_j \{W_{m,j}\}. \quad (10.8)$$

The utilization factor of the stream is simply $\rho_m = \sum_{\text{substream } j} \rho_{m,j}$, so we have

$$\begin{aligned} \rho_m \tilde{W}_m &= \left(\sum_{\text{substream } j} \rho_{m,j} \right) \times \min_n \{W_{m,n}\} \\ &\leq \sum_{\text{substream } j} \rho_{m,j} W_{m,j}. \end{aligned} \quad (10.9)$$

By specifying any stream m with a single (tightest) delay requirement rather than by its individual substream delay requirements, it follows from (10.7) and (10.9) that the delays experienced by some of the other substreams must necessarily increase (the only case where none of the other substreams' delays increase is if all the substreams of stream m have identical delay requirements). Since these substreams were already operating at the limits of their

delay bounds, any further increase in delay would violate their delay requirements. These substreams would have to be removed, decreasing the traffic capacity of the system. Hence, introducing the substream abstraction can promote the allocation of delay at fine granularity, with a corresponding increase in the *delay-limited* capacity of the system. This result is completely general and holds for any substream arrival process, the only assumption being that the queuing discipline is work-conserving.

To obtain a concrete figure on the capacity gain, we would need to characterize the arrival processes further. As a simple example, consider a stream with Poisson arrivals which is segregated into three substreams, as shown in Table 1.

π_i is the fraction of arrivals in the stream that belong to substream i , so $\sum \pi_i = 1$. We order the substreams according to their delay requirements, so that $W_1 \leq W_2 \leq W_3$.

Now consider statistically multiplexing independent Poisson streams with the *same* substream profile as in Table 1, but possibly different rates. If the partitioning of each stream process into substreams is random, then the substreams are also Poisson. The aggregation of all streams is then Poisson, and we have an M/G/1 system.

We wish to compare the maximum load that the system can accommodate (while meeting all stream delay requirements) when the substream structure is hidden, versus when the structure is made visible to the server. The server with no information about the substream structure would try to meet a single (tightest) delay requirement W_1 for each stream; the server which knows about substreams would just satisfy the individual delay requirement of each

Table 1 Sample substream profile for a stream

	fraction of stream arrivals	required average delay
substream #1	π_1	W_1
substream #2	π_2	W_2
substream #3	π_3	W_3

substream. Let $\rho_{no_substreams}$, $\rho_{substreams}$ be the respective system loads. We are interested in the *capacity gain* G , defined as

$$G = \frac{\rho_{substreams}}{\rho_{no_substreams}}. \quad (10.10)$$

It can be shown [9] that

$$G = \frac{\pi}{1 + (\pi - 1) \rho_{no_substreams}}, \quad (10.11)$$

where $\pi \equiv \frac{W_1}{W_1} \pi_1 + \frac{W_2}{W_1} \pi_2 + \frac{W_3}{W_1} \pi_3$.

Intuitively, π can be interpreted as a measure of the spread in the various delay requirements of a stream's components. For example, we note that if $\pi_1 = 1$, corresponding to the case where each stream is comprised entirely of substream one, then π and the capacity gain G is unity. This is as expected, since a system with and without substreams would then be identical. The gain also approaches unity as the load $\rho_{no_substreams}$ approaches one, because from rate considerations the load cannot exceed unity. At all other operating points, the gain in capacity is strictly greater than unity, as shown in Figure 3.

Finally, let us look at a specific numerical example. Let the substream delay requirements be 30, 60 and 90 ms respectively, and let $[\pi_1, \pi_2, \pi_3] = [0.2, 0.35, 0.45]$. If the delay-limited capacity of a substream-less system is $\rho_{no_substreams} = 0.1$, then the load capacity of a system utilizing substreams is $\rho_{substreams} = 0.2$ — a twofold gain in capacity.

We can exploit the variable substream delay concept even further by considering its impact on the interference-limited capacity of wireless multiple-access links. On a wireless link, a substream's reliability requirement is mapped onto a desired signal-to-interference noise ratio (SNR). The capacity of a wireless multiple-access link is interference-limited in the sense that in order for the substream reliability requirements to be satisfied, the following condition must hold at any time [3]:

$$\sum_{all\ users\ k} f(SNR_{any\ substream\ of\ user\ k}) < 1, \quad (10.12)$$

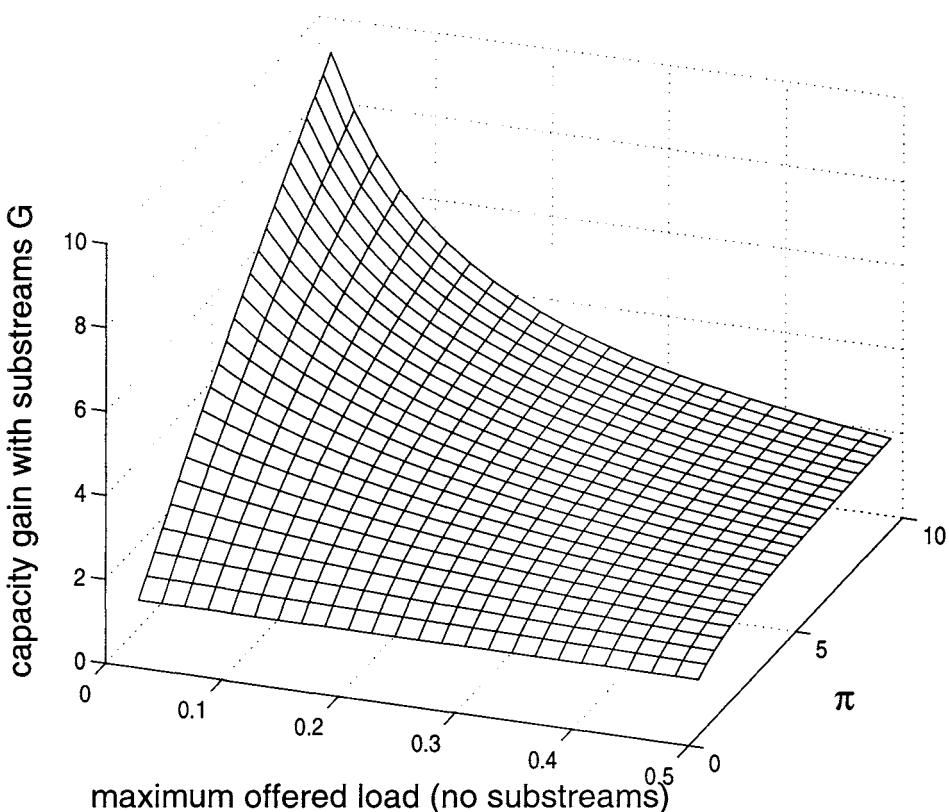


Figure 3 Traffic Capacity Gain Using Substreams

where $f(x)$ is a monotonic function in x . Specifically, (10.12) must hold when all users simultaneously transmit their maximum SNR substreams. A call admission criterion satisfying this worst case may under-utilize system capacity. However, if substreams with variable delay requirements are available, then during times of high SNR traffic loads we can delay the transmission of substreams with relaxed delay requirements until traffic conditions are lighter, thereby increasing capacity. In a similar vein, there may be occasions when (10.12) is satisfied, but the channel conditions are so poor that the level of transmit power dictated by the power allocation algorithm would physically saturate the transmitter. Again, if a given packet has more flexibility in its transmission time, we can wait until channel conditions are more benign before

transmitting, thereby providing an additional mechanism for traffic smoothing and capacity increase.

5 VIDEO CODING FOR A SUBSTREAM TRANSPORT

In Section 3 we discussed a substream abstraction of the transport. Further, in Section 4, we discussed how such a substream transport can be used to provide QOS and increase traffic capacity on a wireless subnet. In this section we discuss a method of coding video for transport via substreams.

5.1 Overview of asynchronous video coding

Video coding for substream transport requires the partitioning of the video stream into substreams of compressed video with different QOS characteristics (i.e., delay, loss, and corruption). Since substreams have different delay characteristics, they arrive at the receiver asynchronously. It is possible to resynchronize substreams at the receiving terminal, but to do so will result in transport delay (including re-synchronizing delay) that is characteristic of the highest-delay substream. We choose a different approach, in which the video is asynchronously reconstructed at the receiving terminal. Our goal is to achieve a perceptual delay that is representative of the *lowest-delay substream*, rather than the highest-delay substream. We define *asynchronous video* (ASV) as video coding for substream transport, where we decode the video presentation for display without re-synchronization of the substreams [8].

Figure 4 illustrates an abstracted view of ASV coding for a substream transport using three substreams. At the source, the ASV coder partitions a video stream into substreams by segregated the semantic components of the stream into three classes: *high motion*, *low motion*, and *no (or very low) motion*. For example, the semantic components might be 8x8 blocks of pixels, where each block represents a particular spatial region in one frame of video. Therefore, the high-motion substream will carry blocks that have a high level of motion, the low-motion substream will carry blocks that have a low level of motion, etc., where motion is determined by some metric (see below). At the sink, the ASV decoder asynchronously reconstructs the video stream from its substreams. Thus, continuing with our example, video blocks that originate in one frame at the source might be reconstructed in a later frame at the sink

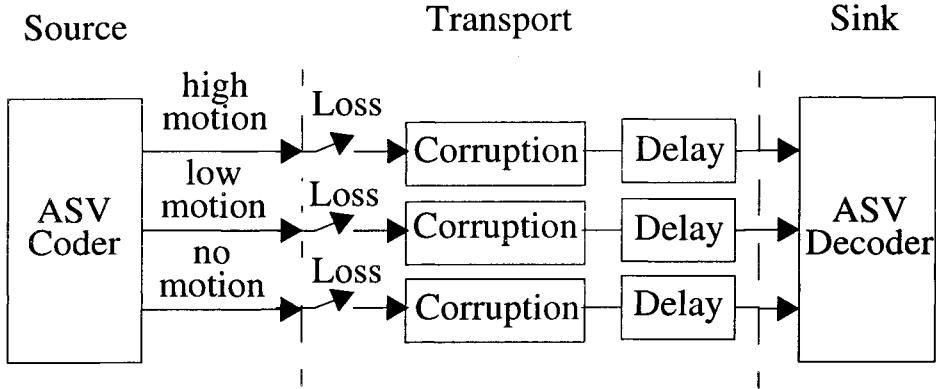


Figure 4 Substream Abstraction for ASV Coding

(see Figure 5). In Section 5.2, we will explain how this can be done without introducing undesirable artifacts.

Within this framework, we can make the following intuitive observations:

- The perceptual delay will be dominated by the transport delay of the high-motion substream. For example, in video conferencing the high-motion areas will typically be the person's head, lips, etc., thus, the audio delay for lip synchronization (or the perceptual delay) will be approximately equal to the delay of the high-motion substream.
- The high-motion substream is more tolerant of loss and corruption than the other substreams since high motion can subjectively mask some of the artifacts resulting from channel and quantization errors. Thus, this substream can have lower resolution and relaxed loss and/or corruption requirements relative to the other substreams.
- The low-motion substreams are less tolerant of loss and corruption in both coding and in transport, but more tolerant of delay. Thus, these substreams can have higher resolution and relaxed delay requirements relative to the high-motion substream.

Nature is kind to us, since blocks with tighter delay requirements have relaxed loss and/or corruption requirements, and *vice versa*. This is precisely the typical tradeoff between delay and loss in subnetworks operating with constant

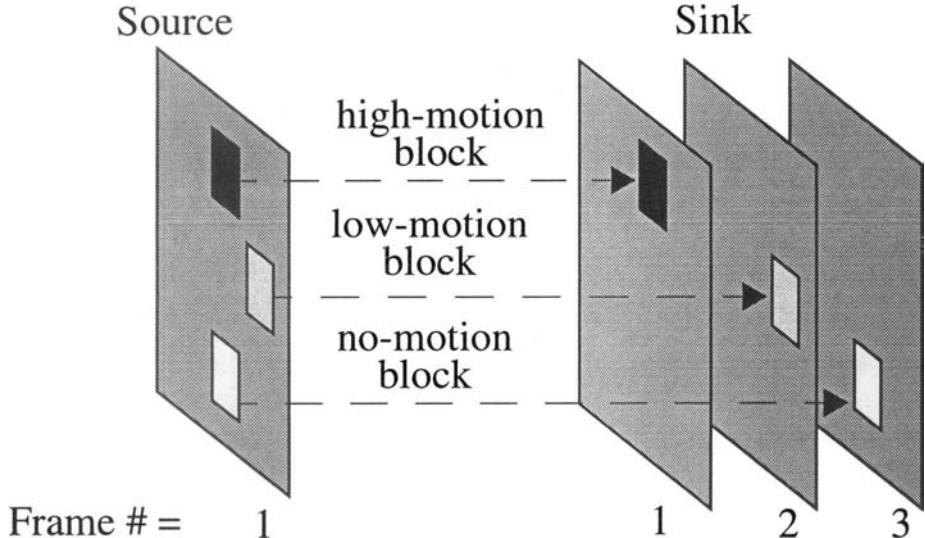


Figure 5 Example of Asynchronous Reconstruction

resource utilization. That is, techniques that improve reliability (e.g., FEC) also introduce extra delay. Therefore, this *motion-adaptive* partitioning is a natural one; that is, the ASV coder's substream abstraction maps nicely into the transport's substream abstraction (see Figure 6).

To reiterate, with this partitioning of video streams into substreams, we expect intuitively that the low-motion substreams can be delayed relative to the high-motion substreams. Further, we expect that the perceptual delay will be dominated by the high-motion blocks, which are transported with the lowest delay. Thus, the perceptual delay will be dominated by the lowest-delay substream, rather than the worst-case transport delay. To the extent that the other substreams' delay requirements can be relaxed, the network traffic capacity can be increased. We believe that this opportunity has the greatest significance on wireless access links, although (as noted before) we propose to use this video coding technique transparently across the entire edge-to-edge connection to avoid extra delay (and other problems [1]) introduced by transcoders.

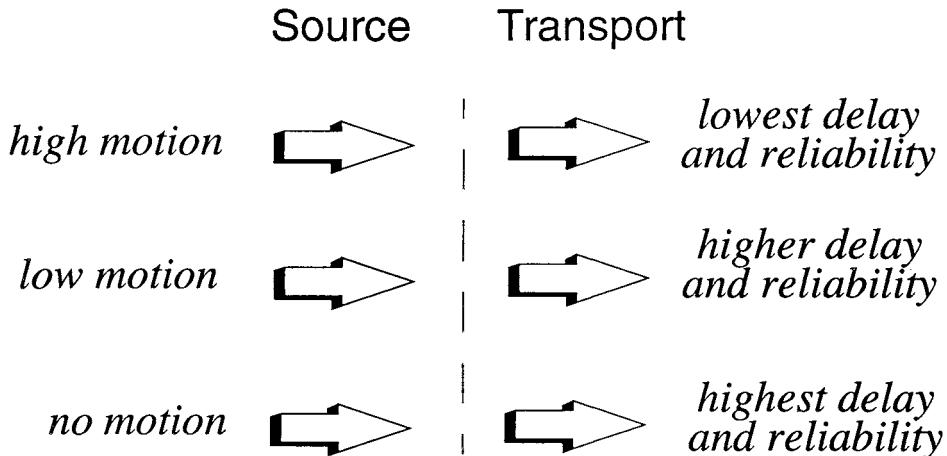


Figure 6 Mapping of Substreams From Source to Transport

5.2 ASV codec design

In this section we give a general description of our ASV codec that serves as a framework for future designs. Here, we emphasize the substream partitioning and asynchronous reconstruction of video, rather than the specific signal processing of each component. (See [8] for details on one such realization.) We discuss results of an actual software implementation in the next section.

Figure 7 shows a block diagram of the coder design. In the top forward-path, frames are first processed through a *transformation* component, which transforms the image into a suitable domain (e.g., frequency domain via subband coding) and representation (i.e., transformed-image blocks). The *substream coder* then performs quantization and compression of the transformed blocks to provide layered resolutions of compressed video blocks and, most importantly, partitions the video stream into substreams. The coder partitions the video stream into substreams according to a bank of motion-driven, *finite-state machines* (FSM), one for each spatial block-region of the image (see Figure 8). Motion detection is performed in the bottom forward-path, where each motion estimate is quantized into one of three motion levels: *No*, *Low*, or *High*. We use the average, squared-difference in pixel values between the current and previous blocks as our metric for estimating motion and a dual threshold to quantize each motion estimate into one of the three levels.

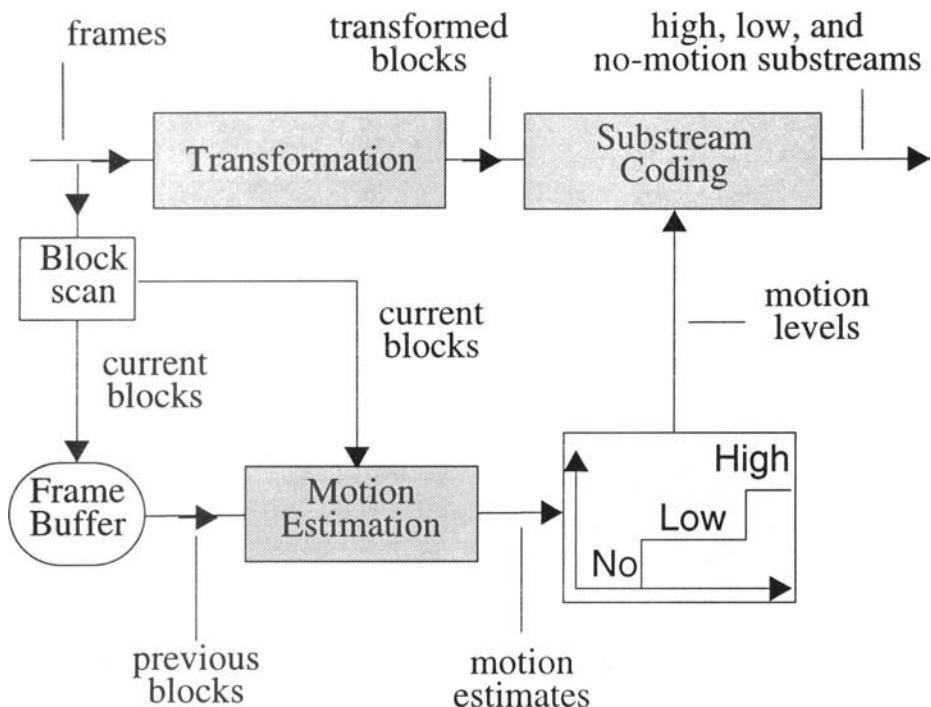


Figure 7 Block Diagram of ASV Coder

Figure 8 illustrates the substream-coder FSM for a particular block-region. The FSM updates its state according to the level of motion detected for that block region. There are three states: *HIGH*, *LOW*, and *NO* that correspond to the three source substreams (see Figure 6) and one state called *IDLE*. In particular, the FSM makes a transition to the *HIGH* state whenever a high level of motion is detected and the corresponding block is transported on the lowest-delay substream. For low levels of motion, the FSM makes a transition into the *LOW* state and the corresponding block is transported on the higher-delay substream. When virtually no motion is detected, then the FSM makes a transition to the *NO* state and the block is transported on the highest-delay substream. If there is still no motion detected for this block at the next frame, then the FSM makes a transition to the *IDLE* state, where no data is transmitted. The FSM remains in the *IDLE* state until motion is detected again.

For each block, a message is sent to the decoder in the form of a packet header that contains the following information:

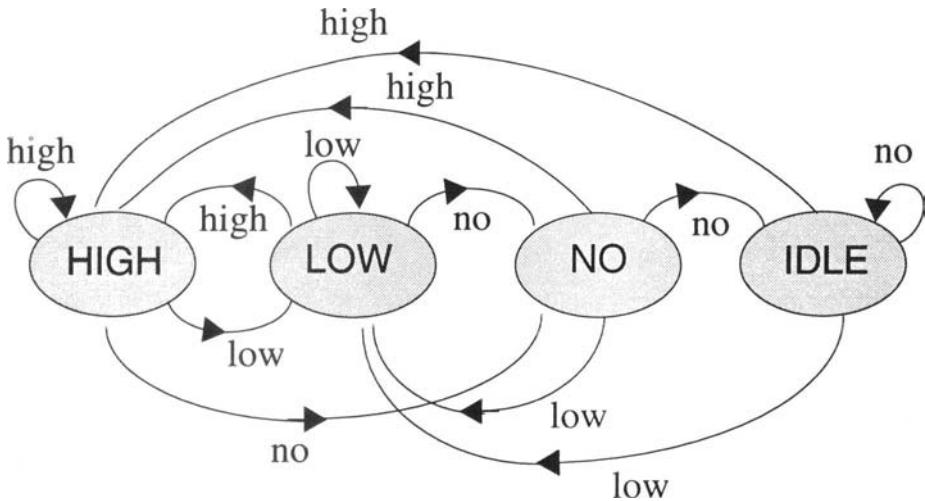


Figure 8 FSM of Substream Coder

- *Temporal locator*, which specifies a *frame number*.
- *Spatial locator*, which specifies the X-Y coordinates of the block.
- *Resolution identifier*, which specifies what layered resolution to decode.
- *Packet size*, which specifies the size of the packet (including header).

Each video packet is variable length with a fixed-length header, thus the payload is variable length. We choose a variable-length payload because ASV coding is variable-rate video coding, which is realized (in part) by adjusting the resolution[s] per substream. Note, the video packet header (as seen by the decoder) contains no information identifying which substream transported the packet. Thus, the ASV decoder maintains no state information about substreams. Below, we discuss how this leads to a simplified decoder design.

Figure 9 shows a block diagram of the ASV decoder. The substream-decoding component asynchronously reconstructs the video stream from the incoming video packets by maintaining an *ordering constraint*, which is expressed by,

$$P < N \leq (P + 8), \quad \text{if } 0 \leq P \leq 7 \\ (P - 8) \leq (N + 7) \bmod 16 < P, \quad \text{if } 8 \leq P \leq 15, \quad (10.13)$$

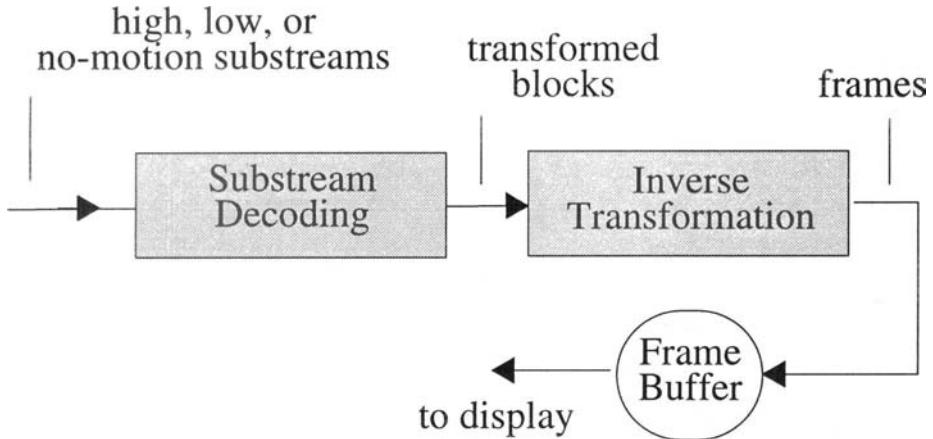


Figure 9 Block Diagram of ASV Decoder

where P is the previous frame number, N is the current frame number, and $N, P \in \{0, 1, \dots, 15\}$. That is, frame numbers are assigned using modulo 16 numbering ($0, 1, \dots, 15, 0, 1, \dots, 15, 0, 1, \dots$). If N does not satisfy (10.13), then the current block is considered to be *stale* and is discarded, otherwise, the current block is considered to be *new* and is displayed. For example, if $P = 4$, then incoming blocks with frame numbers $N \in \{0, 1, 2, 3, 12, 13, 14, 15\}$ are considered stale. Essentially, (10.13) helps the decoder identify blocks that have experienced excessive delay and are no longer useful, but are still delivered by the network because their *time-to-live* in the network did not expire. While we do not attempt to maintain the exact temporal relationship of the blocks, nor do we re-synchronize the substreams, it is necessary to impose this ordering constraint to prevent undesirable artifacts. Referring to Figure 10, if a no-motion block is generated in frame 1 at the source, but doesn't show up at the sink until frame 3 (because of its relaxed delay requirements) after a high-motion block that was generated at the source in frame 2, then the no-motion block is stale and should be discarded. The inverse transformation component produces the actual image representation. In summary, the substream decoder performs a simple calculation to ensure that only the most recent image data for each block region is displayed; stale data is always discarded.

We have not specified the way in which the layered resolution is achieved. There are many ways this could be done, but in the present realizations we have

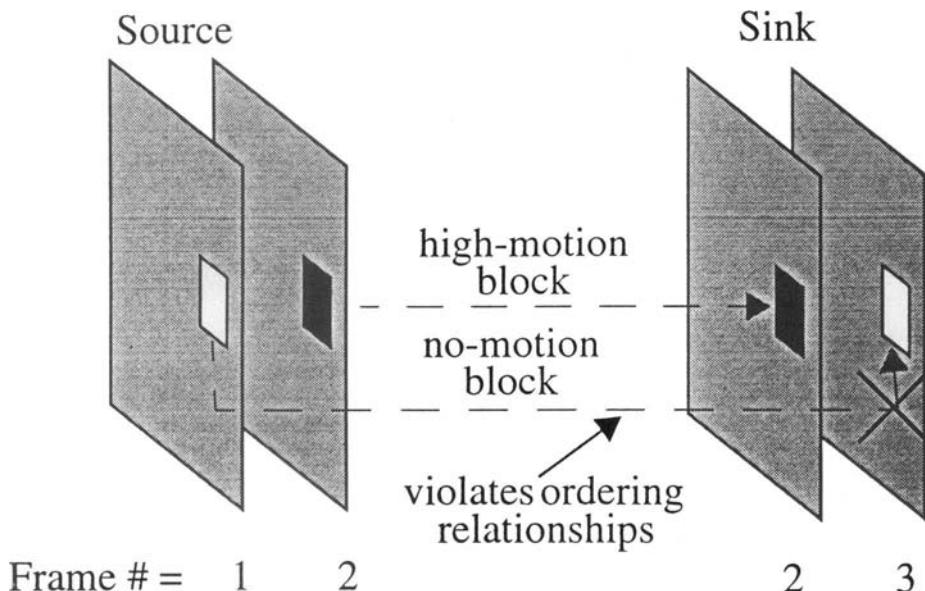


Figure 10 Example of Incorrect Asynchronous Reconstruction

chosen a simplistic subband decomposition. Further, our current ASV coder does not utilize other known compression techniques like motion compensation, but it may be possible to introduce these concepts later. The video coding simulations reported here are very simple, designed to illustrate the perceptual delay benefits of ASV.

6 RESULTS

In this section, we investigate the effects perceptual delay, delay-bound requirements, and channel errors have on the subjective quality of ASV by discussing results of three simulations. The ASV codec used for each simulation was based on subband decomposition with scalar quantization of subband coefficients [8].

6.1 Perceptual delay

To measure the perceptual delay, we used a 15 frames per second *talking-heads sequence*, which we refer to as SEQ. This sequence has significant high-motion content (i.e., head and lip movements), low-motion content (i.e., a slowly panning overlay), and stationary content (i.e., a motionless background).

We simulated ASV coding of SEQ with different delay requirements for each substream. For this experiment, we chose the delay requirement for the high-motion substream to always be less than the 66 ms frame time for SEQ; that is, we require that all high-motion blocks from the same frame arrive at the receiver within 66 ms. For the other substreams, we assigned delay requirements greater than 66 ms. We then played back the processed data with its corresponding audio to observe *lip sync*.

From our observations, we could not differentiate the lip sync of the processed video from the original, which suggest that the perceptual delay for SEQ was indeed dominated by the high-motion substream. We were able to verify this by keeping the delay of the high-motion substream constant (and always greater than 66 ms) and then varying the delay requirements for the low and no-motion substreams. Under these constraints, lip sync was always disrupted regardless of the delay requirements assigned to the medium and fine substreams; that is, even with a zero delay requirement on the low and no-motion substreams, lip sync was disrupted.

In summary, this empirical evidence suggests that we can indeed relax the delay requirements on the low and no-motion substreams without introducing objectionable artifacts, but there are bounds on how much we can relax these requirements. The next section presents some empirical results to quantify these bounds.

6.2 Delay bounds

The effects of relaxed delay bounds on substreams, besides the high-motion substream, offer the transport considerable flexibility in meeting other needs (e.g., error protection) of these substreams. However, by relaxing the delay bounds, we might introduce visual artifacts in the video resulting from the temporal displacement of high, low, and no-motion blocks relative to their original frame. The question is whether subjective quality degradation can be

held in check while the transport delay bounds for certain substreams varies as much as multiple frame intervals.

To measure how relaxed the delay bounds can be on the low and no-motion substreams, we used a 24 frames per second action sequence, which we refer to as MOT. This sequence also had significant high-motion, low-motion, and stationary content.

For this experiment, we kept the delay requirement for the high-motion substream below the 42 ms frame time for MOT and varied the delay requirements for the other substreams until the video quality became objectionable. From our observations, we concluded that the low-motion substream could tolerate a delay up to one frame time and the no-motion substream could tolerate a delay up to three frame times. In other words, low-motion video blocks would not arrive in time for the next output frame but for the one after and no-motion video blocks could be delayed by two additional frames.

In summary, this empirical evidence suggests that it is possible to relax the delay requirements on the low and no-motion substreams (to a lesser extent for low-motion substreams). In particular, video blocks being transported by the highest-delay (i.e., no-motion) substream can be delayed multiple frame times and, thus, lead to substantial gains in traffic capacity as outlined by the strategies presented in Section 4.

6.3 Error resilience and bit rate

It was also of interest to test the resilience of the video quality to error impairments. As discussed before the high-motion substream is more tolerant of errors than the other substreams. For the MOT sequence, we found that the high-motion substream could endure an error rate three to four orders of magnitude higher than the no-motion substream for roughly equivalent subjective impairment. In the absence of any form of error control, we found the video quality was good under error rates on the high-motion substream in the range of 10^{-4} to 10^{-3} , a fairly reasonable error probability to sustain on wireless subnets.

Figure 11 provides an example of the bandwidth gains resulting from our approach. For the MOT sequence displaying 320x240 frames of eight-bit gray-scale data at 24 frames per sec, the average bit rate produced by the coder was about 5 Mbps while its raw bit rate was 14.7 Mbps. For this simple re-

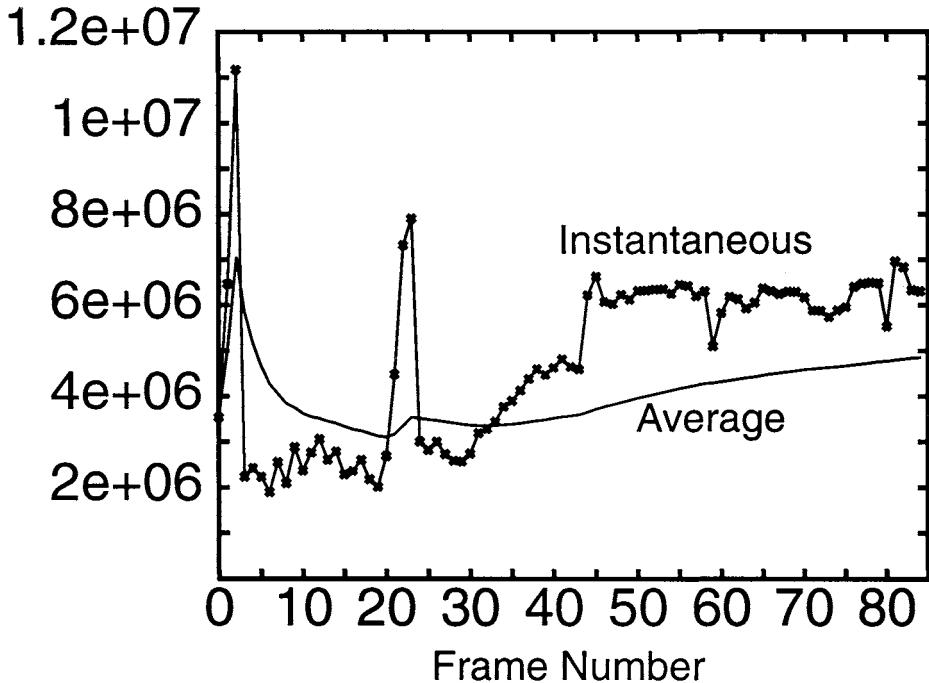


Figure 11 Bit Rate for MOT Sequence

alization of an ASV codec, the compression ratio was at best 3:1 for various test sequences. It should be noted that the work we have discussed here has not focused at all on attempts to merge ASV coding with popular compression techniques such as motion compensation or vector quantization, but doing so should result in significantly higher compression ratios. Nevertheless, our primary goal has been to reduce perceptual delay and increase traffic capacity, which is not necessarily the same as minimizing the bit rate.

7 CONCLUSIONS AND FUTURE WORK

We consider our work on ASV coding and decoding to be very preliminary. Since we could see no path from existing video compression standards like MPEG directly to an ASV version, we instead started from first principles. Thus, our preliminary ASV codec might be considered a form of simple conditional replenishment coding, as it achieves compression only through dis-

carding fine-resolution information in areas of high motion, and transporting low-motion information less often (or only once, where there is no motion at all). The novelty of our codec is in the temporal dimension, where we are moving low-motion and stationary, high-resolution information to later frames, as long as the resulting artifacts are not subjectively objectionable, and thereby gaining traffic capacity advantages.

In future work, we hope to demonstrate more sophisticated versions of ASV codecs, in particular, incorporating other known video compression techniques such as vector quantization (VQ) and motion compensation. One such realization using multiple VQ codebooks is currently being designed. Additional work is needed in verifying ASV video over more accurate transport models. Our efforts to date have focused on demonstrating the perceptual delay advantages of ASV, and the error models used are simplistic. Achieving the best combination of subjective quality and traffic capacity on wireless links will require considerable additional work.

Simultaneously, we are exploring network transport algorithms to realize some of the capacity gains promised by joint source-channel coding via the substream abstraction. Ideally, such an algorithm would need to incorporate and integrate many of the concepts we described in this paper: maximizing the interference-limited and delay-limited capacity through a *coordinated* allocation of reliability (power control) and delay (queuing disciplines). We are currently designing a real-time prototype of ASV transport by wireless CDMA in the context of the Infopad wireless multimedia computing project at Berkeley.

Acknowledgements

This research was supported by the Advanced Research Projects Agency, Tektronix, the University of California MICRO Program, and Asahi Semiconductor. In addition, special thanks goes to Melody Ivory and Jose Pino for helping us successfully convert our FrameMaker document into LaTex.

REFERENCES

- [1] Haskell, P., Messerschmitt, D.G., "A Signal Processing Perspective on Networking for Continuous-Media Services", submitted to IEEE/ACM Trans-

- actions on Networking, November 1994.
- [2] Kleinrock, L., Queuing Systems, Vol II, J. Wiley, 1976.
 - [3] Yun, L.C., Messerschmitt, D.G., "Variable quality of service in CDMA systems by statistical power control", Proc. IEEE International Comm. Conf., June 18-21, 1995, Seattle, WA, vol. 2, pp. 713-719.
 - [4] Gilhousen, K.S. et al, "On the capacity of a cellular CDMA system", IEEE Trans. Vehicular Tech., (40):2, pp. 303-312, May 1991.
 - [5] Proakis, J.G., Digital Communications, 3rd ed., McGraw-Hill, 1995.
 - [6] Lee, E.A., Messerschmitt, D.G., Digital Communication, Second Edition, Boston: Kluwer Academic Press, 1993.
 - [7] Natvig, J.E., Hansen, S., de Brito, J., "Speech processing in the pan-European digital mobile radio system", Proc. GLOBECOM, Dallas, TX, USA, Nov. 27-30, 1989.
 - [8] Lao, A.Y., Reason, J.M., Messerschmitt, D.G., "Asynchronous Video Coding for Wireless Transport", IEEE Workshop on Mobile Computing and Applications, December 1994.
 - [9] Yun, L.C., Transport for Multimedia on Wireless Networks, Ph.D. dissertation, in preparation.
 - [10] Lin, S., Costello, D.J., Jr., Miller, M.J., "Automatic-repeat-request error-control schemes", IEEE Communications Magazine, vol.22, no.12, pp.5-17, December 1994.
 - [11] Lei, S., "Forward error correction codes for MPEG2 over ATM", IEEE Trans. on Cir. and Sys. for Video Tech., vol. 4, no. 2, April 1994.
 - [12] Montreuil, L., "Performance of coded QPSK modulation for the delivery of MPEG2 stream compared to analog FM modulation", 1993.
 - [13] Steele, R., Mobile Radio Communications, 1st ed., IEEE Press, 1992.
 - [14] ISO/IEC/JTC1/SC29/WG11 Systems Committee, "MPEG-2 systems working draft", July 1993.

WIRELESS PUBLISHING: ISSUES AND SOLUTIONS

T. Imielinski and S. Viswanathan[†]

Rutgers University - New Brunswick, NJ

†Bell Communications Research - Morristown, NJ

ABSTRACT

Publishing¹ is a spontaneous and periodic broadcasting (or multicasting) of data on the wireless channel by the MSS to a specific group of clients. Since publishing is server initiated, querying published data does not require any uplink transmission. Thus, publishing is particularly suitable for asymmetric communication environments, which have very narrow uplink channel or no uplink channel at all. We describe data organization methods for publishing based on temporal as well as multicast addressing. We also show how the publishing mode can be efficiently mixed with the standard on-demand mode in an adaptive manner, based on the profile of user requests.

1 INTRODUCTION

Wireless technology is gaining popularity very rapidly. Many forecast a gigantic market with millions of mobile users carrying small, battery powered palmtops equipped with a wireless connection. Mobile users will be in constant need of information such as traffic information, local directories (white/yellow pages), weather information, local sales, stock quotes, sports scores, commodity prices, etc.

Figure 1 shows the structure of a hypothetical system that provides clients with information services. The wireless information servers called Mobile Ser-

¹Other terms which have also been used for wireless publishing include disks on air, caches on air, airdisks etc

vice Stations (*MSS*) will be equipped with wireless transmitters (denoted by tall antennas) capable of reaching thousands of users (denoted by small dark rectangles) residing in cells. Cell sizes as well as the channel characteristics (such as bandwidth) will vary widely. Small picocells may have a very small range covering only relatively few clients. Larger cells will measure tens of miles in diameter. Cells may also overlap. A small cell (based on wireless LAN, for example) may provide a high bit rate through one wireless interface, and be located within a larger, say, Cellular Digital Packet Data (CDPD) cell. A client can use dual interface cards to communicate with the MSSs of both cells.

Bit rates will vary from cell to cell. Wireless LAN will offer relatively high bit rates (Millions of bits per second) within a small distance range, while CDPD, private data networks (RAM, ARDIS) or satellite networks will provide much lower data rates with much wider distance scopes. Additionally, charges and tariffs will depend on the network provider and, usually also on the location of the user. The wireless LAN connectivity will be free (ignoring, the one time fee for network interface cards and driver software) while outdoor radio connection will be charged per connection time or per packet sent/received. While it is probably true that users will be able to communicate and use information resources from any place and anytime, the cost will substantially vary, forcing users to make choices about what to send/receive, where and when. For example, a user may delay transmitting a wireless fax until s/he is in the vicinity of a pocket of high wireless bandwidth (like wireless LAN) rather than pay a lot of money for slow outdoor radio link such as CDPD. In fact, specially designed information kiosks offering very high bandwidth in a very small range and having a low power connectivity, may be available².

Each MSS will provide full connectivity to the internet but it may decide on different local caching policies (for information which is frequently accessed in its cell) as well as different modes of information delivery.

The main objective of this paper is to discuss a mechanism of information delivery, called publishing, which we find particularly suitable for wireless environments.

This paper is organized as follows: Section 2 introduces the notion of the publishing mode. Sections 3 and 4 discuss the ways in which publishing can be efficiently achieved. In section 5 we consider publishing of data items with varying access frequencies. In section 6 other related information delivery modes

²The ongoing *Infostations* project at Rutgers University is investigating different architectures for providing such high bandwidth pockets of connectivity to facilitate expensive data communication such as faxing.

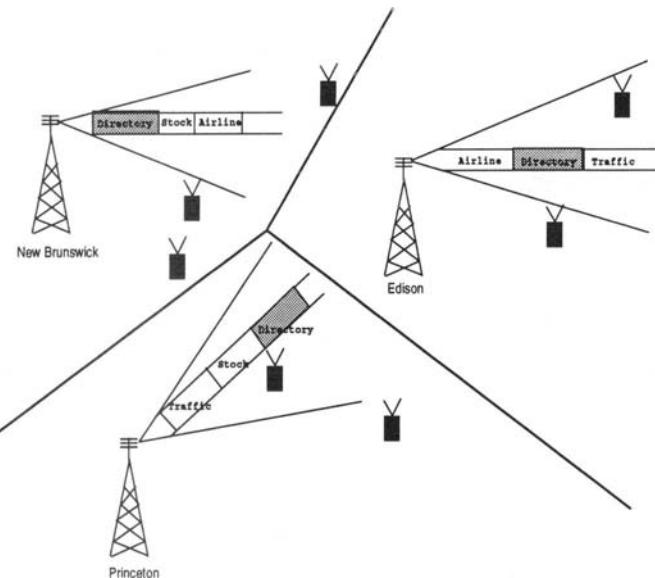


Figure 1 Wireless Information Servers

are discussed and finally section 7 presents the conclusion of the paper, with a brief sketch of the current implementation status of some of the ideas presented in this paper.

2 PUBLISHING MODE

Publishing is a spontaneous and periodic broadcasting (or multicasting) of data on the wireless channel by the MSS to a specific group of clients. Since publishing is server initiated, querying published data does not require any uplink³ transmission from the clients. It involves client initiated filtering of the published data stream which arrives on the downlink channel. The term publishing was first introduced in [4] and that paper also investigated broadcasting over a wireline network as an information dissemination mechanism. In the *Datacycle* project at Bellcore [3] and [6], a database circulates on a high bandwidth network (140 Mbps) and users query this data by filtering relevant information

³We distinguish between the uplink channel, from the client to the MSS and the downlink channel from the MSS to the client. In asymmetric wireless environment, the uplink channel is much more narrow than the downlink channel.

using a special massively parallel transceiver capable of filtering up to 200 million predicates a second. Scheduling issues of periodic broadcast are analyzed in [17]. Wireless publishing was considered by Gifford in [5], where a system called Boston Community Information System is described. In this system newspapers are broadcast over an FM channel and this information is downloaded by PCs equipped with radio receivers. There is a single communication channel (network) and power conservation plays little role since, the PCs (equipped with wireless modems) are connected to a continuous power supply.

The use of publishing as an energy efficient information dissemination mode was first described in [7] and then later in [8] and [9]. Other terms such as “disks on air” [13] and “cache on air” [9] have been subsequently used for wireless publishing as well. While the paper [13] concentrate on the publishing policies, we put more emphasis on the addressing issues for the broadcasted stream.

In this section we summarize most of the work in [9] and later discuss multicast based publishing solutions. In practice, the publishing mode will be rarely used alone. It will rather be mixed with the standard client-server mode (also called *on-demand* mode) of interaction where the client submits an uplink request to the MSS and downloads the necessary information (sent by the MSS). Only the most frequently requested data items, the so called **hot spots** (popular stock quotes, airline schedules a short time before arrival/departure, etc.) will be published. The standard on-demand mode will be used for items which are requested less often, since publishing these items would be a waste of bandwidth. However, even in the on-demand mode, it will make sense to batch requests for the same data item and send the data item once, rather than cater individually to each request.

In general, publishing will be used in *asymmetric* environments where the use of uplink link is either discouraged or simply impossible. The former applies when uplink channel bandwidth is narrow and/or the sending terminal is powered by batteries. The latter applies to the unidirectional communication such as the currently used one-way paging, where the clients can only receive information (be paged).

Let us now list the main advantages of the publishing mode.

The big advantage, first brought out in [3], is the *scalability* of the publishing mode. The access time⁴ for a data item that is published does not depend on the

⁴We define access time, later called latency, more formally below. It is the delay from the time the user issues a request until the time it downloads the data item.

number of clients who need this data item. Consequently, the publishing mode scales up well with the number of clients. For example, if stock information is published every minute, then it doesn't matter whether 10 clients or 10,000 clients are listening, the average waiting time will be 30 seconds. This will not be the case if stock information is provided on-demand.

Another big advantage deals directly with the power savings on the client's side. The publishing mode avoids the power consuming uplink transmissions from the clients (to the MSS) and also avoids keeping the CPU "on" continuously (upon providing a suitable directory - as explained later).

Hence, the publishing mode is very well suited for the wireless environment as it conserves both the power at clients and the wireless bandwidth.

Let us now elaborate further on our claim that the publishing mode allows energy efficient information access. To do this we introduce the concept of *low energy operations*. The low energy operations assume that the CPU has two basic modes of operations: *active mode* and, the energy efficient, *doze mode*. Additionally, we assume that the receiver can be temporarily completely shut off by the client. As indicated in the introduction both features can lead to substantial energy savings⁵.

Overview of the Communication Issues

For a full implementation of the publishing mode a number of communication issues have to be resolved. First, how can reliability be achieved for the publishing mode? Since the information is published by an MSS with no uplink transmissions from the clients (no explicit acknowledgments) - it is a challenging problem to achieve reliability. Secondly, how can a client access a specific published data item? Some form of synchronization is necessary. We assume that the published stream of data is packetized but how does the client know when the relevant packet of data is published? One of the important factor here is the type of the network which is used. In Ethernet, for example, it is impossible to determine the exact arrival time of a packet since there are no reservations. In reservation based protocols such as PRMA, the exact arrival time of a packet on the network is deterministic. We will address most of above communication issues later in the chapter (in section 3.3).

⁵Currently, the WaveLAN/PCMCIA card (a network adapter card providing wireless communication in a PC based LAN) draws approximately 3.4 watts for transmitting and 1.8 watts for receiving.

Some Possible Applications of the Publishing Mode

Below, we list some possible applications of the publishing mode:

- Financial Information System

An extension of the currently used Quotrex information system⁶ could periodically broadcast stock quotes together with the recent news clips about the individual stocks. Each broadcast could limit the stories to the most recent ones, e.g., those covering the last 2 hours. Thus, for example, IBM's stock quote could be followed by the information of the just released quarterly report. Notice, that each "edition" could possibly contain varying amount of information depending on the story size, etc. Individual users are interested only in specific stocks, thus, proper filtering mechanisms has to be established here.

- Airport Information System

Here information about airport facilities (shops, gates, restrooms, etc.) could be periodically published. Additionally, information about the arrivals and departures of airplanes is a good candidate for publishing as well.

- Emergency

Emergency information such as flood and storm warning, traffic jams, etc., can be published periodically.

We will use the first two examples later on, to illustrate some of the publishing techniques described in the paper.

2.1 Addressing Methods

In this subsection we will elaborate in detail on different addressing methods for the packetized downlink stream of published data. We assume that the client is only interested in a small subset of relevant records. For example, the user of a financial information system may be interested only in specific stocks and stories about this stock. Optimally, the client should have its CPU and receiver "on" only when the relevant record is published. In order to get to this goal, the published information should be *addressed* in such a way that

⁶Quotrex periodically broadcasts stock quotes on an FM radio channel.

address matching is a low energy operation. This can be accomplished in two basic ways:

- **Using Multicast Addresses**

The technique of sending a common data item to a group of clients is called *multicasting* and the addresses reserved for multicasting are called *multicast addresses*. In IP (Internet Protocol), this constitutes all 32 bit addresses starting with 1111. If a client is equipped with an Ethernet card then it can match the predefined set of packet addresses and still keep the CPU in a doze mode. Thus, if an individual record is given a multicast address then the client can keep his CPU in a doze mode until the published record's multicast address matches the address specified by the user. Notice that in this technique the client's receiver still should be "on".

Filtering by matching the multicast addresses is one way of ensuring that the clients do not have to process all the data items. A client gets a data item only if it is interested in that item. There are 28 bits available for *multicast groups*, which provide a total of 2^{28} addresses. The client then joins a multicast group (based on directory information, as described later) and can filter the relevant data using only the Ethernet card, without waking up the CPU and the whole OSI protocol stack. Note that there is usually a limit on the number of multicast addresses an Ethernet card can simultaneously listen to (the NS8390 chip can listen to 8 multicast addresses).

The same process can be accomplished if the mobile terminal is equipped with a pager. The low energy consuming hardware of the pager would perform the address matching and only when matching occurs does the CPU wake up.

- **Using Temporal Addresses**

The system clock is a low energy circuit. If the client knows the exact time when the relevant record is published it can switch off the receiver, put the CPU into the doze mode and switch back when the record arrives. To detect the arrival of the required record, the temporal matching has to be performed by the clock circuitry. This consumes very little energy. However, temporal matching requires a perfect or close to perfect synchronization which is not necessary in case multicast addressing is used. On the other hand, the potential energy savings of this method are larger than those of multicast addressing since the receiver can also be temporarily shut off (in addition to the CPU being in the doze mode).

Both methods require that the client knows the mapping between the record keys (primary or secondary) and (multicast, temporal) addresses. This mapping, called a directory, has to be either cached by the clients or published as well. We will discuss different methods of multiplexing the directory and the data in the next section.

2.2 Basic Notions

Now we introduce some of basic notions.

- **Bucket:** The smallest logical unit of publishing is called a **bucket**. We assume packetized data stream and for simplicity assume that a bucket of data fits exactly one packet - the basic unit of message transfer in networks.
- **Edition:** The data that is published is divided into logical segments called **editions**. Each edition consists of the data interleaved with directory information. It is made up of a number of buckets, some carrying data and others carrying the directory information.

A data file consists of a number of buckets, each bucket holding some data items which are identified by an attribute value. An attribute value is the value of the **search key**, i.e., the key based on which data items in a file are looked up. Clients retrieve the required data items (identified by the attribute value) by simple pattern matching.

In this paper, we deal with general purpose information services and not personal data files. Writes to the files are sent to the MSS and the MSS updates the files. However, updates to the file are reflected *only* between successive editions. Hence, the content of the current edition is completely determined before the start of the edition and remains unchanged throughout the current edition.

The following two parameters are of concern for publishing.

- **Tuning Time:** The time spent by a client listening to the publishing channel, in order to retrieve the required data. This determines the power consumed by the client for retrieving that data.

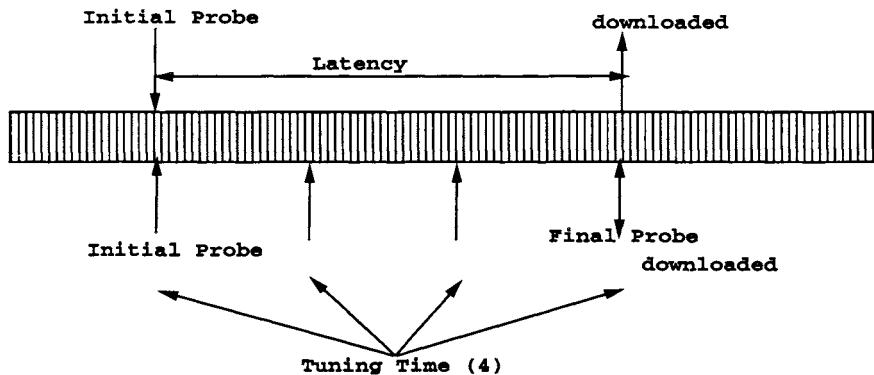


Figure 2 Latency and Tuning Time Illustration

- **Latency:** The time elapsed (on an average) from the moment a client requests data⁷, to the point when the required data is downloaded by the client.

Both the latency and the tuning time will be measured in terms of number of buckets.

Figure 2 illustrates the latency and the tuning time parameters. The latency is the time elapsed between the initial probe and the point when the required data is downloaded. The tuning time is the number of probes made into the publishing channel (in this case, four). Note that the worst case tuning time would be equal to the latency. In other words, the latency subsumes the tuning time.

Publishing requires new data organization and access methods. The *tuning time* in publishing roughly corresponds to the *access time* for disk based files. There is no parameter for disk based files that directly corresponds to the *latency* on the publishing channel.

Let us now look in more detail at the two ways in which publishing can be achieved.

⁷Note that in this case, “requesting” for data does not mean an uplink request to the server. It only means that the user has specified that s/he is interested in some data and the palmtop (client) takes up the task of getting that data. The moment of requesting data refers to the point at which the client takes up the task of getting the data.

3 PUBLISHING USING TEMPORAL ADDRESSES

For publishing using temporal addresses, a directory of the data file has to be published along with the data file. Let us first justify the use of a directory for publishing. If data is published without any form of directory, then in order to filter the required data items, the client will have to tune to the channel continuously until all the required data items are downloaded. On an average, the client has to be tuned to the channel for at least half the duration of the edition. This is unacceptable as it requires the client to be in the *active mode* for a long time, thereby consuming the scarce power resource. We would rather provide a selective tuning ability, enabling the client to come into the active mode only when data of interest is being published. Selective tuning will require that the server in addition to publishing the data, also publishes a directory. This directory will indicate the point in time when particular data items are published on the publishing channel. Clients will remain in the *dope mode* most of the time and tune into the publishing channel only when the required data is published.

One idea is to let all the clients cache a copy of this directory. However, this solution has the following disadvantages:

- In a mobile environment, when a client leaves a cell and enters a new cell, it will need the directory of the data being published in that cell. The directory it had cached in its previous cell may not be valid in the new cell.
- New clients that have no prior knowledge of the data organization of the edition will have to access the directory from the network. Palmtops that are turned off and switched on again can be thought of as such clients.
- Data that is published can change its content and grow or shrink any time between successive editions. In this case, the client has to refresh its cache. This may generate excessive traffic between the clients and the server. In fact, the directory will become a *hot spot*. Since we assume that the published data is very frequently accessed, the directory will also be accessed very frequently. Therefore, the directory is also published.
- If many different files are published on different channels then clients will need excessive storage for the directories of all the files being published. In palmtops storage is a scarce resource and this may not be possible.

Due to the above reasons, we publish the directory of the file in the form of an index or a hash function, along with the data. The publishing channel is the source of *all* information to the clients, including data as well as the directory.

In this section, we describe some of the techniques for publishing using *temporal addresses*. We will not discuss the details of the techniques nor will we provide the analysis of the techniques (these can be found in [7], [8] and [9]). In the following subsection we will consider a directory in the form of an index and in the next subsection we will consider a hash function as a directory in publishing.

3.1 Indexing on Air

The organization of the data file on the publishing channel involves determining how the data buckets and index buckets are interleaved to constitute an *edition*. We provide methods for *allocating* index together with data on the publishing channel. We do not provide new *types* of indexes, but rather new index allocation methods that will multiplex index and data in an efficient way with respect to conserving the power at the clients and utilizing the publishing bandwidth efficiently. Our methods allocate index and data for any type of index.

Below, the organization of buckets within an edition is described. In order to make each bucket self-identifying, all buckets have the following header information:

- *bucket_id*: the offset of the bucket from the beginning of the edition
- *edition_pointer*: the offset to the beginning of the next edition
- *index_pointer*: the offset to the beginning of the next index segment
- *bucket_type*: data bucket or index bucket

Pointer to a specific bucket *B* within the edition can be provided by specifying the *offset* of bucket *B*. The *offset* of bucket *B* from the current bucket is computed as the difference between the *bucket_id* of *B* and the *bucket_id* of the current bucket. The actual time of publishing of bucket *B* from the current bucket, is the product of (*offset* – 1) and the time necessary to publish a single bucket.

An index bucket is arranged as a sequence of (*attribute_value*, *offset*), where *offset* is a pointer to the bucket that contains the data item identified by

attribute_value. We consider multi-level index trees. The index tree will provide a sequence of pointers which eventually lead to the required data item. A data bucket is arranged as a sequence of data items.

Buckets containing the index are called **index buckets** and buckets containing the data are called **data buckets**. An **index segment** refers to the set of contiguous index buckets in an edition and a **data segment** refers to the set of data buckets published between two consecutive index segments.

The first bucket of each index segment has a tuple with the first field as the primary key of the data item that was published last and the second field as the offset to the beginning of the next edition. This is to guide the clients who have missed the required data item in the current edition and have to tune to the next edition.

Distributed Indexing Methods

How do we multiplex index with the data so that the latency and tuning time are minimized? As a naive approach we can broadcast index once for every broadcast of the file. In this way, however, the user who missed the index in the first tune-in (tuned in too late) will have to wait until the beginning of the next edition to get the next copy of the index, increasing the latency. Alternatively, the user could continuously tune to the broadcast and filter all incoming packets, increasing the tuning time. One can broadcast the index m times during each broadcast of the data file, minimizing the “bad luck” effect of missing the index in the first place. This technique, called $(1,m)$ indexing increases the overall size of the broadcast. The increase in the size of the broadcast is dependent on parameter m . There is an optimal value of m for which the latency is minimized, the value of the optimal m and other analysis of $(1,m)$ indexing is provided in [8]

The $(1,m)$ indexing algorithm can be improved by cutting down on the replication of the index. Distributed indexing is a technique in which the index is partially replicated. This method is based on the observation that there is no need to replicate the entire index between successive data segments - it is sufficient to have only the portion of index which indexes the data segment which follows it. Although the index is interleaved with data in both $(1,m)$ and distributed indexing, in distributed indexing only the *relevant* index is interleaved as opposed to $(1,m)$ indexing where the whole index is interleaved.

Three different index distribution methods can be considered. In all these three methods, the index is interleaved with data and the index segment describes only data in the data segment which *immediately* follows it. The methods differ in the degree of replication of index information.

- Non-replicated Distribution

Different index segments are disjoint. Hence, there is no replication.

- Entire Path Replication

The path from the root of the index (tree) to an index bucket B is replicated just before the occurrence of B .

- Partial Path Replication (Distributed Indexing)

Consider two index buckets B and B' . It is enough to replicate just the path from the least common ancestor of B and B' just before the occurrence of B' , provided we add some additional index information for navigation.

The non-replicated distribution and the entire path replication methods are two extremes. Distributed indexing aims at getting the best of both of these schemes.

Below, we give an example to illustrate the three kinds of index distribution methods. Consider a *financial information system* with the first level of the index listing various categories of companies (Electronics, Computer, Chemical, etc.). We will be concerned about the stock information (and the news clips that follow it) of the computer companies. Figure 3 shows different index distribution methods. Each method is illustrated showing two index levels.

Consider a client who is interested in news clips of IBM. In the non-replicated distribution, the first level of index (*Elec*, *Comp*, *Chem*) will be published at the beginning of each edition. The client who tunes in between points '1' and '2' will miss the first level of index and will have to wait until the beginning of the next edition. Thus, the latency in this method can be large.

In the entire path replication method, the whole path (the first level of the index and the specific company index) is replicated in front of each data segment. Clients who tune in between points '1' and '2' will be directed to the closest bucket containing the first level of index. The next step is to follow the pointers in the index hierarchy and ultimately download the required data. However,

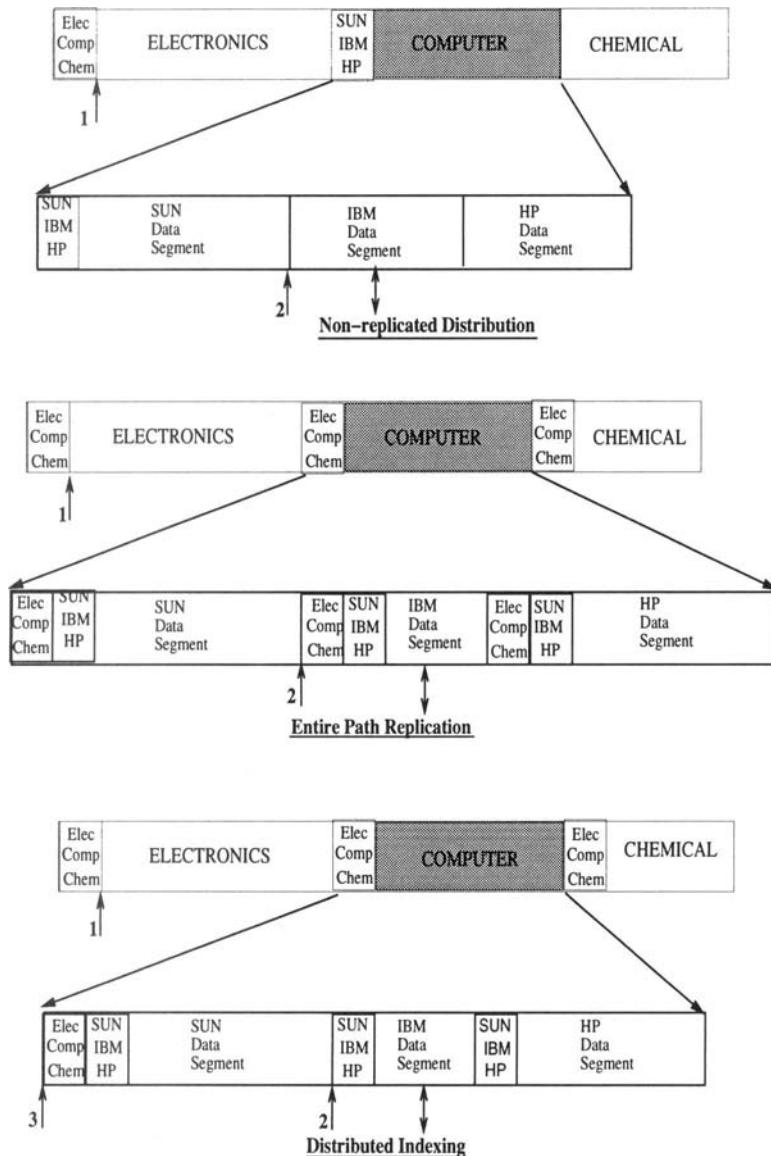


Figure 3 Index Distribution Methods

even in this method, the latency can still be large. Replication of the first index level results in the increase of the size of edition, and consequently, in the increase in latency.

In the distributed indexing method, only the “relevant” index is replicated in front of each data segment. In this method only the index corresponding to computer companies is replicated in front of each data segment. The first level index is published just once at the beginning of each category of the companies (once in the *Electronics* category, once in the *Computer* category and once in the *Chemical* category). Clients who tune in between points ‘1’ and ‘3’ are directed to the first level at point ‘3’, and clients who tune in between points ‘3’ and ‘2’ are directed to the index at point ‘2’. Next, the clients follows the index hierarchy to get to the relevant data. Since a client does not have to wait until the next edition if the data it is seeking is in front of it, the latency in this method is smaller than the latency in the previous two methods.

Distributed indexing reduces unnecessary replication of the first level of the index. As the number of levels of index and the capacity of an index bucket (number of records an index bucket can hold) increases, the saving due to distributed indexing becomes quite significant.

[8] describes distributed indexing in detail. Distributed indexing results in approximately the best latency and the best tuning time.

Secondary Indexing

Various data organization and access techniques based on secondary indexing are described in [9]. The $(1, m)$ indexing algorithm for general, secondary, clustering indexes⁸ is very similar to the $(1, m)$ indexing for primary index except that the access protocol changes. The access protocol has to take into account the fact that the client has to receive all the S buckets of data items with the required key (as opposed to just one data item as in the case of primary indexing). This is accomplished by downloading data items until a data item with a key other than the query key K is encountered. The distributed indexing algorithm for a clustering index is also very similar to the distributed indexing algorithm for a primary index. The access protocol has to be modified to take into account the selectivity of the attribute. Again, all data items satisfying

⁸Clustering index is an index on an attribute such that all records with the same value of that attribute (say S buckets of records) are located nearby on the disk, or in our case are broadcast successively.

the search key have to be downloaded since, in principle, there may be more than one record satisfying a query.

Usually, at most one index is clustered. Other indexes are defined over non-clustered attributes. An index can be allocated for non-clustered attributes by generalizing the distributed indexing technique. The proposed generalization is based on the observation that even if a file is not clustered on a particular attribute, it can be always decomposed into smaller fragments which are clustered.

Contrary to disk based files, the latency for accessing data items based on some attribute is dependent on the presence or absence of the index for other attributes. Indeed, each additional index increases the overall latency while decreasing the tuning time for the indexed attribute.

3.2 Hashing on Air

In general, different types of users may need different trade-offs between tuning time and latency. Some may value lower latency and may have more leverage in terms of the tuning time (larger laptops which have more powerful batteries), while others may prefer lower tuning time and may be ready to pay for it in terms of latency. Thus, we need *flexible* publishing techniques capable of accommodating different classes of users⁹. The indexing techniques provided in the previous subsection are not flexible in this sense since they do not benefit from a more lenient tuning time requirement.

Hashing based techniques can be provided to improve the direct access properties and reduce the tuning time. The hashing function can be given to a mobile client as part of the handoff protocol and it can be invalidated when the hashing function changes. If the hashing function changes very often, then the above method may not be good. In this case, the hashing function has to be published as well. Hashing based techniques do not require a separate directory to be published together with the data. The hashing parameters can be simply included in the data buckets.

Figure 4 illustrates two possible hashing methods. The first method, called Hashing A, uses the conventional hashing method for publishing, where every data item is hashed into a bucket numbered between 1 and n .

⁹Notice this whole discussion arises because we have two basic performance parameters instead of just one as in the case of disk based files.

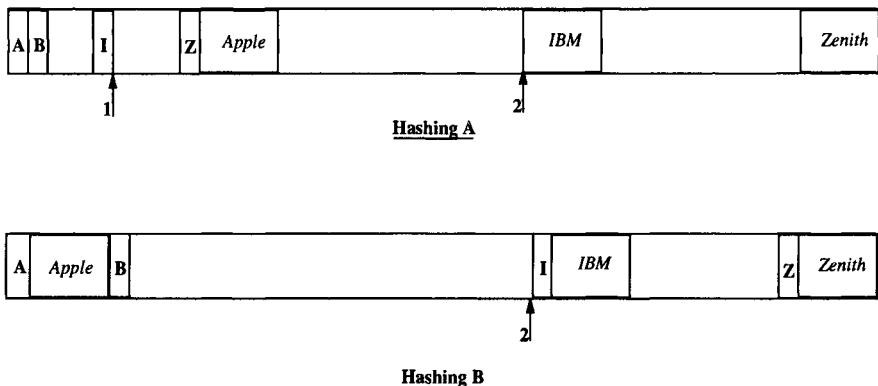


Figure 4 Hashing Techniques

For illustration purpose, consider a hashing function that maps the company name to the ordinal of the first character of the company name, e.g., Apple is hashed to 1, IBM is hashed to 9 and Zenith is hashed to 24. The overflow buckets (in case there are many companies with names starting with the same first letter) are published starting from the 25th bucket (after all 24 “primary” buckets are published). Thus all the information about companies whose name starts with ‘A’ will be continued after bucket 25. This will be followed by the information corresponding to companies whose name starts with ‘B’, etc. The first 24 buckets will contain, in addition to the company data, the pointer to the position in the broadcast where the overflow data (corresponding to this bucket) is published. We assume that the hashing function is acquired by the client upon registration. It may also be published as a part of each published bucket.

A client searching for information about IBM would determine that it has to tune to the 9th bucket (since the hash value of IBM is 9). If the initial tuning took place before the point ‘1’, then the client could successfully download the content of the bucket 9, switch into the doze mode, wake up at ‘2’ and download the rest of the required information. If the client tunes in between points ‘1’ and ‘2’ then it would miss bucket 9 (thereby missing the pointer to the position of occurrence of IBM) and it will have to tune in at the 9th bucket of the next edition¹⁰. This increases the latency for the client.

¹⁰If the client tunes in after the required information has passed (point ‘2’ in this case), then it will access it in the next edition.

Note that the server knows the content of next edition of information to be published, prior to publishing that edition. This observation can be used by the server to provide a hashing function which is “optimal” for that specific edition. This hashing technique which is better than Hashing A, is called Hashing B.

The detailed description and analysis of flexible publishing techniques based on hash functions (including Hashing A and Hashing B) are provided in [7].

Which hashing function minimizes the latency for a given file? For disk based files, the best function is the perfect hashing function (a function with no overflow). Contrary to disk based files, perfect hashing does not provide the minimum latency in publishing. This critical difference comes from the fact that the total number of buckets in an edition has immediate impact on the latency (the more one has to wait for the next edition). The perfect hashing function does not minimize the number of physical buckets necessary for the edition. On the contrary, the more overflow buckets are used, the smaller the total number of buckets in an edition. Indeed, the more overflow buckets the file has, the lesser “half-empty” buckets are present in an edition and this consequently results in better bucket utilization. The smaller the overflow buckets, the lower the tuning time. This reaches the minimum for the perfect hashing function. This further shows the basic differences between the data organization for publishing and data organization for disks. Hashing based technique for the publishing displays a *random access* behavior for the tuning time and a *sequential access* behavior for the access time (where the size of the file matters). Hence, hashing functions used in disk based files cannot be used directly for efficient publishing.

In the next subsection we elaborate on the communication issues.

3.3 Communication Issues

A number of practical communication issues have to be considered when publishing using temporal addresses. We now discuss these communication issues briefly. More details can be found in [9].

Synchronization:

For selective tuning clients have to be guaranteed that data will appear on the publishing channel at a pre-specified time, i.e., clients and the server (which publishes the data) have to be synchronized. This calls for the network to be

deterministic. This is a reasonable assumption for exclusively used channels such as Motorola's Satellite Networks (Embarc) where information is guaranteed to come at a specific time since there is only one server and the clients only listen.

In general, synchronization can be easily achieved if the MSS has a higher access priority than any client. This is not the case for Ethernet where the publishing traffic has to be interleaved with the on-demand traffic and it is very difficult to adequately predict the time of transmission. For such channels, the techniques for publishing have to be modified. One solution is to have the clients tune in *epsilon* (buckets) *ahead* of time (the required bucket is expected to arrive on the publishing channel). Epsilon varies from client to client depending on the accuracy of its clock and the time after which it is tuning into the publishing channel (the smaller the time period, the smaller the epsilon). Epsilon tuning has to be done taking the setup time into consideration, i.e., the client has to be in the active mode epsilon buckets in *advance*. Another solution, as described in section 4, is to use multicast addresses. In this case, each data item is multicast on a different multicast address and the client joins only multicast groups of data items it is interested in.

Setup Time:

The setup time refers to the time taken by the CPU for switching between the doze mode and the active mode (and vice versa). It can also refer to the time necessary to switch the receiver on (or off). In reality, each tuning out and tuning in operation will not occur instantaneously, but will take some small amount of time. Usually, the setup time is negligible compared to the time it takes to publish a bucket [14] and hence it can be ignored. In case the setup time cannot be ignored, then all data access techniques have to be modified - this is discussed in [9].

Reliability:

The error rates in wireless transmissions are much higher than the error rates on fixed networks. Thus, maintaining the reliability of publishing in wireless networks is an important issue. Publishing is inherently unreliable and methods are needed for recovery, especially in an error prone wireless environment. The conventional method of achieving reliability by means of retransmissions and acknowledgments is not suited for publishing. A flood of negative acknowledgments will result in the acknowledgment implosion problem.

However, publishing being *periodic* in nature, data is retransmitted anyway in the next edition. Thus, mitigating to a large extent, the effect of lost buckets. Clients who miss buckets in a particular edition can receive them in the next

edition with a very high probability. This results in an increase in latency for those clients who perceive lost data.

Security:

An information stream should be made secure and be available only to a selected group of clients (who pay for it). This can be done by assigning a multicast address for each service and providing this address only to clients who pay for the service¹¹.

4 PUBLISHING USING MULTICAST ADDRESSES

Hashing is a much more effective directory organization technique for multicast based addressing than it is for the temporal based addresses. Indeed, the multicast addresses do not need to be consecutive and therefore the multicast addressing space is much easier to handle using hashing than the strictly ordered temporal domain. There are two basic cases: the hashing function is acquired by the client upon entering the cell, or it is published under some well known multicast address.

The access protocol on the client side is simple. Depending on the data item required, the client computes the hash value of the data item. It then sets the network interface card to receive (listen to) all buckets on the multicast address which corresponds to the hash value computed. The hashing function can be given to the client as part of the handoff protocol. If the data file is very dynamic (changes often) then the hashing function may have to change. In this case, it might be better to have a multicast address reserved for publishing the hashing function. Whenever the hashing function changes, the altered hashing function is published on a predefined multicast address. This multicast address can then be passed on to the clients as part of the registration in a new cell.

The latency for any data item depends on the publishing strategy used by the server. If the server publishes all the data items periodically one after another, then the latency for any data item is half the duration required to publish all the data items (duration of an edition). A randomized publishing algorithm which minimizes the expected latency is described in [10].

¹¹Embarc uses this kind of solution.

The tuning time for accessing any data item depends on the number of multicast addresses available. If the number of multicast addresses is greater than or equal to the number of data items to be published, then the tuning time for any data item is equal to one. If the number of multicast addresses is less than the number of data items, then the tuning time for accessing a data item depends on the way in which the data items are distributed among the multicast addresses. In [15] an algorithm is provided that optimally partitions the data items into the available multicast addresses so as to get the minimum tuning time for accessing any data item.

Notice that no synchronization between the client and the server is necessary in case the multicast addressing is used.

Below, an example of a hypothetical information service that is structured in a hierarchical fashion is given.

4.1 An Example

In this example, we assume a hyperlink structure for the information service. In this structure, in order to access a given bucket of information, the user has to first access the parent of that bucket. Thus, the root of the structure and its children are likely to be hotspots of requests and are good candidates for publishing.

Consider figure 5, which shows an information service near an airport. The root bucket of the service will have a number of options. The user can navigate through the tree by selecting the required hyperlinks. The buckets with dark background are published (the multicast address for the bucket is specified on the top of the bucket) and the rest of the buckets are provided on demand. A leaf bucket has various information about a particular flight, like the terminal it is expected to depart from, the boarding gate, the time of arrival, etc. Some of these buckets could be provided on-demand throughout the day, but 2 hours before the time of departure of the flight, they can be published. As the time for departure draws nearer, the frequency of publishing of that bucket can increase and eventually after a few minutes of the departure of the flight, this bucket can no longer be published and it can be provided on-demand again. Thus, a bucket goes through different data delivery modes through the duration of the day. Most of the time a bucket is identified by a specific IP address, using which a client can access the bucket by the on-demand mode. A couple of hours before the departure of the flight, the bucket is published using a temporal

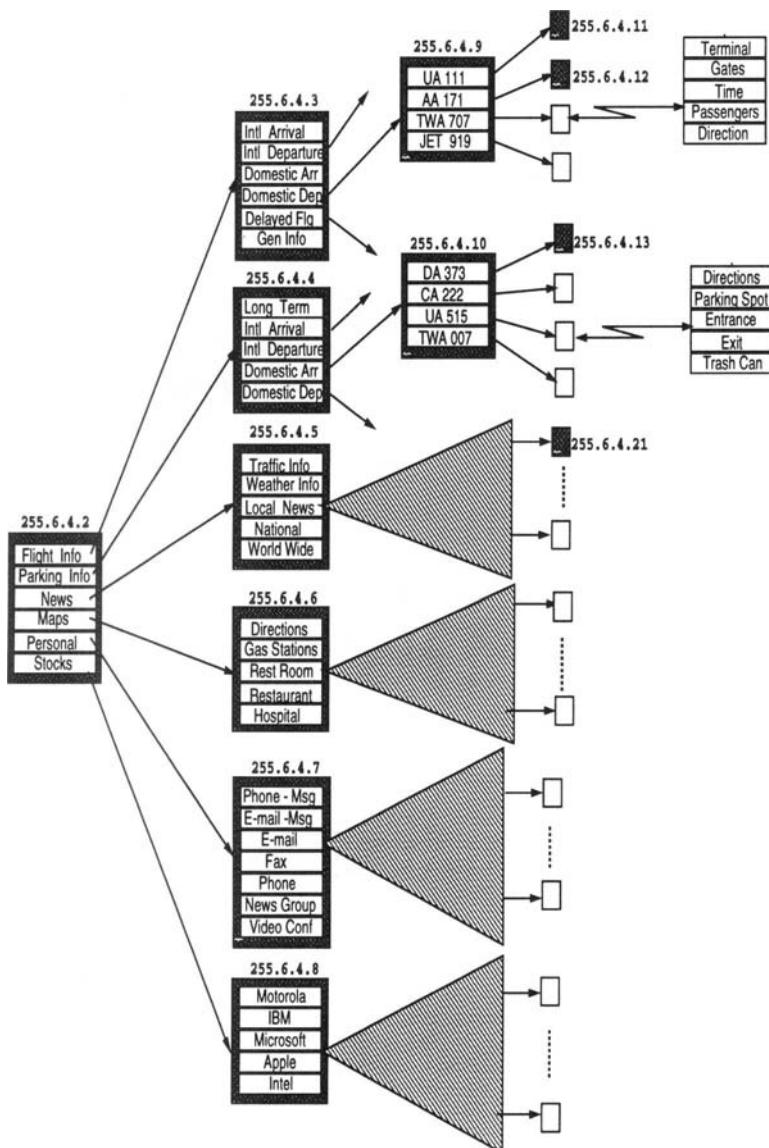


Figure 5 An Information Service near an Airport

address. Clients can access the bucket by listening to the publishing channel, say, at every 10th minute of the hour. Immediately, say 15 minutes, before the departure of the flight, the bucket is published using a multicast address. Clients join the multicast group corresponding to the required bucket.

Note that if the user is familiar with the layout of the information services (say, the client has cached the directory), s/he can avoid going through the root and accessing all the buckets along the path down to the leaf. For example, if the user knows the flight number and needs to get the flight's arrival time s/he can just specify the flight number followed by the required information.

5 ADAPTIVE PUBLISHING

Usually, the frequencies of requests for data items vary. Some data items are requested very often and some others are rarely requested. The general philosophy should be to publish the more frequently requested data items more often.

In [10] we consider publishing based on the access frequencies of the data items. Additionally, as indicated earlier, the popular data items will be published and the rest will be provided on demand. An algorithm called *adaptive scheduling* algorithm is described in [10] for deciding which data items to publish and which data items to provide on demand. This algorithm also optimally divides the overall channel bandwidth between the publishing channel and the on-demand channel.

5.1 General Model of a Server

How does the MSS decide which data items to publish and which provide on demand? Let us consider all options starting from the standard client server model (pure on-demand mode).

- **Exclusive On-demand**

All the data items are assigned to the on-demand channel and the scheduling is by the M/M/1 queuing model. The queuing model can be varied.

This method can only handle up to a certain number of requests per unit of time, beyond which the system will be blocked and the access time becomes

infinite. This method has low power efficiency - any request requires a power consuming uplink request.

This solution is desirable in case power efficiency is not an issue (a mobile client powered from a car battery) and the volume of requests is not very high.

- **Batching**

The on-demand requests are not handled immediately. All requests for a data item are collected over a period of L time units and are served by the same single download. In this case, each request is delayed by at most L time units before it is downloaded.

The bandwidth efficiency is better than in the case of the exclusive on-demand method. However, the power efficiency remains the same as in the previous method. The recommendations are similar as in the previous case.

- **Exclusive Publishing**

All the data items are assigned to the publishing channel and the scheduling is simple - the data items are published one after another. All the data items are published equal number of times.

For small number of requests this method has a very low bandwidth efficiency. However, irrespective of the number of requests, there is no blocking at all. If the directory is provided then, this method has the best power efficiency, since uplink transmissions are completely avoided.

Wireless broadcasting services such as radio and TV use this publishing method. This method is useful when dealing with one way communication, for instance, it is useful for one way pagers and pager based communication. As indicated earlier, this method is very useful for the dissemination of hot spots. Another possible use is in situations when uplink transmission is possible, but requires lot of power due to the large distance between the mobile client and the MSS.

- **Frequency Based Exclusive Publishing**

The data items are published in the ratio of their frequencies. This method has a better bandwidth efficiency than the exclusive publishing method and is equally power efficient. Recommendations are similar as in the previous case.

The next two strategies mix the publishing and the on-demand modes. They strike a compromise between the best bandwidth efficiency and the best power efficiency.

- **Greedy Publishing**

In this method the on-demand traffic has a higher priority than publishing traffic and consequently, the data items that are published are downloaded by the MSS only when there is no on-demand traffic. Multicast addresses are used by the MSS for publishing. Before submitting an uplink request for the required data item, the client first *listens* to the publishing channel. Only if that data item is not provided within a specific period L , does the client submit an uplink request.

Data items are published during the unused time between the retrieval of data items by the on-demand mode. During the unused time, data items are published in such a way that, the number of uplink transmissions are minimized. Let λ_i denote the number of requests per unit time (access frequency) for the data item D_i . Let T_i denote the time period for which data item D_i hasn't been sent (either by publishing or by on-demand). The algorithm is to publish data item D_j such that the following inequality is satisfied:

$$\lambda_j * T_j \geq \lambda_i * T_i \quad \forall i$$

This protocol uses the bandwidth very efficiently but, the blocking problem characteristic of the pure on-demand mode cannot be avoided. This method is not as power efficient as the pure publishing methods, since uplink transmissions cannot be completely avoided.

- **Adaptive Publishing**

Decisions about the bandwidth allocation between the two channels are made dynamically, based on the request patterns of clients. For a given bandwidth efficiency, such protocols have the best power efficiency. One such algorithm is described in [10].

The above list is by no means exhaustive and there is a broad range of other possible ways in which the MSS can schedule its services between the publishing and the on-demand channels.

5.2 Gathering Access Profile of Clients

For adaptive publishing the frequencies of access for the various data items have to be available. Gathering statistics for the data items that are accessed on-demand is straightforward. The key difficulty is to obtain the frequency

of access for the data items that are published. Obtaining such *feedback* is contradictory to the nature and motivation of publishing which assumes no uplink transmissions from the clients regarding the published data items.

Below, we give some methods for gathering statistics for the data items that are published. They vary on how much the MSS is “up to date” regarding the access frequencies of the data items.

- When the client sends a request to the MSS for accessing a data item provided by the on-demand mode, it piggybacks an additional message. The piggybacked message will include the number of times it accessed each data item (that was published) between the last access of a data item accessed by the on-demand mode and the current request.
- If the information service is organized as a tree (as in the WWW), then it is may be the case that the top few levels are published and the rest, especially the leaves, are provide on-demand. Hence, the request for each leaf goes to the MSS. The frequency of access for each of the data items in the interior of the tree can be extrapolated. The frequency of access of a data item (in the interior of the tree) is equal to the sum of frequency of access of its children.
- In case the client informs the MSS just before disconnection (of its impending disconnection), then it can piggyback a message indicating the number times it accessed each data item (belonging to the set of published items) in the duration of its registration under that MSS.
- The demand for the published data items can be measured periodically, by taking a data item that is being currently published and providing it on-demand for a while, thereby estimating as to how frequently that data item is requested. This frequency of access of a data item is maintained until the next time that data item is dropped again from the set of published items. Depending on how accurately the MSS has to be informed of the frequency of access for various data items, the data items can be dropped often or just once in a while.
- A representative sample of the client population can be monitored to extrapolate the access frequencies of the various data items.

6 OTHER INFORMATION DELIVERY METHODS

In this subsection we describe data dissemination methods in case the data changes often and in case the clients may frequently be disconnected.

Disconnection will be a feature not a bug in the wireless environment. Users will disconnect in order to save batteries or access cost and hence proper disconnection and reconnection protocols will have to be designed. Frequent *disconnection* is a major factor that will affect caching solutions [2]. In [2] we argue that solutions which require the server to maintain the precise state of the clients, including information about who is disconnected, are impractical due to mobility of the clients and possible frequent disconnections. Hence, we have advocated a *stateless* solution in which the server does not maintain any information about the state of the clients but rather periodically multicasts an *invalidation report* which, for each data item provides the last timestamp when this data item changed. Publishing information about changes (invalidations) rather than the information itself is called *delta publishing*. Delta publishing is useful for disconnected clients because when a client becomes active, it can compare the timestamp in the cache with the one obtained from the invalidation report and decide to submit the “uplink message” only if the timestamp of the cache is earlier (smaller). The granularity of invalidation report could be much higher than individual cached data items. For example, one entry in the invalidation report can represent to all NorthEast bound flights of a particular airline. Invalidiation report entry (NE, 11:05AM) will mean that the last change to the North East schedules of that airline occurred at 11:05AM. Thus, the clients who were disconnected, say, at 11:30AM and had a copy of the cache with timestamp of 11:15 AM may avoid energy consuming uplink request.

A new set of issues come up when *location dependent data* is cached. Data such as the name of nearest printer, supermarket, hospital, restaurant, etc. change when users move. Assuming that the client caches several location dependent data items, we would like to avoid a situation where each move by the client would cause a “cache refresh” request to be sent uplink by the client. Different data items may vary in their sensitivity to the location of the user. We discuss the proposed solution to cache consistency of location dependent data.

The initial work in this direction has been published in [2]. In that manuscript different types of invalidation reports has been introduced, namely broadcasting timestamps, amnesic terminals and signatures. In the *broadcasting timestamps* method, the invalidation report is composed of the timestamps of the latest

change for items that have had updates in the last w seconds. The mobile unit listens to the report and updates the status of its cache. For each item cached, the mobile unit either purges it from the cache (when the item is reported to have changed at a time larger than the timestamp stored in the cache), or updates the cache's timestamp to the timestamp of the report (if the item was not mentioned there). Note that this is done for every item in the cache, regardless of whether there is a pending query to this item or not. In the *amnesic terminals* strategy, the server only broadcasts the identifiers of the items that have changed since the last invalidation report. A mobile client that has been disconnected for while, needs to start rebuilding its cache from scratch. As before, we assume that if the client is awake, it listens constantly to reports and modifies its cache by dropping invalidated items. Finally, in *signatures* the checksums computed over the values of items are sent in the invalidation reports. This technique has been used for efficient testing if two versions of a file differ. In [2], signatures is used to provide a compressed view of all data items which have changed.

In [2] we provide extensive analysis comparing the performance the three techniques mentioned above. To this end we divide all mobile clients into sleepers (units which are disconnected for a long time) and workaholics (units which are on most of the time). The comparison shows that different caching techniques are good for different client populations. For example, signatures are best for long sleepers when the period of disconnection is long and difficult to predict. Broadcasting with timestamps is effective for scenarios when the rate of queries is greater than the volume of updates. Finally, the amnesic terminals are best for workaholics.

The work described in [2] is only the initial effort in this direction. More work is necessary in order to provide a comprehensive solution for the problem of cache invalidation in mobile wireless environments. How do we decide whether to use stateless caching techniques (when MSS does not maintain information about the state of client's caches) or stateful methods? . When do we decide whether to simply publish a new value rather than to use delta publishing by sending an invalidation report? The last question is similar to the distinction between change propagation and cache invalidation in standard caching. It is important to develop methods to decide when it is more beneficial to publish a data item rather than offer it in the delta publishing mode. It is also necessary to examine methods based on quasicopies [1] in the context of delta publishing, when invalidation report is based on larger granularity of changes (e.g., only stock moves by more than 1% lead to cache invalidation). Flexible methods based on quasicopies are extremely important in environments with widely

varying bandwidth - the lesser the bandwidth available, the more imprecise the quasicopies become.

Finally, the issue of caching location dependent data deserves more attention. To avoid refreshing the cache of location dependent data items upon every move, we associate with each location dependent data item a concept of *scope*. The scope is defined as the set of locations surrounding the current client's location where the data item has still the same value. For example, the nearest printer may be the same for the whole building and moving between different floors would not affect the value of the nearest printer¹². In the proposed solution the client caches a data item along with its scope and refreshes the cache only when it moves out of the scope of that data item. We would like to extend this technique for more complex queries that depend on location dependent data.

7 CONCLUSIONS AND IMPLEMENTATION STATUS

Future wireless information services will be provided through collections of autonomous cells offering information services of widely varying geographic scope. The information content, as well as the methods of its delivery may differ from cell to cell. In this paper, we have described how the publishing mode (for information delivery) can be combined with the standard client server interactions to provide solutions which are both bandwidth and energy efficient.

In general, the more frequent the requests for a particular information, the more it is worthy of publishing. It may also be appropriate to publish location dependent data if the client population is highly mobile. One such example is an airport setting, where there are a lot of requests for common information such as airport layout and plane schedules. Although the airport layout (locations of the restrooms, shops, etc.) does not change, the population of clients is highly transient and, thus, it makes sense to publish such information. Time dependent data, i.e., information which changes often, may also be appropriate for publishing. This is true irrespective of the mobility of the clients. The data delivery mode in a cell will also be determined by its size. The smaller the cell

¹²The mobile user usually is aware of his location down to the level of a cell by recognizing a beacon periodically transmitted by the MSS. Using Global Positioning System (GPS) may provide more precise determination of location in the future.

size (which translates to smaller number of clients) the less likely is the use of publishing mode.

Implementation Status

The publishing mode is currently being investigated and implemented at our lab, both in a wireless LAN setting (Wavelan) and in an outdoor radio (CDPD) setting. The current implementation which is based on multicast addressing, is a stock server application. In this application stock quotes are periodically published on a wireless LAN and the clients filter the published data stream to access information about individual stocks.

The earlier mentioned *Infostations* project will use publishing mode as one of the basic modes in which the *infostations* will provide information to their clients. The adaptive methods described in this paper, which are different combinations of the publishing and the on-demand modes, will be used for the *Infostations* project.

REFERENCES

- [1] Rafael Alonso and Hank Korth, "Database Issues in Nomadic Computing," *MITL Technical Report-36-92*, December 1992.
- [2] Daniel Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environment," *Proc. of ACM-SIGMOD, International Conference on Data Management* - Minnesota - pp.1-12, May 1994.
- [3] T. F. Bowen et al., "The Datacycle Architecture," *Communications of the ACM*, Vol. 35, No. 12, pp. 71-81, December 1992.
- [4] David Cheriton, "Dissemination-Oriented Communication Systems," *Stanford University, Technical Report*, 1992.
- [5] David Gifford et. al., "The Application of Digital Broadcast Communication to Large Scale Information Systems," *IEEE Journal on selected areas in communications*, Vol 3, No. 3, pp. 457-467, May 1985.
- [6] G. Herman et al., "The Datacycle Architecture for Very Large High Throughput Database Systems," *Proc of ACM SIGMOD Conference*, San Francisco-California, pp. 97-103, May 1987.

- [7] T. Imielinski, S. Viswanathan and B. R. Badrinath, "Power Efficient Filtering of Data on Air," *Proc. of 4 th International Conference on EDBT (Extending DataBase Technology)*, Cambridge-U.K, pp. 245-258, March 1994.
- [8] T. Imielinski, S. Viswanathan and B. R. Badrinath, "Energy Efficient Indexing on Air," *Proc. of ACM-SIGMOD, International Conference on Data Management*, Minnesota, pp. 25-36, May 1994.
- [9] T. Imielinski, S. Viswanathan and B. R. Badrinath, "Data on Air : Organization and Access," To appear in *IEEE Transactions in Data and Knowledge Engineering*, 1995.
- [10] T. Imielinski and S. Viswanathan, "Adaptive Wireless Information Systems," *Proc. of SIGDBS (Special Interest Group in DataBase Systems) Conference*, pp. 19-41, Tokyo-Japan, October 94.
- [11] Christopher O'Malley, "Turning Down the Power," *Mobile Office*, pp. 118-120, June 1993.
- [12] Samuel Sheng, Ananth Chandrasekaran, and R. W. Broderson, "A Portable Multimedia Terminal for Personal Communications," *IEEE Communications Magazine*, pp. 64-75, December 1992.
- [13] Swarup Acharya, Rafael Alonso, Michael Franklin and Stanley Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments," *Proc. of ACM-SIGMOD, International Conference on Management of Data*, San Jose, pp. 199-210, June 1995.
- [14] Steele, Raymond., "Mobile Radio Communications," *Pentech Press, Publishers - London*, 1992.
- [15] S. Viswanathan, "Publishing in Wireless and Wireline Environments," *Ph.D. Thesis - Rutgers, The State University of New Jersey*, 1994.
- [16] Lee, William C. Y., "Mobile Cellular Telecommunications Systems," *New York: McGraw-Hill*, 1989.
- [17] J.W. Wong., "Broadcast Delivery," *Proc. of the IEEE*, Vol. 76, No. 12, pp. 1566-1577, December 1988.

BROADCAST DISKS: DATA MANAGEMENT FOR ASYMMETRIC COMMUNICATION ENVIRONMENTS

Swarup Acharya*, Rafael Alonso**,
Michael Franklin***, and Stanley Zdonik*

* Department of Computer Science, Brown University
Providence, Rhode Island 02912

** Matsushita Information Technology Laboratory
Princeton, NJ 08540

*** Department of Computer Science, University of Maryland
College Park, Maryland 20742
USA

ABSTRACT

This paper proposes the use of repetitive broadcast as a way of augmenting the memory hierarchy of clients in an asymmetric communication environment. We describe a new technique called "Broadcast Disks" for structuring the broadcast in a way that provides improved performance for non-uniformly accessed data. The Broadcast Disk superimposes multiple disks spinning at different speeds on a single broadcast channel — in effect creating an arbitrarily fine-grained memory hierarchy. In addition to proposing and defining the mechanism, a main result of this work is that exploiting the potential of the broadcast structure requires a re-evaluation of basic cache management policies. We examine several "pure" cache management policies and develop

Previously appeared in Proceedings of the ACM SIGMOD International Conference on the Management of Data, 1995, pp. 199-210. Copyright ©1995 by the Association for Computing Machinery, Inc. Reprinted by permission. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept, ACM Inc., fax +1 (212) 869-0481, or (permissions@acm.org)

and measure implementable approximations to these policies. These results and others are presented in a set of simulation studies that substantiates the basic idea and develops some of the intuitions required to design a particular broadcast program.

1 INTRODUCTION

1.1 Asymmetric Communication Environments

In many existing and emerging application domains the *downstream* communication capacity from servers to clients is much greater than the *upstream* communication capacity from clients back to servers. For example, in a wireless mobile network, servers may have a relatively high bandwidth broadcast capability while clients cannot transmit or can do so only over a lower bandwidth (e.g., cellular) link. Such systems have been proposed for many application domains, including traffic information systems, hospital information systems, public safety applications, and wireless classrooms (e.g.,[Katz94, Imie94a]). We refer to these environments as *Asymmetric Communications Environments*.

Communications asymmetry can arise in two ways: the first is from the bandwidth limitations of the physical communications medium. An example of physical asymmetry is the wireless environment as described above; stationary servers have powerful broadcast transmitters while mobile clients have little or no transmission capability. Perhaps less obviously, communications asymmetry can also arise from the patterns of *information flow* in the application. For example, an information retrieval system in which the number of clients is far greater than the number of servers is asymmetric because there is insufficient capacity (either in the network or at the servers) to handle the simultaneous requests generated by the multiple clients.

Because asymmetry can arise due to either physical devices or workload characteristics, the class of asymmetric communications environments spans a wide range of important systems and applications, encompassing both wired and wireless networks. Examples include:

- Wireless networks with stationary base stations and mobile clients.
- Information dispersal systems for volatile, time-sensitive information such as stock prices, weather information, traffic updates, factory floor information, etc.

- Cable or satellite broadcast television networks with set-top boxes that allow viewers to communicate with the broadcasting home office, and video-on-demand servers.
- Information retrieval systems with large client populations, such as mail-order catalog services, mutual fund information services, software help desks, etc.

1.2 Broadcast Disks

In traditional client-server information systems, clients initiate data transfers by sending requests to a server. Such systems are *pull-based*; the clients “pull” data from the server in order to provide data to locally running applications. Pull-based systems are a poor match for asymmetric communications environments, as they require substantial upstream communications capabilities. To address this incompatibility, we have proposed a new information system architecture that exploits the relative abundance of downstream communication capacity in asymmetric environments. This new architecture is called *Broadcast Disks*. The central idea is that the servers exploit their advantage in bandwidth by *broadcasting* data to multiple clients. We refer to this arrangement as a *push-based* architecture; data is pushed from the server out to the clients.

In this approach, a server continuously and repeatedly broadcasts data to the clients. In effect, the broadcast channel becomes a “disk” from which clients can retrieve data as it goes by. Broadcasting data has been proposed previously [Herm87, Giff90, Imie94b]. Our technique differs, however, in two ways. First, we superimpose *multiple disks* of different sizes and speeds on the broadcast medium, in effect, creating an arbitrarily fine-grained memory hierarchy. Second, we exploit client storage resources as an integral part of this extended memory hierarchy.

The broadcast is created by assigning data items to different “disks” of varying sizes and speeds, and then multiplexing the disks on the broadcast channel. Items stored on faster disks are broadcast more often than items on slower disks. This approach creates a memory hierarchy in which data on the fast disks are “closer” to the clients than data on slower disks. The number of disks, their sizes, and relative speeds can be adjusted, in order to more closely match the broadcast with the desired access probabilities at the clients. If the server has an indication of the client access patterns (e.g., by watching their previous activity or from a description of intended future use from each client), then hot pages (i.e., those that are more likely to be of interest to a larger part

of the client community) can be brought closer while cold pages can be pushed further away.

1.3 Scope of the Paper

In this paper, we focus on a restricted broadcast environment in order to make an initial study of the broadcast disk approach feasible. The restrictions include:

- The client population and their access patterns do not change. This implies that the content and the organization of the broadcast program remains static.
- Data is read-only; there are no updates either by the clients or at the servers.
- Clients retrieve data items from the broadcast on demand; there is no prefetching.
- Clients make no use of their upstream communications capability, i.e., they provide no feedback to servers.

In this environment, there are two main interrelated issues that must be addressed:

1. Given a client population and a specification of their access probabilities for the data items, how does the server construct a broadcast program to satisfy the needs of the clients?
2. Given that the server has chosen a particular broadcast program, how does each client manage its local data cache to maximize its own performance?

The remainder of the paper is organized as follows. Section 2 discusses the way in which we structure the broadcast program and Section 3 shows how the client's cache management policy should be designed to complement this choice. Section 4 describes our simulation model and Section 5 develops the main experimental results derived from this model. Section 6 compares our work to previous work on repetitive broadcast. Section 7 summarizes our results and describes our future work.

2 STRUCTURING THE BROADCAST DISK

2.1 Properties of Broadcast Programs

In a push-based information system, the server must construct a broadcast “program” to meet the needs of the client population. In the simplest scenario, given an indication of the data items that are desired by each client listening to the broadcast, the server would simply take the union of the requests and broadcast the resulting set of data items cyclicly. Such a broadcast is depicted in Figure 1.

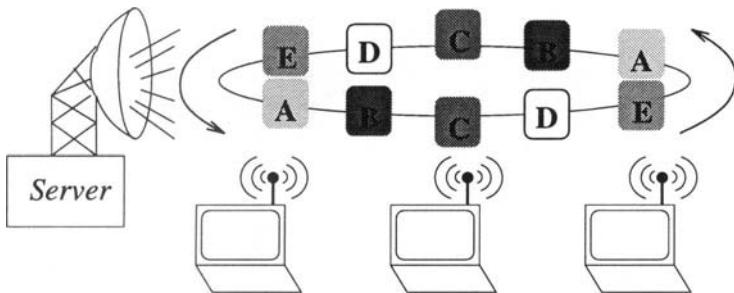


Figure 1 A Flat Broadcast Program

When an application running on a client needs a data item, it first attempts to retrieve that item from the local memory or disk. If the desired item is not found, then the client monitors the broadcast and waits for the item to arrive.¹ With the flat broadcast, the expected wait for an item on the broadcast is the same for all items (namely, half a broadcast period) regardless of their relative importance to the clients. This “flat” approach has been adopted in earlier work on broadcast-based database systems such as Datacycle[Bowe92] and [Imie94a].

Alternatively, the server can broadcast different items with differing frequency. Such a broadcast program can emphasize the most popular items and de-emphasize the less popular ones. Theoretically, the generation of such non-flat broadcast programs can be addressed as a *bandwidth allocation* problem; given all of the client access probabilities, the server determines the optimal percentage of the broadcast bandwidth that should be allocated to each item. The broadcast program can then be generated randomly according to those

¹This discussion assumes that broadcast items are self-identifying. Another option is to provide an index, as is discussed in [Imie94b].

bandwidth allocations, such that the *average* inter-arrival time between two instances of the same item matches the needs of the client population. However, such a random broadcast will not be optimal in terms of minimizing expected delay due to the variance in the inter-arrival times.

A simple example demonstrating these points is shown in Figure 2. The figure shows three different broadcast programs for a data set containing three equal-length items (e.g., pages). Program (a) is a flat broadcast, while (b) and (c) both broadcast page A twice as often as pages B and C. Program (b) is a *skewed* broadcast, in which subsequent broadcasts of page A are clustered together. In contrast, program (c) is *regular*; there is no variance in the inter-arrival time for each page. The performance characteristics of program (c) are the same as if page A was stored on a disk that is spinning twice as fast as the disk containing pages B and C. Thus, we refer to program (c) as a *Multi-disk* broadcast.

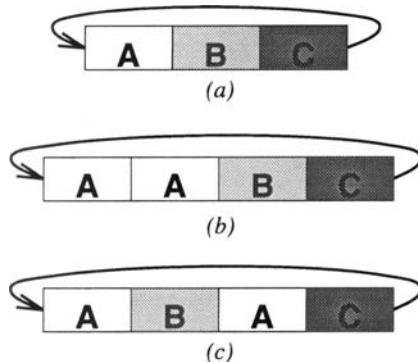


Figure 2 Three Example Broadcast Programs

Table 1 shows the expected delay for page requests given various client access probability distributions, for the three different broadcast programs. The expected delay is calculated by multiplying the probability of access for each page times the expected delay for that page and summing the results. There are three major points that are demonstrated by this table. The first point is that for uniform page access probabilities (1/3 each), a flat disk has the best expected performance. This fact demonstrates a fundamental constraint of the Broadcast Disk paradigm, namely, that due to fixed bandwidth, *increasing the broadcast rate of one item must necessarily decrease the broadcast rate of one or more other items*. The second point, however, is that as the access probabilities become increasingly skewed, the non-flat programs perform increasingly better.

Access Probability			Expected Delay (in broadcast units)		
A	B	C	Flat (a)	Skewed (b)	Multi-disk (c)
0.333	0.333	0.333	1.50	1.75	1.67
0.50	0.25	0.25	1.50	1.63	1.50
0.75	0.125	0.125	1.50	1.44	1.25
0.90	0.05	0.05	1.50	1.33	1.10
1.0	0.0	0.0	1.50	1.25	1.00

Table 1 Expected Delay For Various Access Probabilities

The third point demonstrated by Table 1 is that the Multi-disk program always performs better than the skewed program. This behavior is the result of the so-called Bus Stop Paradox. If the inter-arrival rate (i.e., broadcast rate) of a page is fixed, then the expected delay for a request arriving at a random time is one-half of the gap between successive broadcasts of the page. In contrast, if there is variance in the inter-arrival rate, then the gaps between broadcasts will be of different lengths. In this case, the probability of a request arriving during a large gap is greater than the probability of the request arriving during a short gap. Thus the expected delay increases as the variance in inter-arrival rate increases.

In addition to performance benefits, a Multi-disk broadcast has several other advantages over a random (skewed) broadcast program. First, the randomness in arrivals can reduce the effectiveness of some prefetching techniques that require knowledge of exactly when a particular item will next be broadcast [Zdon94]. Second, the randomness of broadcast disallows the use of “sleeping” to reduce power consumption (as in [Imie94b]). Finally, there is no notion of “period” for such a broadcast. Periodicity may be important for providing correct semantics for updates (e.g., as was done in Datacycle [Herm87, Bowe92]) and for introducing changes to the structure of the broadcast program. For these reasons, we argue that a broadcast program should have the following features:

- The inter-arrival times of subsequent copies of a data item should be fixed.

- There should be a well defined unit of broadcast after which the broadcast repeats (i.e., it should be periodic).
- Subject to the above two constraints, as much of the available broadcast bandwidth should be used as possible.

2.2 Broadcast Program Generation

In this section we present a model for describing the structure of broadcast programs and describe an algorithm that generates broadcast programs with the desired features listed in the previous section. The algorithm imposes a Multi-disk structure on the broadcast medium in a way that allows substantial flexibility in fitting the relative broadcast frequencies of data items to the access probabilities of a client population.

The algorithm has the following steps (for simplicity, assume that data items are “pages”, that is, they are of a uniform, fixed length):

1. *Order the pages from hottest (most popular) to coldest.*
2. *Partition the list of pages into multiple ranges*, where each range contains pages with similar access probabilities. These ranges are referred to as *disks*.
3. *Choose the relative frequency of broadcast for each of the disks.* The only restriction on the relative frequencies is that they must be integers. For example given two disks, disk 1 could be broadcast three times for every two times that disk 2 is broadcast, thus, $\text{rel_freq}(1) = 3$, and $\text{rel_freq}(2) = 2$.
4. *Split each disk into a number of smaller units.* These units are called *chunks* (C_{ij} refers to the j^{th} chunk in disk i). First, calculate max_chunks as the Least Common Multiple (LCM) of the relative frequencies. Then, split each disk i into $\text{num_chunks}(i) = \text{max_chunks} / \text{rel_freq}(i)$ chunks. In the previous example, $\text{num_chunks}(1)$ would be 2, while $\text{num_chunks}(2)$ would be 3.
5. *Create the broadcast program* by interleaving the chunks of each disk in the following manner:

```

01 for  $i := 0$  to  $\text{max\_chunks} - 1$ 
02   for  $j := 1$  to  $\text{num\_disks}$ 
03      $k := i \bmod \text{num\_chunks}(j)$ 
04     Broadcast chunk  $C_{j,k}$ 
05   endfor
06 endfor

```

Figure 3 shows an example of broadcast program generation. Assume a list of pages that has been partitioned into three disks, in which pages in disk 1 are to be broadcast twice as frequently as pages in disk 2, and four times as frequently as pages in disk 3. Therefore, $\text{rel_freq}(1) = 4$, $\text{rel_freq}(2) = 2$, and $\text{rel_freq}(3) = 1$. These disks are split into chunks according to step 4 of the algorithm. That is max_chunks is 4, so $\text{num_chunks}(1) = 1$, $\text{num_chunks}(2) = 2$, and $\text{num_chunks}(3) = 4$. Note that the chunks of different disks can be of differing sizes. The resulting broadcast consists of 4 *minor cycles* (containing one chunk of each disk) which is the LCM of the relative frequencies. The resulting broadcast has a period of 16 pages. This broadcast produces a three-level memory hierarchy in which disk one is the smallest and fastest level and disk three is the largest and slowest level. Thus, the multi-level broadcast corresponds to the traditional notion of a memory hierarchy.

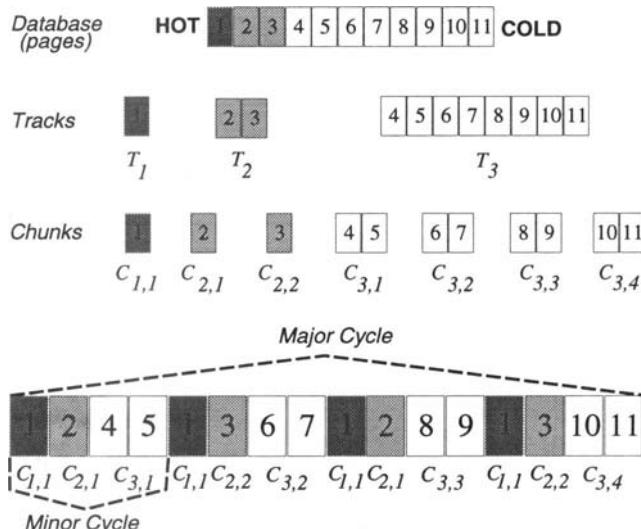


Figure 3 Deriving a Server Broadcast Program

The algorithm produces a periodic broadcast program with fixed inter-arrival times per page. Some broadcast slots may be unused however, if it is not possible to evenly divide a disk into the required number of chunks (i.e., in Step 4 of the algorithm). Of course, such extra slots need not be wasted, they can be used to broadcast additional information such as indexes, updates, or invalidations; or even for extra broadcasts of extremely important pages. Furthermore, it is anticipated that the number of disks will be small (on the order of 2 to 5) and the number of pages to be broadcast will be substantially larger, so that unused slots (if any) will be only a small fraction of the total number of slots; also, the relative frequencies can be adjusted slightly to reduce the number of unused slots, if necessary.

The disk model, while being fairly simple, allows for the creation of broadcast programs that can be fine-tuned to support a particular access probability distribution. There are three inter-related types of knobs that can be turned to vary the shape of the broadcast. First, the number of disks (num_disks) determines the number of different frequencies with which pages will be broadcast. Then, for each disk, the number of pages per disk, and its relative frequency of broadcast ($rel_freq(i)$) determine the size of the broadcast, and hence the arrival rate (in real, rather than relative time) for pages on each disk. For example, adding a page to a fast disk can significantly increase the delay for pages on the slower disks. Intuitively, we expect that fast disks will be configured to have many fewer pages than the slower disks, although our model does not enforce this constraint.

Recall that the only constraint on the relative broadcast frequencies of the disks is that they be expressed as positive integers. Thus, it is possible to have arbitrarily fine distinctions in broadcasts such as a disk that rotates 141 times for every 98 times a slower disk rotates. However, this ratio results in a broadcast that has a very long period (i.e., nearly 14,000 rotations of the fast disk). Furthermore, this requires that the slower disk be of a size that can be split into 141 fairly equal chunks. In addition, it is unlikely that such fine tuning will produce any significant performance benefit (i.e., compared to a 3 to 2 ratio). Therefore, in practice, relative frequencies should be chosen with care and when possible, approximated to simpler ratios.

While the algorithm specified above generates broadcast programs with the properties that we desire, it does not help in the selection of the various parameter values that shape the broadcast. The automatic determination of these parameters for a given access probability distribution is a very interesting optimization problem, and is one focus of our on-going work. This issue is beyond the scope of the current paper, however. In this paper we focus on examining

the basic properties of this new paradigm of broadcast disks. The broadcast disk changes many basic assumptions on which traditional pull-based memory hierarchies are founded. As a result, it is imperative to first develop an understanding of the fundamental tradeoffs that affect the performance of a broadcast system. The performance study described in Section 5 presents an initial investigation of these issues.

3 CLIENT CACHE MANAGEMENT

The shared nature of the broadcast disk, while in principle allowing for nearly unlimited scalability, in fact gives rise to a fundamental tradeoff: *tuning the performance of the broadcast is a zero-sum game*; improving the broadcast for any one access probability distribution will hurt the performance of clients with different access distributions. The way out of this dilemma is to exploit the local memory and/or disk of the client machines to cache pages obtained from the broadcast. This observation leads to a novel and important result of this work: namely, that the introduction of broadcast *fundamentally changes the role of client caching* in a client-server information system. In traditional, pull-based systems clients cache their *hottest* data (i.e., the items that they are most likely to access in the future). In the push-based environment, this use of the cache can lead to poor performance if the server's broadcast is poorly matched to the client's page access distribution. This difference arises because of the serial nature of the broadcast disk — broadcast pages are *not* all equidistant from the client.

If the server can tailor the broadcast program to the needs of a particular client, then the client can simply cache its hottest pages. Once the client has loaded the hottest pages in its cache, then the server can place those pages on a slower spinning disk. This frees up valuable space in the fastest spinning disks for additional pages. In general, however, there are several factors that could cause the server's broadcast to be sub-optimal for a particular client:

- The access distribution that the client gives the server may be inaccurate.
- A client's access distribution may change over time.
- The server may give higher priority to the needs of other clients with different access distributions.

- The server may have to average its broadcast over the needs of a large client population. Such a broadcast program is likely to be sub-optimal from the point of view of any one client.

For these reasons, in a push-based system clients must use their cache not to store simply their hottest pages, but rather, to store *those pages for which the local probability of access is significantly greater than the page's frequency of broadcast*. For example, if there is a page P that is accessed frequently only by client C and no other clients, then that page is likely to be broadcast on a slow disk. To avoid long waits for the page, client C must keep page P cached locally. In contrast, a page Q that is accessed frequently by most clients (including client C), will be broadcast on a very fast disk, reducing the value of caching it.

The above argument leads to the need for *cost-based page replacement*. That is, the cost of obtaining a page on a cache miss must be accounted for during page replacement decisions. A standard page replacement policy tries to replace the cache-resident page with the lowest probability of access (e.g., this is what LRU tries to approximate). It can be shown that under certain assumptions, an optimal replacement strategy is one that replaces the cache-resident page having the lowest ratio between its probability of access (P) and its frequency of broadcast (X). We refer to this ratio (P/X) as \mathcal{PIX} (P Inverse X). As an example of the use of \mathcal{PIX} , consider two pages. One page is accessed 1% of the time at a particular client and is also broadcast 1% of the time. A second page is accessed only 0.5% of the time at the client, but is broadcast only 0.1% of the time. In this example, the former page has a lower \mathcal{PIX} value than the latter. As a result, a page replacement policy based on \mathcal{PIX} would replace the first page in favor of the second, even though the first page is accessed twice as frequently.

While \mathcal{PIX} can be shown to be an optimal policy under certain conditions, it is not a practical policy to implement because it requires: 1) perfect knowledge of access probabilities and 2) comparison of \mathcal{PIX} values for *all* cache-resident pages at page replacement time. For this reason we have investigated implementable cost-based algorithms that are intended to approximate the performance of \mathcal{PIX} . One such algorithm, adds frequency of broadcast to an LRU-style policy. This new policy is called \mathcal{LIX} and is described and analyzed in Section 5.4.

4 MODELING THE BROADCAST ENVIRONMENT

In order to better understand the properties of broadcast program generation and client cache management we have constructed a simulation model of the broadcast disk environment. The simulator, which is implemented using CSIM [Schw86], models a single server that continuously broadcasts pages and a single client that continuously accesses pages from the broadcast and from its cache. In the simulator, the client generates requests for *logical* pages. These logical pages are then mapped to the *physical* pages that are broadcast by the server.

The mapping of *logical* pages to *physical* pages allows the server broadcast to be varied with respect to the client workload. This flexibility allows the simulator to model the impact of a large client population on the performance of a single client, without having to model the other clients. For example, having the client access only a subset of the pages models the fact that the server is broadcasting pages for other clients as well. Furthermore, by systematically perturbing the client's page access probabilities with respect to the server's expectation of those probabilities, we are able to vary the degree to which the server broadcast favors the particular client that we are modeling. The simulation model is described in the following sections.

4.1 Client Execution Model

The parameters that describe the operation of the client are shown in Table 2. The simulator measures performance in logical time units called *broadcast units*. A broadcast unit is the time required to broadcast a single page. In general, the results obtained from the simulator are valid across many possible broadcast media. The actual response times experienced for a given medium will depend on the amount of real time required to broadcast a page.

<i>CacheSize</i>	Client cache size (in pages)
<i>ThinkTime</i>	Time between client page accesses (in broadcast units)
<i>AccessRange</i>	# of pages in range accessed by client
θ	Zipf distribution parameter
<i>RegionSize</i>	# of pages per region for Zipf distribution

Table 2 Client Parameter Description

The client runs a continuous loop that randomly requests a page according to a specified distribution. The client has a cache that can hold $CacheSize$ pages. If the requested page is not cache-resident, then the client waits for the page to arrive on the broadcast and then brings the requested page into its cache. Client cache management is done similarly to buffer management in a traditional system; if all cache slots are occupied, then a page replacement policy is used to choose a victim for replacement.² Once the requested page is cache resident, the client waits $ThinkTime$ broadcast units of time and then makes the next request. The $ThinkTime$ parameter allows the cost of client processing relative to page broadcast time to be adjusted, thus it can be used to model workload processing as well as the relative speeds of the CPU and the broadcast medium.

The client chooses the pages to access from the range 1 to $AccessRange$, which can be a subset of the pages that are broadcast. All pages outside of this range have a zero probability of access at the client. Within the range the page access probabilities follow a Zipf distribution [Knut81, Gray94], with page 1 being the most frequently accessed, and page $AccessRange$ being the least frequently accessed. The Zipf distribution is typically used to model non-uniform access patterns. It produces access patterns that become increasingly skewed as θ increases — the probability of accessing any page numbered i is proportional to $(1/i)^\theta$. Similar to earlier models of skewed access [Dan90], we partition the pages into regions of $RegionSize$ pages each, such that the probability of accessing any page within a region is uniform; the Zipf distribution is applied to these regions. Regions do not overlap so there are $AccessRange/RegionSize$ regions.

4.2 Server Execution Model

The parameters that describe the operation of the server are shown in Table 3. The server broadcasts pages in the range of 1 to $ServerDBSize$, where $ServerDBSize \geq AccessRange$. These pages are interleaved into a broadcast program according to the algorithm described in Section 2. This program is broadcast repeatedly by the server. The structure of the broadcast program is described by several parameters. $NumDisks$ is the number of levels (i.e., “disks”) in the multi-disk program. By convention disks are numbered from 1 (fastest) to $N=NumDisks$ (slowest). $DiskSize_i$, $i \in [1..N]$, is the number of pages assigned to each disk i . Each page is broadcast on exactly one disk, so the sum of $DiskSize_i$ over all i is equal to the $ServerDBSize$.

²We discuss the performance of various replacement policies in Section 5.

$ServerDBSize$	Number of distinct pages to be broadcast
$NumDisks$	Number of disks
$DiskSize_i$	Size of disk i (in pages)
Δ	Broadcast shape parameter
$Offset$	Offset from default client access
$Noise$	% workload deviation

Table 3 Server Parameter Description

In addition to the size and number of disks, the model must also capture their relative speeds. As described in Section 2, the relative speeds of the various disks can be any positive integers. In order to make experimentation tractable, however, we introduce a parameter called Δ , which determines the relative frequencies of the disks in a restricted manner. Using Δ , the frequency of broadcast $rel_freq(i)$ of each disk i , can be computed relative to $rel_freq(N)$, the broadcast frequency of the slowest disk (disk N) as follows:

$$\frac{rel_freq(i)}{rel_freq(N)} = (N - i)\Delta + 1$$

When Δ is zero, the broadcast is flat: all disks spin at the same speed. As Δ is increased, the speed differentials among the disks increase. For example, for a 3-disk broadcast, when $\Delta = 1$, disk 1 spins three times as fast as disk 3, while disk 2 spins twice as fast as disk 3. When $\Delta = 3$, the relative speeds are 7, 4, and 1 for disks 1, 2, and 3 respectively. It is important to note that Δ is used in the study only to organize the space of disk configurations that we examine. It is not part of the disk model as described in Section 2.

The remaining two parameters, $Offset$ and $Noise$, are used to modify the mapping between the *logical* pages requested by the client and the *physical* pages broadcast by the server. When $Offset$ and $Noise$ are both set to zero, then the logical to physical mapping is simply the identity function. In this case, the $DiskSize_1$ hottest pages from the client's perspective (i.e., 1 to $DiskSize_1$) are placed on disk 1, the next $DiskSize_2$ hottest pages are placed on disk 2, etc. However, as discussed in Section 3, this mapping may be sub-optimal due to client caching. Some client cache management policies tend to fix certain pages in the client's buffer, and thus, those pages do not need to be broadcast frequently. In such cases, the best broadcast can be obtained by shifting the hottest pages from the fastest disk to the slowest. $Offset$ is the number of pages that are shifted in this manner. An offset of K shifts the access pattern by K

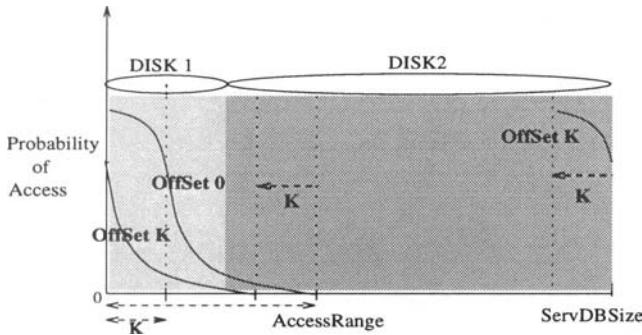


Figure 4 Using Offset to vary client access

pages, pushing the K hottest pages to the end of the slowest disk and bringing colder pages to the faster disks. The use of offset is demonstrated in Figure 4.

In contrast to *Offset*, which is used to provide a better broadcast for the client, the parameter *Noise* is used to introduce disagreement between the needs of the client and the broadcast program generated by the server. As described in Section 2, such disagreement can arise in many ways, including dynamic client access patterns and conflicting access requirements among a population of clients. *Noise* determines the percentage of pages for which there may be a mismatch between the client and the server. That is, with probability *Noise* the mapping of a page may be switched with a different page.

The generation of the server broadcast program works as follows. First, the mapping from logical to physical pages is generated as the identity function. Second, this mapping is shifted by *Offset* pages as described above. Third, for each page in the mapping, a coin weighted by *Noise* is tossed. If based on the coin toss, a page i is selected to be swapped then a disk d is uniformly chosen to be its new destination.³ To make way for i , an existing page j on d is chosen, and i and j exchange mappings.

5 EXPERIMENTS AND RESULTS

In this section, we use the simulation model to explore the performance characteristics of the broadcast disk. The primary performance metric employed

³Note that a page may be swapped with a page on its own disk. Such a swap does not affect performance in the steady state, so *Noise* represents the upper limit on the number of changes.

<i>ThinkTime</i>	2.0
<i>ServerDBSize</i>	5000
<i>AccessRange</i>	1000
<i>CacheSize</i>	50(5%), 250(25%), 500(50%)
Δ	1, 2, . . . 7
θ	0.95
<i>Offset</i>	0, <i>CacheSize</i>
<i>Noise</i>	0%, 15%, 30%, 45%, 60%, 75%
<i>RegionSize</i>	50

Table 4 Parameter Settings

in this study is the response time at the client, measured in *broadcast units*. The server database size (*ServerDBSize*) was 5000 pages, and the client access range *AccessRange* was 1000 pages. The client cache size was varied from 1 (i.e., no caching) to 500 (i.e., half of the access range). We studied several different two-disk and three-disk configurations of broadcast programs. All of the results presented in the paper were obtained once the client performance reached steady state. The cache warm-up effects were eliminated by beginning our measurements only after the cache was full, and then running the experiment for 15,000 or more client page requests (until steady state).

Table 4 shows the parameter settings used in these experiments. It should be noted that the results described in this section are a very small subset of the results that have been obtained. These results have been chosen because they demonstrate many of the unique performance aspects and tradeoffs of the broadcast disk environment, and because they identify important areas for future study.

5.1 Experiment 1: No Caching, 0% Noise

The first set of results examine the case where the client performs no caching (i.e., it has a cache size of one page). Figure 5 shows the client response time vs. Δ for a number of two and three disk configurations. In this graph, *Noise* is set to 0%, meaning that the server is providing preferential treatment to the client (i.e., it is giving highest priority to this client's pages). As Δ is increased along the x-axis of the figure, the skew in the relative speeds of the disks is increased (as described in Section 4). As shown in the figure, the general trend in these cases is that response time improves with increasing disk skew. When $\Delta = 0$, the broadcast is flat (i.e., all disks rotate at the same speed). In this

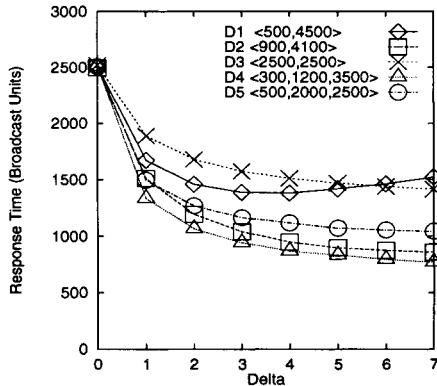


Figure 5 Client Performance, Cache Size = 1, Noise = 0%

case, as would be expected, all disks result in a response time of 2500 pages — half the *ServerDBSize*. As Δ is increased, all of the disk configurations shown provide an improvement over the flat disk. The degree of improvement begins to flatten for most configurations around a Δ value of 3 or 4.

Turning to the various disk configurations, we first examine the two-disk configurations: D1, D2, and D3. For D1, 500 pages fit on the first (i.e., fastest) disk. Because *Noise* and *Offset* are both zero, the hottest half of the client's access range is on the fast disk, and the colder half is on the slower disk. As Δ is increased, performance improves until $\Delta = 3$ because the hotter pages are brought closer. Beyond this point, the degradation caused by the access to the slow pages (which get pushed further away) begins to hurt performance. In contrast, D2, which places 90% of the client access range (900 pages) on the fast disk improves with increasing Δ for all values of Δ in this experiment. Because most of the accessed pages are on the fast disk, increasing Δ pushes the colder and unused pages further away, allowing the accessed pages to arrive more frequently. At some point, however, the penalty for slowing down the 10% will become so great that the curve will turn up again as in the previous case. The final two-disk configuration, D3, has equal sized disks. Although all of the accessed data fits on the fast disk, the fast disk also includes many unaccessed pages. The size of the fast disk causes the frequencies of the pages on this disk to be lower than the frequencies of pages on the fast disks of D2 and D1 at corresponding values of Δ . As a result, D3 has the worst performance of the two-disk configurations for most of the Δ values shown.

Turning to the three-disk configurations: D4 and D5, it can be seen that configuration D4, which has a fast disk of 300 pages has the best performance across the entire range. At a Δ of 7, its response time is only one-third of the flat-disk

response time. D5, which is simply the D3 disk with its first disk split across two disks, performs better than its two-disk counterpart. The extra level of disk makes it easier to match the broadcast program to the client's needs. However, note that response time for D5 is typically higher than the two-disk D2, and thus, the extra disk level does not necessarily ensure better performance.

5.2 Experiment 2: Noise and No Caching

In the previous experiment, the broadcast program generation was done giving our client's access pattern the highest priority. In this experiment we examine the performance of the broadcast disk as the server shifts its priority away from this client (i.e., as *Noise* is increased). These results are shown in Figures 6 and 7, which show how the client performs in the presence of increasing noise for configurations D3 (two-disks) and D4 (three-disks) from Figure 5 respectively. As expected, performance suffers for both configurations as the *Noise* is increased; as the mismatch between the broadcast and the client's needs increases, the skew in disk speeds starts to hurt performance. Ultimately, if the mismatch becomes great enough, the multi-disk approach can have worse performance than the flat disk. This is shown in the performance disk of D3 (Figure 6). This susceptibility to a broadcast mismatch is to be expected, as the client accesses all of its data from the broadcast channel. Thus, it is clear that if a client does not have a cache, the broadcast must be well suited for that client's access demands in order to gain the benefits of the multi-disk approach.

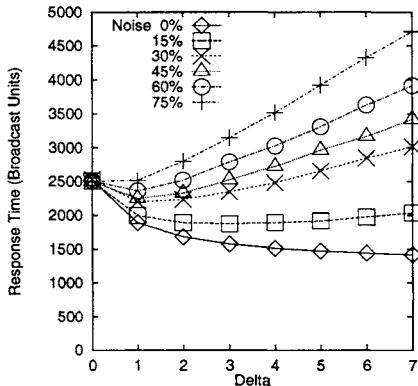


Figure 6 Noise Sensitivity - Disk D3($<2500, 2500>$), CacheSize = 1

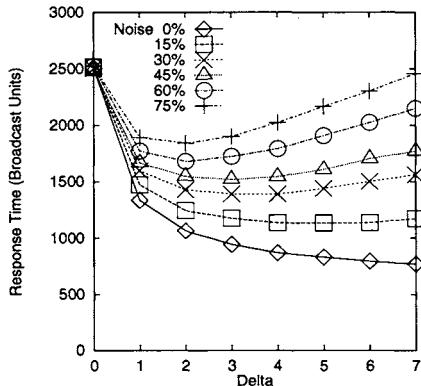


Figure 7 Noise Sensitivity - Disk D4($<300, 1200, 3500>$), CacheSize = 1

5.3 Experiment 3: Caching and Noise

The previous experiments showed that even in the absence of caching, a multi-level disk scheme can improve performance, but that without a cache, performance can suffer if the broadcast program is poorly suited to the client's access demands. In this experiment we introduce the use of a client cache, to reduce the expected page access delay and to increase the client's tolerance to mismatches in the broadcast program. We use an idealized page replacement policy called \mathcal{P} , which keeps the pages with the highest probability of access in the cache. \mathcal{P} , however, it is not an implementable policy, as it requires perfect knowledge of access probabilities and a great deal of local computation.⁴ We use \mathcal{P} , therefore, to gain an understanding of the performance in a simplified setting and as a point-of-reference for other (implementable) policies.

In steady state, a client using the \mathcal{P} replacement policy will have the $CacheSize$ hottest pages in its cache. Consequently, broadcasting these cache-resident pages on the fastest disk is a waste of bandwidth. Thus, as stated in Section 4.2, the best broadcast program will be obtained by shifting the $CacheSize$ hottest pages from the fastest disk to the slowest. Such shifting is accomplished in the simulation model by setting $Offset = CacheSize$.⁵ Given this $Offset$, we now examine the effectiveness of a cache (using the idealized \mathcal{P} replacement policy) in allowing a client to tolerate *Noise* in the broadcast. Figure 8 shows the impact of increasing *Noise* on the performance of the three-disk configuration D4 as Δ is varied. In the case shown, $CacheSize$ and $Offset$ are both set to 500 pages. Comparing these results with the results obtained in the no caching case (see Figure 7), we see that although as expected the cache greatly improves performance in an absolute sense, surprisingly, the cache-based numbers are if anything, somewhat *more* sensitive to the degree of *Noise* than the non-caching numbers. For example, in the caching case, when Δ is greater than 2, the higher degrees of noise have multi-disk performance that is worse than the flat disk performance, whereas this crossover did not occur for similar Δ values in the non-caching case. The reason for this additional sensitivity is that when *Noise* is low and $Offset = CacheSize$, \mathcal{P} does exactly what it should do — it caches those hot pages that have been placed on the slowest disk, and it obtains the remainder of the hottest pages from the fastest disk. However, as noise increases, \mathcal{P} caches the same pages regardless of what disk they are stored on. Caching a page that is stored on the fastest disk is often not a good use of the cache, as those pages are broadcast frequently. As noise increases, \mathcal{P} 's cache hit rate remains the same, but its cache *misses* become more expensive, as it

⁴It is trivial to implement \mathcal{P} in the simulator, as the probability of each page is known from the client access distribution.

⁵The impact the $Offset$ parameter is discussed in more detail in [Acha94].

has to retrieve some pages from the slower disks. These expensive cache misses are the cause of \mathcal{P} 's sensitivity to *Noise*.

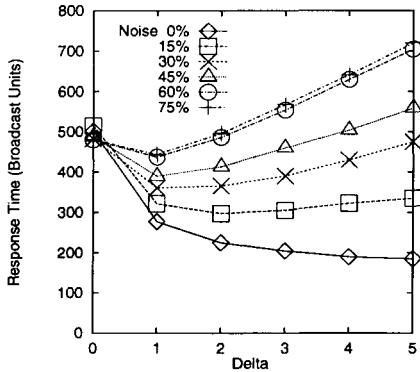


Figure 8 Noise Sensitivity - Disk D4, CacheSize = 500, Rep. Policy = \mathcal{P}

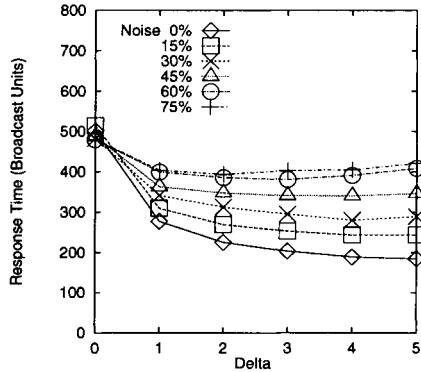


Figure 9 Noise Sensitivity - Disk D4, CacheSize = 500, Rep. Policy = $\mathcal{P}IX$

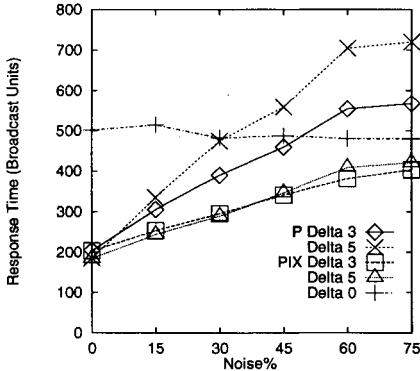


Figure 10 \mathcal{P} vs. $\mathcal{P}IX$ With Varying Noise, Disk D4, CacheSize = 500

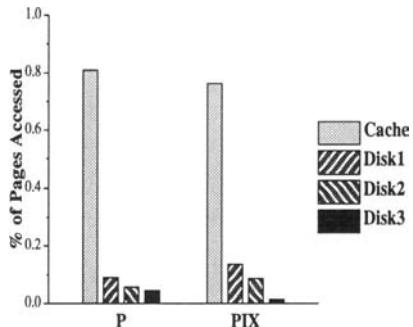


Figure 11 Access Locations for \mathcal{P} vs. $\mathcal{P}IX$, Disk D4, CacheSize = 500, Noise = 30%, Δ = 3

5.4 Cost Based Replacement Algorithms

In the previous section, it was shown that while standard caching can help improve performance in a multi-disk broadcast environment, it can actually increase the client's sensitivity to *Noise*. Recall that *Noise* represents the degree to which the server broadcast deviates from what is best for a particular client. It is likely, therefore, that some type of "noise" will be present in any application in which there are multiple clients that access the broadcast disk. The \mathcal{P} replacement policy was found to be sensitive to noise because it ignored the *cost*

of re-acquiring a page when choosing a victim for replacement. To address this deficiency, we examine a second idealized algorithm called \mathcal{PITX} , that extends \mathcal{P} with the notion of cost. As stated in Section 3, \mathcal{PITX} always replaces the page with the lowest ratio of access probability to broadcast frequency. Thus, the cost of re-accessing a replaced page is factored into the replacement decision.

Experiment 4: \mathcal{PITX} and Noise

Figure 9 shows the response time of the client using \mathcal{PITX} for the same case that the previous experiment showed for \mathcal{P} (see Figure 8). Comparing the two figures it can be seen that \mathcal{PITX} is much more successful at insulating the client response time from effects of *Noise*. Of course, an increase in *Noise* still results in a degradation of performance; this is to be expected. However, unlike the case with \mathcal{P} , using \mathcal{PITX} the performance of the client remains better than the corresponding flat disk performance for all values of *Noise* and Δ in this experiment. Under \mathcal{PITX} , the performance of the client for a given *Noise* value remains stable as Δ is increased beyond a certain point. In contrast, under \mathcal{P} , in the presence of noise, the performance of the client quickly degrades as Δ is increased beyond a value of 1 or 2. This experiment demonstrates the potential of cost-based replacement for making the broadcast disk practical for a wider range of applications.

Figure 10 shows results from the same set of experiments in a slightly different light. In this figure, the effect of increasing noise on the response time of the two algorithms for $\Delta = 3$ and $\Delta = 5$ is shown. The performance for the flat disk ($\Delta = 0$) is given as a baseline.⁶ Note that \mathcal{P} degrades faster than \mathcal{PITX} and eventually becomes worse than the flat disk at around $Noise = 45\%$. \mathcal{PITX} rises gradually and manages to perform better than the flat disk within these parameters. Also, notice how \mathcal{P} 's performance degrades for $\Delta = 5$; unlike \mathcal{PITX} it fails to adapt the cache contents with increasing differences in disk speeds.

The performance differences between the two algorithms result from the differences in the places from which they obtain their pages (as shown in Figure 11 for the case where $Noise = 30\%$). It is interesting to note that \mathcal{PITX} has a lower cache hit rate than \mathcal{P} . A lower cache hit rate does not mean lower response times in broadcast environments; the key is to reduce expected latency by caching important pages that reside on the slower disks. \mathcal{PITX} gets fewer pages from the slowest disk than does \mathcal{P} , even though it gets more pages from

⁶Note that at $\Delta = 0$ (i.e., a flat disk), \mathcal{P} and \mathcal{PITX} are identical, as all pages are broadcast at the same frequency.

the first and second disks. In this case, this tradeoff results in a net performance win.

5.5 Implementing Cost Based Policies

The previous sections have shown that multi-disk broadcast environments have special characteristics which when correctly exploited can result in significant performance gains. They also demonstrated the need for cost-based page replacement and examined a cost-based algorithm ($\mathcal{P}\mathcal{IX}$). Unfortunately, like \mathcal{P} , the policy on which it is based, $\mathcal{P}\mathcal{IX}$ is not an implementable algorithm. However, based on the insight that we gained by examining \mathcal{P} and $\mathcal{P}\mathcal{IX}$ we have designed and implemented an approximation of $\mathcal{P}\mathcal{IX}$, which we call $\mathcal{L}\mathcal{IX}$.

$\mathcal{L}\mathcal{IX}$ is a modification of LRU that takes into account the broadcast frequency. LRU maintains the cache as a single linked-list of pages. When a page in the cache is accessed, it is moved to the top of the list. On a cache miss, the page at the end of the chain is chosen for replacement.

In contrast, $\mathcal{L}\mathcal{IX}$ maintains a number of smaller chains: one corresponding to each disk of the broadcast ($\mathcal{L}\mathcal{IX}$ reduces to LRU if the broadcast uses a single flat disk). A page always enters the chain corresponding to the disk in which it is broadcast. Like LRU, when a page is hit, it is moved to the top of *its own* chain. When a new page enters the cache, $\mathcal{L}\mathcal{IX}$ evaluates a *lix* value (see next paragraph) only for the page at the bottom of each chain. The page with the smallest *lix* value is ejected, and the new page is inserted in the appropriate queue. Because this queue might be different than the queue from which the slot was recovered, the chains do not have fixed sizes. Rather, they dynamically shrink or grow depending on the access pattern at that time. $\mathcal{L}\mathcal{IX}$ performs a constant number of operations per page replacement (proportional to the number of disks) which is the same order as that of LRU. Figure 12 shows an example of $\mathcal{L}\mathcal{IX}$ for a two-disk broadcast. Pages g and k are at the bottom of each chain. Since g has a lower *lix* value it is chosen as the victim. The new page z , being picked from the second disk, joins Disk2Q. Note the relative changes in the sizes of both the queues.

In order to compute the *lix* value, the algorithm maintains two data items per cached page i : a running probability estimate (p_i) and the time of the most recent access to the page (t_i). When the page i enters a chain, p_i is initially set to zero and t_i is set to the current time. If i is hit again, the new probability estimate for i is calculated using the following formula:

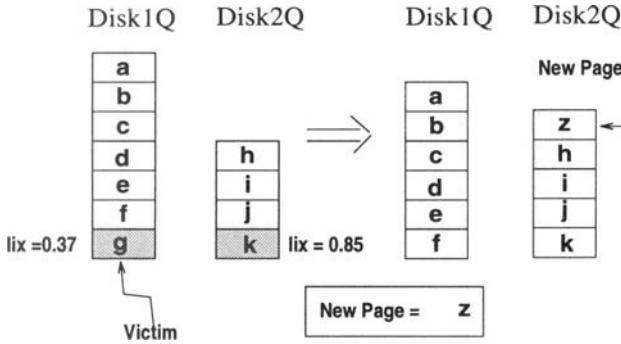


Figure 12 Page replacement in LIX

$$p_i = \Lambda / (\text{CurrentTime} - t_i) + (1 - \Lambda) p_i$$

t_i is then subsequently updated to the current time. Λ is a constant used to appropriately weigh the most recent access with respect to the running probability estimate; in these experiments, it is set to 0.25. This formula is evaluated for the least recently used pages of each chain to estimate their current probability of access. This value is then divided by the frequency for the page (which is known exactly) to get the *lix* value. The page with the lowest *lix* value is ejected from the cache. LIX is a simple approximation of PIX , yet in spite of this, it performs surprisingly well (as is shown below). Better approximations of PIX , however, might be developed using some of the recently proposed improvements to LRU like 2Q[John94] or LRU-k[ONei93].

Experiment 5: LIX vs. LRU

The next set of experiments are similar to those for \mathcal{P} and PIX and compare LIX and LRU. However, unlike \mathcal{P} , the best performance for LRU isn't at an offset equal to the cache size. Being only an approximation of \mathcal{P} , LRU isn't able to retain all of the hot pages that are stored on the slowest disk and thus, it performs poorly at this offset. For similar reasons, LIX also does not perform best at this offset. As a result, we also compared the performance of LIX and LRU to a modified version of LIX called \mathcal{L} . \mathcal{L} behaves exactly like LIX except that it assumes the same value of frequency for all pages. Thus, the difference in performance between \mathcal{L} and LRU indicates how much better (or worse) an approximation of probability \mathcal{L} provides over LRU, and the performance difference between LIX and \mathcal{L} shows the role that broadcast frequency plays (if any) in the performance of the caching strategies.

Figure 13 shows the performance of the three algorithms for different values of Δ . These results show the sensitivity of the algorithms to changing Δ for the same case as in Figure 10 (i.e., $Offset=CacheSize=500$), with $Noise$ set to 30%. In this experiment, LRU performs worst and consistently degrades as Δ is increased. \mathcal{L} does better at $\Delta = 1$ but then degrades. The benefits of using frequency are apparent from the difference in response time between \mathcal{LIX} and \mathcal{L} . The response time of \mathcal{LIX} is only between 25% to 50% that of \mathcal{L} . The solid line on the bottom of the graph shows how the ideal policy (\mathcal{PILX}) performs; it does better than \mathcal{LIX} , but only by a small margin. The factors underlying these results can be seen in Figure 14, which shows the distribution of page access locations for the results of Figure 13 when Δ is set to 3. In this case, \mathcal{LIX} obtains a much smaller proportion of its pages from the slowest disk than do the other algorithms. Given that the algorithms have roughly similar cache hit rates, the differences in the distributions of access to the different disks is what drives the performance results here.

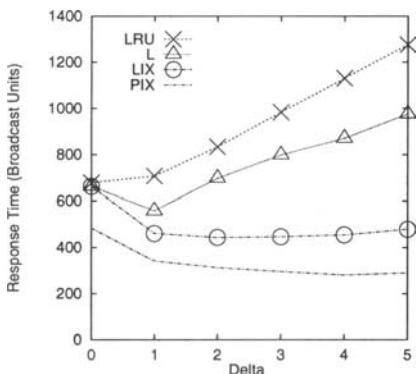


Figure 13 Sensitivity to Δ - Disk D4, CacheSize = 500, Noise = 30%

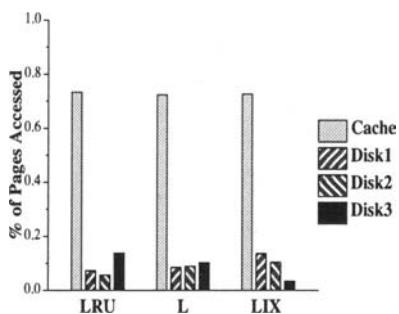


Figure 14 Page Access Locations - Disk D4, CacheSize = 500, Noise = 30%, $\Delta = 3$

Figure 15 shows the performance of the three algorithms with varying $Noise$ with $\Delta = 3$. In this case, it can be seen that \mathcal{L} performs only somewhat better than LRU. The performance of \mathcal{LIX} degrades with noise as expected, but it outperforms both \mathcal{L} and LRU across the entire region of $Noise$ values. These results demonstrate that the frequency-based heuristic of \mathcal{LIX} can provide improved performance in the presence of noise.

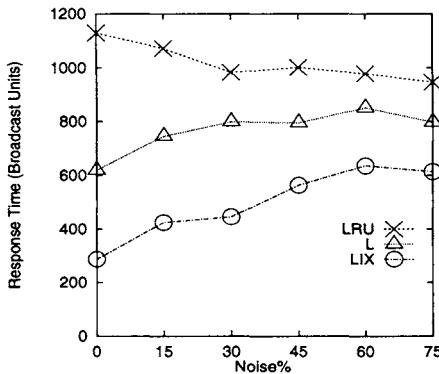


Figure 15 Noise Sensitivity - Disk D4, $\text{CacheSize} = 500$, $\Delta = 3$

6 PREVIOUS WORK

While no previous work has addressed multilevel broadcast disks and the related cache management techniques described in this paper, several projects in mobile databases and other areas have performed related work. As stated previously, the notion of using a repetitive broadcast medium for database storage and query processing was investigated in the Datacycle project at Bellcore [Herm87, Bowe92]. Datacycle was intended to exploit high bandwidth, optical communication technology and employed custom VLSI data filters for performing associative searches and continuous queries on the broadcast data. Datacycle broadcast data using a flat disk approach and so the project did not address the multi-level disk issues that we have addressed in this paper. However, the Datacycle project did provide an optimistic form of transaction management which employed an “upstream network” that allowed clients to communicate with the host. We intend to investigate issues raised by allowing such upstream communication through low-bandwidth links as part of our ongoing work. An early effort in information broadcasting, the Boston Community Information System (BCIS) is described in [Giff90]. BCIS broadcast newspapers and information over an FM channel to clients with personal computers specially equipped with radio receivers. Like Datacycle, they too used a flat disk approach.

More recently, the mobile computing group at Rutgers has investigated techniques for indexing broadcast data [Imie94b]. The main thrust of this work has been to investigate ways to reduce power consumption at the clients in order to preserve battery life. Some of the indexing techniques described in

[Imie94b] involve the interleaving of index information with data, which forms a restricted type of multilevel disk. However, this work did not investigate the notion of replicating the actual data to support non-uniform access patterns and did not investigate the impact of caching. In our current work we have assumed a fixed broadcast program, so that indexing was not needed. However, we are currently investigating ways to integrate indexes with the multilevel disk in order to support broadcast program changes due to client population changes and updates. Caching in a mobile environment has been considered in [Barb94]. However, their model was different in that it considered volatile data and clients who could be inactive (and/or disconnected) over long periods of time. Thus, the focus of both broadcasting and caching in this work was to efficiently detect and avoid access to stale data in the cache. Very recently, another approach to broadcasting data for video on demand has been taken in [Vish94]. The technique, called pyramid broadcasting, splits an object (e.g., a video clip) into a number of segments of increasing sizes. To minimize latency the first segment is broadcast more frequently than the rest. While similar in spirit, a key difference is that the data needed by the client is known *a priori* once the first segment (the choice of movie) is decided upon and thus, they do not need to address the issues related to caching dealt in this paper.

The issues that arise due to our use of a broadcast medium as a multi-level device also arise in other, more traditional types of complex memory hierarchies. The need for cost-based caching and page replacement has been recognized in other domains in which there is a wide variation in the cost of obtaining data from different levels of the storage hierarchy. For example, [Anto93] describes the need for considering “cost of acquisition” for page replacement in deep-store file systems involving tertiary mass storage. This issue is also addressed for client-server database systems in which a global memory hierarchy is created by allowing clients to obtain data from other clients that have that data cached [Fran92]. In this work, server page replacement policies are modified to favor pages that are not cached at clients, as they must be obtained from disk, which is more expensive. Recently, a technique called “Disk-Directed I/O” has been proposed for High Performance Computing applications [Kotz94]. Disk-Directed I/O sends large requests to I/O devices and allows the devices to fulfill the requests in a piecemeal fashion in an order that improves the disk bandwidth. Finally, the tradeoff between replication to support access to hot data while making cold data more expensive to access has been investigated for magnetic disks [Akyu92].

7 SUMMARY AND FUTURE WORK

In this paper, we have described our design of a multilevel broadcast disk and cache management policies for this style of memory. We believe that this approach to data management is highly applicable to asymmetric network environments such as those that will naturally occur in the NII as well as many other modern data delivery systems. We have demonstrated that in designing such disks, the broadcast program and the caching policy must be considered together.

It has been shown that there are cases in which the performance of both two and three level disks can outperform a flat broadcast even when there is no caching. We have argued that our scheme for interleaving the data is desirable because it provides a uniform expected latency.

We have further shown that introducing a cache can provide an advantage by smoothing out disagreement between the broadcast and the client access patterns. The cache gives the clients a way to hoard their hottest pages regardless of how frequently they are broadcast. However, doing page replacement solely on probability of access can actually increase a client's sensitivity to the server's broadcast.

We then introduced a caching policy that also took into account the broadcast frequency during replacement. We showed that this not only improves client performance but also shields it from vagaries of the server broadcast. This is because the clients can cache items that are relatively hot and reside on a slow disk and thus, avoid paying high cache miss penalties.

Finally, we demonstrated a straightforward implementation technique that approximates our ideal cost-based caching scheme. This technique is a modification of LRU which accounts for the differences in broadcast frequency of the data.

We believe that this study while interesting and useful in its own right, is just the tip of the iceberg. There are many other opportunities that can be exploited in future work. Here, we have only considered the static read-only case. How would our results have to change if we allowed the broadcast data to change from cycle to cycle? What kinds of changes would be allowed in order to keep the scheme manageable, and what kinds of indexing would be needed to allow the client to make intelligent decisions about the cost of retrieving a data item from the broadcast?

We are currently investigating how prefetching could be introduced into the present scheme. The client cache manager would use the broadcast as a way to opportunistically increase the temperature of its cache. We are exploring new cache management metrics for deciding when to prefetch a page.

We would also like to provide more guidance to a user who wants to configure a broadcast. We have experimental results to show that good things can happen, but given a workload, we would like to have concrete design principles for deciding how many disks to use, what the best relative spinning speeds should be, and how to segment the client access range across these disks. We are pursuing an analytic model to address this.

Finally, once the basic design parameters for broadcast disks of this kind are well-understood, work is needed to develop query processing strategies that would exploit this type of media.

Acknowledgements

The authors would like to thank M. Ranganathan for providing them with a number of important insights into the properties of broadcast programs. Franklin's research was supported in part by a grant from the University of Maryland General Research Board, NSF grants IRI-9409575 and IRI-9501353 and by a gift from Intel Corporation. Acharya and Zdonik were supported in part by ONR grant number N00014-91-J-4085 under ARPA order number 8220 and by a gift from Intel Corporation. Acharya was also supported in part by a fellowship from IBM Corporation.

REFERENCES

- [Acha94] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments", Tech. Report CS-94-43, Brown Univ.; Tech. Report CS-TR-3369, Univ. of Maryland, Oct. 1994.
- [Akyu92] S. Akyurek, K. Salem, "Placing Replicated Data to Reduce Seek Delays" *Proc. USENIX File System Conf.*, May 1992.
- [Anto93] C. Antonelli, P. Honeyman, "Integrating Mass Storage and File Systems", *Proc. 12th IEEE Symp on Mass Storage Sys.*, 1993.
- [Barb94] D. Barbara, T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments", *Proc. ACM SIGMOD Conf.*, May, 1993.
- [Bowe92] T. Bowen, et al. "The Datacycle Architecture" *CACM* 35,(12), Dec., 1992.
- [Dan90] A. Dan, D. M. Dias, P. Yu, "The Effect of Skewed Access on Buffer Hits and Data Contention in a Data Sharing Environment", *Proc. 16th VLDB Conf.*, Aug., 1990.
- [Fran92] M. Franklin, M. Carey, M. Livny, "Global Memory Management in Client-Server DBMS Architectures", *Proc. 18th VLDB Conf.*, Aug., 1992.
- [Giff90] D. Gifford, "Polychannel Systems for Mass Digital Communications", *CACM*, 33(2), Feb., 1990.
- [Gray94] J. Gray, et al., "Quickly Generating Billion-Record Synthetic Databases", *Proc. ACM SIGMOD Conf.*, May, 1994.
- [Herm87] G. Herman, G. Gopal, K. Lee, A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems", *Proc. ACM SIGMOD Conf.*, May, 1987.
- [Imie94a] T. Imielinski, B. Badrinath, "Mobile Wireless Computing: Challenges in Data Management", *CACM*, 37(10), Oct., 1994.
- [Imie94b] T. Imielinski, S. Viswanathan, B. Badrinath, "Energy Efficient Indexing on Air" *Proc. ACM SIGMOD Conf.*, May, 1994.
- [John94] T. Johnson, D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm", *Proc. 20th VLDB Conf.*, Sept., 1994.

- [Katz94] R. Katz, "Adaption and Mobility in Wireless Information Systems", *IEEE Personal Comm.*, 1st Quarter, 1994.
- [Knut81] D. Knuth, "The Art of Computer Programming, Vol II", Addison Wesley, 1981.
- [Kotz94] D. Kotz, "Disk-directed I/O for MIMD Multiprocessors", *1st Symposium on OS Design and Implementation*, USENIX, Nov., 1994.
- [ONei93] E. J. O'Neil, P. E. O'Neil, G. Weikum, "The LRU-k Page Replacement Algorithm for Database Disk Buffering", *Proc. ACM SIGMOD Conf.*, May, 1993.
- [Schw86] H. D. Schwetman, "CSIM: A C-based process oriented simulation language", *Proc. 1986 Winter Simulation Conf.*, 1986.
- [Vish94] S. Vishwanath, T. Imielinski, "Pyramid Broadcasting for Video on Demand Service", Rutgers Univ. Tech. Report DCS TR-311, 1994.
- [Zdon94] S. Zdonik, M. Franklin, R. Alonso, S. Acharya, "Are 'Disks in the Air' Just Pie in the Sky?", *IEEE Wkshp on Mobile Comp. Sys. and Applications*, Santa Cruz, CA, Dec., 1994.

APPLICATION DESIGN FOR WIRELESS COMPUTING

Terri Watson

*Department of Computer Science & Engineering,
University of Washington, Seattle, WA 98195
USA*

ABSTRACT

As mobile computing becomes more prevalent, systems and applications must deal with a growing disparity among resource types and availabilities at the user interface device. Network properties, display size, input method, and local storage are some of the more obviously affected characteristics. Although the challenges of effectively managing these resources are partially addressed by existing systems, their solutions are hindered by the system's lack of knowledge about application behavior. This, combined with the narrower application domain commonly used during mobile computing, suggests addressing these concerns from the application level. This paper presents strategies for designing applications for a wireless environment, by reducing the demands placed on the wireless network. A World Wide Web client browser is used to illustrate this design.

1 INTRODUCTION

Today's wireless technologies have latency and bandwidth characteristics that are anachronistic in the context of current wired networks and applications' expectations. Although these limitations are being addressed in successive technologies, the performance disparity between wireless and wired networks is likely to remain. The problems of wireless links, such as high and variable latency, low bandwidth, high cost, and incomplete coverage, argue for a soft-

This research was sponsored by a Presidential Young Investigator Award from the National Science Foundation, with matching funds from the Xerox Corporation.

Copyright ©1994 by IEEE. Reprinted, with permission, from Proceedings of the Workshop on Mobile Computing Systems and Applications, December 1994.

ware architecture that reduces the demands placed on the wireless link, and supports disconnected operation.

Several systems exist that provide various solutions for dealing with network resource scarcity. Coda [10, 6] and Little Work [5] both use caching to permit continued operation during disconnection. RUSE [7] and newer revisions of the ParcTab [1] download code to the mobile device permitting some local interactions. This reduces user-perceived latency and often the amount of data transferred for those operations.

Although techniques for tolerating network constraints (for example, caching and prefetching) are often employed at the system level, their effectiveness is hindered by a lack of information. We chose to attack the problem from a higher level, and present a strategy for designing applications whose data and functional organization provide implicit information about structure and reference patterns to increase the benefits of traditional techniques.

If one examines the use of emerging PDAs and other handheld devices, their utility is derived from a fairly small set of applications. In inventory, medical and other vertical markets, the number of applications is even fewer, and applications are tailored to the device. In light of this, it is reasonable to (re)design wireless applications to perform well in the presence of resource constraints and diversity. This application-level approach directly addresses two goals: first, to exploit an application's knowledge of its own behavior, and second, to specify user interactions in a more abstract manner, to reduce the dependency on particular input and output devices. This paper focuses on the first goal, as it applies to using the network link more effectively.

The remainder of the paper is structured as follows. Section 2 presents a strategy for designing applications that use the network link more effectively. Section 3 briefly describes our wireless platform. Section 4 uses W^* , a World Wide Web client browser, to illustrate the design of a wireless application. Section 5 details our experiences with W^* . Section 6 contains the conclusions and future work.

Technique	Potential Benefits
caching	reduce latency and bandwidth
prefetch	hide latency, reduce burstiness
data encoding	reduce bandwidth and latency
lazy evaluation	reduce bandwidth
futures	hide latency through concurrency
data reduction	reduce bandwidth and latency

Table 1 Communication Optimization Techniques

2 APPLICATION DESIGN FOR A WIRELESS ENVIRONMENT

Central to the design of our mobile software is the concept of a partitioned application. Application functionality is partitioned into components that execute either on the mobile device or a machine on the wired network. This has similarities to a client/server or network agent [2, 1] architecture, but with slightly different characteristics than are typical for either of those models. The interface boundary is chosen based on efficiency and constrained by network connectivity and data or function location. A typical partition might place data gathering and parsing functions in the wired network component, and user interface functions in the mobile component. In some cases, the boundary may change dynamically.

Partitioning is only one part of designing applications that make effective use of the mobile network. Table 1 lists several well-known techniques for more effective use of the communications link and their potential benefits. Unfortunately, the policy decisions for when to exercise many of these usually rely on predictions concerning data access patterns.

The following design strategies strive to maximize the accuracy of these predictions, provide additional information about relative data importance, and increase the usability of the wireless application.

Expose network costs. Structure the user interface to unobtrusively convey information about high cost operations, offer lower cost versions of the same, facilitate user hints to designate background data transfer, and facilitate user-specified data filters that are applied on the wired-network side.

Organize to provide reference information. Structure the application to imply what data will be accessed next based on the current context, and to allow data transfer during user latency, transfer of important data first or by itself (hierarchical), and continued operation on available data when new data requests are unsatisfied.

Utilize wired-network resources. Use a wired-network workstation to reduce or filter the data sent to the client, provide additional resources to the mobile device (CPU, disk space), hide limitations of the mobile device from network applications (including disconnection, high latency), and allow the mobile device to exploit services in the network.

Adapt to variations in network connectivity. Structure the application to allow adaptation to connectivity changes through migrating function and adaptive data transfer, and by providing functionality on the mobile device to support some form of disconnected operation.

In general, the application should be structured to expose information about which data is necessary and important (to the user as well as to the application), which data is most likely to be needed next, and alternative actions to be taken if the data is unavailable. This information is used to more effectively exploit traditional techniques.

3 WIRELESS PLATFORM

The software system platform for our wireless applications is Wit [12], the successor to BNU [13]. Wit is a system infrastructure that supports wireless connection of a palmtop to the wired LAN. Wireless communication occurs through infra-red transceivers and software developed at Xerox PARC. Wit provides a primitive windowing system, non-preemptive threads, and connections, all of which are controlled by applications through a Tcl [9] interpreter. These services support local execution, which is required by disconnected operation, and permit a wide range of solutions for hiding low bandwidth constraints.

The system hardware platform includes HP100lx palmtops (PC-XT compatible, 80c186 processor) as the mobile devices, SUN workstations and PARC IR transceivers. The transceivers communicate with an attached palmtop or workstation via a serial link at 38000 bps, and over the wired network at 19200

bps. In practice, the data throughput is significantly lower (3000-4000 bps) due to a combination of transceiver and protocol overheads.

Wireless communication links are shared with ParcTabs [1] using the PARC IRNet. Communication is routed as follows. Packets travel from the palmtop to its attached transceiver, wirelessly to another transceiver attached to a workstation, and then to the controlling gateway process on the workstation. The gateway forwards the packet on to the appropriate Wit process for that palmtop, which may pass the data on to an application component. The ParcTab software tracks location and performs cell handoff as appropriate.

4 W* : A WIRELESS APPLICATION

We focused our initial application efforts on a World Wide Web (WWW) [3, 15] client browser. This is a hypertext browser that retrieves documents based on the user's selection of a hyperlink. Hyperlinks point to documents that are related to the text in which they appear. There are several motivations for choosing this application:

1. The growing popularity of WWW [8] is increasing the amount of interesting and useful information accessible through web browsers.
2. Information retrieval and management are two of the primary uses of PDAs. A WWW client is an obvious choice for extending information retrieval over the wireless connection.
3. The highly structured nature of hypertext documents is well suited to exploring application specific optimizations for low bandwidth links.
4. The hypertext interface has minimal input requirements that are well suited to pen or limited keyboard interactions.

Other applications that allow interactive retrieval of information from the wired network, such as email, netnews, or documentation readers, are also good candidates for this approach.

W*, Wit's wireless WWW browser, is used to illustrate how application structure and data can be tailored for better use of communication optimization techniques. Table 2 lists the techniques along with some example applications

Technique	Applied to WWW client
caching	cache frequently accessed documents
prefetch	prefetch the first page of the first N hyperlinks
data encoding	compress documents
lazy evaluation	display first screen and lazily evaluate page down
futures	display first screen with rest of document as future
data reduction	demand for uncached document retrieves outline only

Table 2 Application of Communication Optimization Techniques

in a WWW client. Caching, prefetching and data reduction are implemented in W*. Lazy evaluation and futures are supported in the HTML parsing and display routines, but the current version of W* transfers only complete documents or outlines. Data encoding is not implemented.

4.1 User Interface

A document's hyperlinks are used to convey additional information about the cost (latency) of traversing a hyperlink. This can be done unobtrusively through underlining (the percentage underlined reflects relative cost) or through a status line for the current hyperlink. This feedback helps set user expectations, although it is not necessary for the user to understand the new information in order to continue using the application. Prefetching is assisted by allowing the user to specify a set of hyperlinks in which they are interested. The system works on obtaining these while the user is reading the current document. An alternative way to traverse a hyperlink is to retrieve just an outline consisting of the headings and hyperlinks for a document. Like a "folding editor", the user can expand sections of the document by selecting section headings.

4.2 Workstation Resources

W* uses a two level cache for documents, one at the palmtop, and one at the workstation. Prefetching documents to the workstation, but not all the way to the palmtop, hides some latency without placing additional demands on the wireless link. The workstation cache also facilitates user-specified data reduction. Scripts, possibly written or parametrized by the user on the palmtop,

can be executed remotely on the workstation. A simple example is a filtering script that returns only documents or sections of documents that match some criteria. The script examines a set of cached documents on the workstation by starting in one document and following its hyperlinks to a specified depth or until an uncached reference. It selects matching sections and sends an outline of the results to the palmtop. General purpose scripts or templates can be shared among users.

4.3 Application and Data Structure

In W*, parsing functionality is duplicated on both sides of the wireless link. Disconnected access to cached documents requires local parsing. But in some cases (for example, large documents or wireless contention) it becomes desirable to either send outlines or, if the target of a hyperlink is not at the top of a document, only transfer the desired page and lazily bring over other pages as needed. These solutions require that the workstation parse the document as well.

When browsing a hypertext document, the most likely operations are page up, page down, or traversal of a visible hyperlink. The structure of the application and data provide good “hints” about which data to prefetch. Prefetching the first screen of output from each of these actions can improve response times. The number of visible hyperlinks, current resource constraints, and document or user specific access patterns suggest when prefetching can be profitably employed. Another concern with both prefetching and user scripts that access HTTP servers is the possibility of overloading the servers with a large number of successive requests. Guidelines for WWW crawlers [4], programs that explore and index the web, have been developed to minimize the adverse impact of automatically accessing sets of documents.

5 EXPERIENCES

Table 3 shows the component costs for a typical HTML document access from the palmtop to a remote HTTP server. The document size of 6.2K is an average derived from httpd access logs. Wireless file-transfer occurs over a reliable connection that uses an unoptimized send/acknowledge protocol with no sliding windows. This connection is a good approximation of typical wide area wireless communication bandwidth.

Access 6.2K HTML document	seconds	percent
HTTP GET (to workstation)	2.9	17
workstation to palmtop	14.2	82
other communication	0.2	1
Total (to palmtop)	17.3	100

Table 3 Sample Access Times for a Typical Document

Palmtop file caching has the largest impact on performance. The prototype does not currently make any attempt to validate the cached copy, but the user can request that it be re-loaded from the network. User specified background fetch of one or more documents also improves the usability of W*, since the user can ask for other documents early, then continue to read the current document.

The HTTP GET is performed by executing a perl script, “hget” from the Tcl interpreter. Although the variance for the HTTP GETs that provided the data in Table 3 was not very high, in general, document retrieval via HTTP exhibits unpredictable latencies. This may make workstation caching and prefetching more desirable.

We are currently conducting a study on World Wide Web document access patterns [14] to provide additional data on the expected effectiveness of caching and prefetching. It is often not advisable to prefetch all visible hyperlinks. When viewing a densely hyperlinked document or during periods of high network contention it is useful to know that a particular hyperlink traversal is likely. We are analyzing server access logs to look for patterns where an access to document A is typically followed by an access to document B. This uncovers document-specific access patterns. We also plan to use client traces to determine user-specific access patterns. Although there are a number of limitations to this approach, preliminary results indicate certain subtrees of documents are good candidates for prefetching. These are usually documentation files or on-line books that are traversed in section order.

We do not yet have enough experience with the outline retrieval to evaluate its usefulness in practice. The effectiveness of this technique depends largely on the relative sizes of the outline and the original document. On small or densely hyperlinked documents, little benefit is obtained. Where there is a significant

difference, the outline can be effective if either the user is familiar with the contents, or the headers and hyperlink names are self-explanatory.

6 CONCLUSIONS

The performance disparity between wireless and wired networks motivates the need for applications that better tolerate slow communication links. This paper presents a strategy for designing applications that are partitioned across the wireless link. The application data and functional organization provide implicit information about reference patterns that can be utilized to increase the benefits of traditional techniques.

Based on our experiences with W* and other wireless applications, we are developing a new system. Wit II [11] is designed to provide effective management of mobile computing resources using a cooperative approach between applications and the system. The system manages data transfer between the components of a partitioned application, employing techniques such as filtering, prefetching and caching. Application-supplied information about data structure, priority and access patterns is used to improve system policy decisions. This approach reduces the complexity of developing mobile applications that perform well by decoupling the implementation of optimization techniques from the specification of application data semantics that guide policy decisions.

Acknowledgments

The author would like to thank Brian Bershad and Mark Weiser, who initiated the Wit project and contributed valuable insights and advice during its design. Many of the application design strategies presented here were influenced by discussions with members of the Tab project at Xerox PARC, and the TIP and Coda projects at CMU. Dylan McNamee and Stefan Savage provided helpful comments on this paper.

REFERENCES

- [1] N. Adams, R. Gold, B. Schilit, M. Tso, and R. Want. An Infrared Network

- for Mobile Computers. *Proceedings of the 1st Usenix Symposium on Mobile & Location-Independent Computing*, pp. 41-51, August 1993.
- [2] A. Athan and D. Duchamp. Agent-Mediated Message Passing for Constrained Environments. *Proceedings of the 1st Usenix Symposium on Mobile & Location-Independent Computing*, pp. 103-107, August 1993.
 - [3] T. Berners-Lee, R. Cailliau, A. Loutonen, H. F. Nielsen, A. Secret. The World Wide Web. *Comm. of the ACM*, vol 37, no 8, pp. 76-82, August 1994.
 - [4] Guidelines for Robot Writers. <http://web.nexor.co.uk/mak/doc/robots/guidelines.html>
 - [5] L. B. Huston, P. Honeyman. Disconnected Operation for AFS. *Proceedings of the First Usenix Symposium on Mobile & Location-Independent Computing*, pp. 1-10, August 1993.
 - [6] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda Filesystem. *ACM Trans. on Computer Systems*, 10(1), February 1992.
 - [7] J. A. Landay and T. R. Kaufmann. User Interface Issues in Mobile Computing. *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pp. 40-47, October 1993.
 - [8] NSFNET Statistics at GVU Center. <http://www.cc.gatech.edu/gvu/stats/NSF/merit.html>
 - [9] J. K. Ousterhout, *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, 1994.
 - [10] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A Highly Available Filesystem for a Distributed Workstation Environment. *IEEE Trans. on Computers*, 39(4), April 1990.
 - [11] T. Watson. Effective Wireless Communication through Application Partitioning. *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, pp. 24-27, May 1995. http://snapple.cs.washington.edu/papers/hot_os.ps
 - [12] T. Watson. Wit: An Infrastructure for Wireless Palmtop Computing. Technical Report UW-CSE-94-11-08. University of Washington, November 1994. <http://snapple.cs.washington.edu/papers/wit.ps>

- [13] T. Watson, B. Bershad. Local Area Mobile Computing on Stock Hardware and Mostly Stock Software. *Proceedings of the 1st Usenix Symposium on Mobile & Location-Independent Computing*, pp. 109-115, August 1993. <http://snapple.cs.washington.edu/papers/bnu.ps>
- [14] Wit Web Document Access Study. <http://snapple.cs.washington.edu/wit/webstats/webstudy.html>
- [15] World Wide Web initiative at CERN. <http://info.cern.ch/hypertext/WWW/TheProject.html>

Wireless World Wide Web (W4) project [2]. The W4 system consists of a Web client that runs on an Apple Newton MessagePad and communicates with the wired network via a ~2400 baud Motorola CELLect modem and MicroTac phone. The Web client is structured so that a workstation host on the remote end of the modem link does most of the document formatting, thereby relieving the less powerful PDA of the CPU intensive formatting task and conserving as much of the limited wireless bandwidth as possible. The W4 project has shown that current PDAs are feasible platforms for running Web clients when care is taken in implementing the client.

The Dynamic Documents project at MIT has also investigated providing mobile access to the World Wide Web [6]. Dynamic Documents are Tcl scripts that, once received, are executed in a "safe" Tcl interpreter in a modified Mosaic Web client. The output of the Tcl script is an HTML document that is displayed by the client. Dynamic Documents can interact with the Web client, for example, to configure their output for displaying on small displays. In addition, Dynamic Documents can use the Tk library to create their own user interfaces to implement more complex Web applications from within the client.

The primary difference between the Dynamic Documents project and Mobisaic is that the Dynamic Documents project has focused on the problems of overcoming the limited bandwidth of the wireless communication link and the display limitations of mobile devices, whereas Mobisaic has focused on enhancing the utility of the Web by incorporating information from the user's mobile computing context into the system.

1.2 Paper outline

The rest of this paper is organized as follows. Section 2 gives an overview of the World Wide Web system, and describes the extensions to the system used for incorporating a user's mobile computing context into the WWW. Sections 3 and 4 describe dynamic URLs and active documents. Section 5 discusses how Mobisaic can be useful in the desktop environment as well as the mobile environment. Section 6 describes the implementation of Mobisaic. Section 7 discusses future work, and section 8 summarizes.

2 SYSTEM OVERVIEW

This section first provides a high-level overview of the World Wide Web information system, and then describes the extensions used by Mobisaic for incorporating a user's mobile computing environment into the system.

2.1 The World Wide Web

The three main components of a World Wide Web (WWW) information system are clients, documents, and information servers. The user interacts with the system using a Web client, which lets the user name and load documents from servers for viewing. Web clients typically support a number of different document types, such as files available via ftp, netnews, and Hypertext Markup Language (HTML), and support connections with a variety of information servers, such as ftp daemons, news servers, and Hypertext Transport Protocol (HTTP) daemons. Documents are files on a server referenced by Uniform Resource Locators (URLs), and they can contain a variety of data types, including ASCII text formatted according to HTML directives, embedded pictures, and audio and video clips, as well as embedded URLs that are used as hypertext links to other documents. URLs can also name programs on HTTP daemons that, when executed, produce an HTML document as output.

2.2 Extensions for a mobile WWW

In its current form, the Web infrastructure cannot easily accommodate mobile clients because the dynamic information it supports is either returned from the server without incorporating any user context at all, or is incorporated explicitly using forms-based interfaces that require user input on the client. Moreover, there is no support for automatically updating a document when it, or the reason for displaying it, changes.

To better support the use of dynamic information, we have extended the Web infrastructure to include:

- A network server that maintains mobile computing contexts within a client-specific domain;
- An asynchronous callback mechanism to notify Web clients when a user's dynamic computing environment changes;

MOBISAIC: AN INFORMATION SYSTEM FOR A MOBILE WIRELESS COMPUTING ENVIRONMENT

Geoffrey M. Voelker and Brian N. Bershad

*Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195*

ABSTRACT

Mobisaic is a World Wide Web information system designed to serve users in a mobile wireless computing environment. Mobisaic extends the Web by allowing documents to both refer and react to potentially changing contextual information, such as one's current location in the wireless network. Mobisaic relies on client-side processing of HyperText Markup Language documents that support two new concepts: *Dynamic Uniform Resource Locators (URLs)* and *Active Documents*. A dynamic URL is one whose results depend upon the state of the user's mobile context at the time it is resolved. An active document automatically updates its contents in response to changes in a user's mobile context. This paper describes the design of Mobisaic, the mechanism it uses for representing a user's mobile context, and the extensions made to the syntax and function of Uniform Resource Locators and HyperText Markup Language documents to support mobility.

1 INTRODUCTION

This paper describes Mobisaic, a system that uses the World Wide Web [3] to enable information browsing in a mobile computing environment. Information browsing is an ideal mobile application because it allows users to interact with their environment as they work within it; it places minimal requirements on a user-input device; and it cannot be handled with large on-board caching. With mobile information browsing, users can discover who and what is in their im-

Copyright ©1994 by IEEE. Reprinted, with permission, from Proceedings of the Workshop on Mobile Computing Systems and Applications, December 1994.

mediate surroundings, whether it is colleagues at a business meeting, speakers at a presentation, projects in a lab, or displays in a museum. A mobile information system can also allow users to execute general queries that incorporate information from their environment, such as finding the nearest cafe or the nearest bus stop that will take them to a specific location.

Users of the World Wide Web (WWW) rely on client browsers to access information servers on the Internet. With Web clients, users browse documents written in the HyperText Markup Language (HTML) by traversing hypertext links, called Uniform Resource Locators (URLs), that load documents or invoke programs on the information servers.

Mobisaic extends standard client browsers to take advantage of mobility in two ways. First, Mobisaic allows authors to reference dynamic information, such as a user's location, in hypertext links called *dynamic URLs*. When the user traverses a dynamic URL, the client resolves any references to dynamic information it may contain and sends the result to the server. Second, Mobisaic supports *active documents*, documents that present and automatically update information for the user as the information they contain changes or otherwise becomes invalid. The update is done by the client browser, which receives notifications when the dynamic information changes.

Dynamic information in Mobisaic is represented using dynamic environments [9]. Just as standard UNIX shells provide environment variables to customize applications started from the shell (e.g., **DISPLAY**), dynamic environments provide environment variables for customizing mobile applications. For example, a dynamic document might include a reference to the **Location** dynamic environment variable to customize its contents according to the user's current location.

1.1 Related work

Researchers at Xerox PARC have broadly introduced the idea of context-aware applications [8]. They also initially proposed the notion of dynamic environments as a means of representing and disseminating information from users' mobile contexts throughout the system. Mobisaic is essentially an application of those ideas to World Wide Web browsing.

Joel Bartlett at DEC WRL has investigated the feasibility of implementing and using a Web client on a handheld personal digital assistant (PDA) in the

- A syntax for referencing dynamic information in URLs and documents.

Representing mobile computing contexts

Mobisaic uses dynamic environments to represent a user's mobile computing context. The basic unit in a dynamic environment is the dynamic environment variable, which is conceptually similar to a standard UNIX shell environment variable: dynamic environment variables have a name and a value, and they can be accessed and changed to customize applications to the user's mobile computing environment just as shell environment variables customize applications launched from the shell. However, unlike shell environment variables that are associated with a login process, dynamic environment variables are associated with users and places, and have indefinite lifetimes. Applications on the network with sufficient privilege can access and change dynamic environment variables, and whatever changes they make can be seen by other applications with sufficient access privileges.

Notification of changes in mobile computing contexts

Active documents allow environmental changes to be reflected in the information displayed to the user. If the information in an active document becomes invalid, then the client can be notified of the change so that it can display a more relevant document. Included in such notifications are the name of the variable that changed and its new value.

For example, say that the user changes cell locations in the wireless environment. The wireless communications system that is monitoring the user's location can publish the new location by updating the **Location** variable in the user's dynamic environment. If the user were displaying a document that was sensitive to location, the client would have subscribed to the **Location** variable, and would receive a notification informing it of the change. At this point, the client could take action in response to the change, such as loading a new document that directly relates to the user's new location.

Syntax and scope

A Mobisaic client relies on a syntax for referencing dynamic environment variables within dynamic URLs and active documents. The syntax supported by Mobisaic is of the form $\$(environment.variable)$, where *environment* and *variable*

able denote a dynamic environment and dynamic environment variable, respectively. For example, `$(voelker.Location)` would reference the name of Geoff's current location in the wireless network. Mobisaic also supports a shorthand notation for referencing variables in the user's own dynamic environment. If the reference doesn't contain the name of an environment, then Mobisaic assumes that the variable referenced is in the environment associated with the user. Thus, `$(Location)` refers to the **Location** environment variable in the user's dynamic environment.

Mobisaic supports recursive references to dynamic environment variables so that environment or variable names can themselves be references to dynamic environment variables. For example, given that locations have dynamic environments associated with them, `$($(Location).Printer)` would reference the **Printer** variable in the environment associated with the user's current location.

3 USING DYNAMIC URLs

Dynamic URLs allow a single URL to return different documents or execute different commands depending upon the state of the user's dynamic environment at the time the URL is selected. A URL is dynamic if it references at least one dynamic environment variable. For example, in our department we have written HTML documents describing ourselves and the places in which we work (see Figure 1). The name space of these documents on our server is well structured, enabling the following dynamic URL to return the document describing the user's current location:

```
http://www/places/$(Location).html1
```

Another example of a dynamic URL is a Web server that has a program **busroute** which takes a starting location, a destination, and a time as arguments, and returns an HTML document detailing how to get to the closest bus stop on the shortest bus route to the destination. A dynamic URL to find the bus route to the Space Needle in Seattle would appear as:

```
http://www/htbin-post/busroute?\$ (Location)+SpaceNeedle+$ (Time.TIME)
```

¹For brevity, we use *www* in place of our server at *www.cs.washington.edu*.

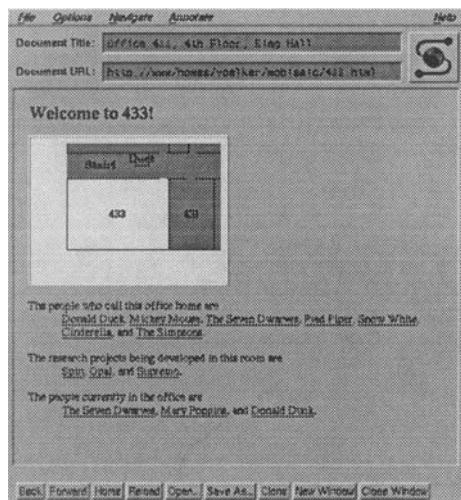


Figure 1 The WhereAmI active document referenced by the dynamic URL `http://www/places/$(Location).html`.

Note that the question mark and plus signs in the query are standard URL syntax denoting the arguments that are passed to the program invoked on the server.

3.1 Resolving dynamic URLs

When a user selects a dynamic URL in a document, the client browser is responsible for resolving all references to dynamic environment variables within the URL. The client obtains the values of dynamic environment variables from the appropriate dynamic environment and replaces the references with the values as strings. When all variable references have been resolved, the result is a standard URL that the client then sends to the server. For example, if a user were in office 433 and selected the location description dynamic URL in the previous section, the client would resolve the **Location** dynamic environment variable in the user's context and send the following URL to the server:

```
http://www/places/433.html
```

Having the Mobisaic client resolve the dynamic environment variable references gives the most flexibility to the system. The variable references could have been resolved in two other places: the application, if the dynamic URL named a program on the server to execute, or the server. If applications had to resolve the variable references, then dynamic URLs would be limited to naming only those applications that were modified to understand and use dynamic environment variables. Likewise, if the server were to resolve the references, then dynamic URLs would be limited to using only dynamic environment aware servers. When the client resolves the references, however, dynamic URLs can evaluate to a URL that names any document or program on any server that a standard URL can name.

4 ACTIVE DOCUMENTS

Active documents are HyperText Markup Language documents that enable the Web client to automatically react to changes in a user's mobile computing context. They give the client the ability to update the information being displayed without the user having to take action, and place less of a burden on the user to search for information by placing more of a burden on the author to organize it. This tradeoff is possible in a mobile environment because users who roam the wireless network are quite likely to be interested in who and what is in their immediate surroundings, and the information in a user's mobile context is enough to enable a Web client to do the searching on the user's behalf. In this way, active documents can change the way users interact with the Web: they spend less time searching for information because the client presents it to them as they interact with their surroundings.

This section describes how to write active documents and how the client handles them when they are being viewed by the user. To illustrate these processes, this section also describes a set of pages collectively called the WhereAmI active document, which is a guide for visitors to our department. Each page corresponds to a place in our wireless network and contains a brief description of the place, links to the home pages describing the occupants of the place, and links to pages describing any projects hosted in the place. When a user selects this active document, the client loads the page corresponding to the user's location. Then, as the user changes rooms, the client automatically discards the page describing the old room and replaces it with the one describing the new room. Each page in the WhereAmI active document can be loaded using the dynamic URL from the previous section:

`http://www/places/$(Location).html`

Authors write active documents just like they write standard HTML documents, with one addition. They must place a *subscribe* command in the document which lists the dynamic environment variables that the client must subscribe to when it loads the document. In effect, the variables listed in the subscribe command are the elements of a user's mobile context that, when they change value, invalidate the information in the document. The new values of the variables also provide the information necessary for the client to determine which document to load in place of the current one.

A subscribe command is embedded in an HTML comment line. The command has the following form:

`<!-- (subscribe to variable variable ...) -->`

Variable is a standard reference to a dynamic environment variable. When the client loads the document and parses subscribe commands, it subscribes to each variable specified in the command.

When the client receives a notification for a subscribed variable indicating that the variable has changed value, the new value of the variable in the notification determines what action the client will take at the moment of notification. The new value of the variable can be an explicit directive to the client:

reload Re-execute the URL that loaded the current document. If the URL is dynamic, the references to dynamic environment variables are resolved again.

load URL Execute a new URL and load the document in the same window.

spawn URL Execute a new URL and load the document in a new window.

close Close the current window.

Otherwise, the client does not interpret the new value and simply reloads the document. If the URL naming the document is a dynamic URL, then the client will evaluate the dynamic URL as if the user had selected it.

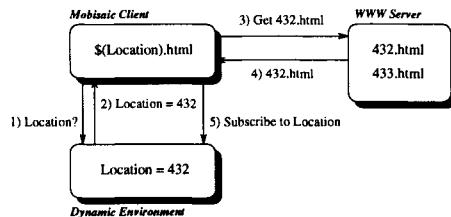


Figure 2 An example showing a user loading the WhereAmI active document named by the dynamic URL `$(Location).html`.

For the WhereAmI active document, each page has the following subscribe command:

```
<!-- (subscribe to $(Location)) -->
```

The command tells the client that, when it loads the page, it should subscribe to the **Location** variable in the user's dynamic environment. When this variable changes value, the client will receive a notification with the new value of **Location**. Since the value is not an explicit command to the client, it will ignore the value itself and use the notification as a signal to reload the active document. And since the URL naming the document is dynamic, the client will re-evaluate it and load the active document that describes the user's new location.

Figures 2 and 3 show an example of how dynamic URLs and active documents in the WhereAmI application interact with a user's changing mobile computing environment. Figure 2 shows a user running a Mobicasic client browser, a WWW server, and the user's dynamic environment. In this figure, the user is in room 432 and is loading the dynamic URL `$(Location).html`. The client resolves this dynamic URL by sending a message to the user's dynamic environment, asking for the value of the **Location** variable (message 1). The dynamic environment responds with the value of **Location** (message 2). The client now has a fully resolved dynamic URL to the active document **432.html**, which it asks the server to return (message 3). The server responds with the document **432.html** (message 4). Since **432.html** is an active document that contains a subscription to the variable **Location**, the client subscribes itself to the variable in the user's dynamic environment (message 5).

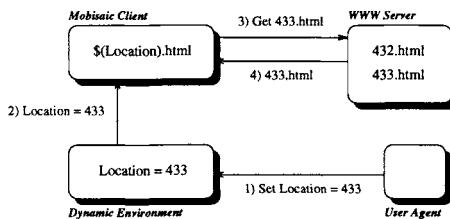


Figure 3 The example continued showing what occurs when the user changes location.

Figure 3 shows what takes place when the user moves from room 432 to room 433. In this figure, a “user agent” process has been added to represent a process that tracks the user’s movement in the wireless networking environment. When the user moves from room 432 to room 433, the agent detects the movement, performs its normal tasks for such an event (such as updating network routes), and lastly updates the variable **Location** in the user’s dynamic environment with the user’s new location (message 1). Upon receiving the update, the dynamic environment notifies all processes subscribed to the **Location** variable or the environment. Since the client is subscribed to the variable, it receives a notification (message 2). Upon receiving the notification, the client reloads the active document by executing the dynamic URL again. In a process similar to that shown in Figure 2, the client resolves the dynamic URL to **433.html**, which corresponds to the document describing the user’s new location. The client requests this document from the server (message 3), the server returns it (message 4), and the client displays it to the user.

Arbitrary client notifications

In addition to receiving notifications for any active documents the client is displaying, the client can also receive notifications that do not refer to any of its displayed documents. To receive such notifications, the client subscribes to the **MOBISAIC-STREAM** variable in the user’s dynamic environment. Any process can then send notifications to the client, for example, to spawn a window and load a new document that might be of interest to the user.

5 MOBISAIC ON THE DESKTOP

Although Mobisaic was originally inspired for use in a mobile computing environment, it can also be useful in the desktop environment. There are a number of information sources in the WWW that produce information periodically, and it is straightforward to write documents in Mobisaic that tap into these sources. Some documents that have already been written include simple daily scripts that, early in the morning, publish the URLs for the Dr. Fun and Dilbert comic pages to a list of interested users running Mobisaic. The published URLs spawn new windows showing the contents of the pages. When users come in to work in the morning, the daily comic pages are already showing on their screens.

As another example, we use active documents together with electronic mail to implement a distributed, recommendation-based “hot list” of new and interesting pages. Our local departmental version of Mosaic has been modified to make it easy to forward URLs to others in our department using email. It is now common practice for people in the department to forward to colleagues URLs that they have discovered and find interesting. The email messages generated by Mosaic have a special mail tag, *X-URL*, that contains the URL being forwarded. A user’s incoming mail filter detects these tags and publishes the URLs in them to the user’s Web client. The result is that, when users have a URL forwarded to them, a window automatically appears on their screen displaying the document referenced by the URL.

A third application uses active documents to display and update stock quotes. A background filter monitors a stock quote information source, and, in a dynamic environment for stock prices, publishes the latest values of the stocks it monitors as they change. Documents displaying the information for a given stock subscribe to the dynamic environment variable associated with the stock in the stock dynamic environment. When users view a document describing or referencing a stock, the document will update itself as new stock values are published in the dynamic environment.

6 IMPLEMENTATION

This section discusses our implementation of Mobisaic and the changes we applied to a standard Web client to use active documents and dynamic URLs.

6.1 Dynamic environments

Dynamic environments [9] provide a network-based publish and subscribe infrastructure [5]. A dynamic environment is maintained by a subscription-based server to which applications broadcast queries and from which applications receive multicast notifications. We use the Zephyr notification system [4] to implement the publish and subscribe facilities.

We chose to use Zephyr because it provides network transparency, automatic subscriber-based routing, authentication, asynchronous notification, and the ability to run redundant Zephyr clients supporting redundant instances of the same dynamic environment. (We presently do not take advantage of Zephyr's authentication system or support for redundancy). Zephyr is not without its drawbacks, however. It operates only within relatively small administrative domains, such as a department or campus. It cannot distribute information quickly to many hosts, which can become a problem in the current system during heavy load. (This bottleneck and possible solutions to it are discussed in [7].) Fortunately, a C library interface hides the use of Zephyr from Web clients, so changing to a new transport should be relatively easy.

In our implementation, dynamic environments are maintained by a *dynamic environment server*, a process that maintains the data structures for an arbitrary number of dynamic environments. For each dynamic environment it maintains, the server subscribes to a Zephyr message class that uniquely corresponds to the dynamic environment by sending a Zephyr subscription to the Zephyr server. This message class is the one to which clients address messages in order to make requests of the dynamic environment.

Client processes subscribe to dynamic environments and their variables by sending a subscription to the Zephyr server that names the Zephyr message class corresponding to the dynamic environment and variable. Note that client subscriptions are not sent to the dynamic environment servers; the Zephyr server keeps track of which clients are subscribed to which dynamic environments and variables through the subscriptions sent to it by both the dynamic environment clients and servers. When a dynamic environment server wants to notify clients of a change in a dynamic environment variable, it sends a message to the Zephyr server naming the Zephyr message class that corresponds to the dynamic environment variable. The Zephyr server then multicasts this message to all clients subscribed to this message class.

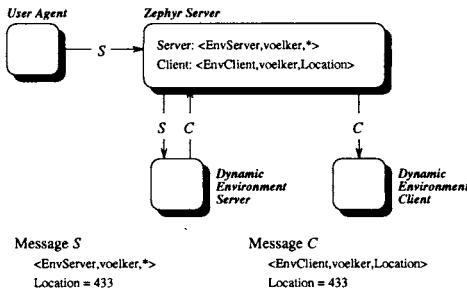


Figure 4 An example showing how dynamic environment events are propagated through the Zephyr system.

Figure 4 shows an example of how dynamic environment events are propagated through the Zephyr system. This figure shows the Zephyr server, a dynamic environment server maintaining the **voelker** dynamic environment, a dynamic environment client, and a user agent process that keeps track of the user in the wireless network. All entities are processes that can run on any machine in the Zephyr administrative domain.

The dynamic environment server has subscribed to the Zephyr message class **<EnvServer,voelker,*>** at the Zephyr server. Dynamic environment clients send messages to this message class to send requests to the dynamic environment server. The dynamic environment client has subscribed to the Zephyr message class **<EnvClient,voelker,Location>** at the Zephyr server. The dynamic environment server sends messages to this message class to notify clients of changes to the variable **Location** in the **voelker** dynamic environment.

In Figure 4, the agent process is updating the variable **Location** in the **voelker** dynamic environment. To do so, it sends the message S , which names the dynamic environment server and contains the new value for the variable, to the Zephyr server. The Zephyr server then multicasts the message S to all processes subscribed to this message class. By design, only the dynamic environment server subscribes to this class and only it receives the message from the Zephyr server.

The dynamic environment server updates the **Location** variable in the **voelker** environment, and then proceeds to notify all dynamic environment clients subscribed to this variable. To notify clients, the dynamic environment server sends the message C , which names all clients subscribed to this variable and contains the new value of the variable, to the Zephyr server. The Zephyr server

then multicasts the message C to all processes subscribed to this message class. Since the dynamic environment client is subscribed to this message class, it receives the message and is thereby notified that the **Location** variable in the **voelker** dynamic environment has changed to the new value specified in the message.

6.2 Client modifications

Any Web browser can be modified to support dynamic URLs and active documents provided that it supports an interface for loading and reloading documents and spawning and closing windows, and it has the ability to add an asynchronous input descriptor to its set of inputs. A client library handles all communication with dynamic environments, and parses dynamic environment variable references. Filters, supplied by the library, are applied to the input and output communication paths. The filter on the output stream resolves references to dynamic environment variables embedded in dynamic queries, and the filter on the input stream subscribes the client to dynamic environment variables embedded in active documents. If the client supports an internal interface for document and window manipulation, then it can be directly linked with the Mobisaic client library. For those clients that only support an external interface or do not have an interface for adding asynchronous input descriptors, we have a wrapper program that handles dynamic environments and controls the Web client as a child process.

Our prototype Mobisaic client is the X Mosaic client [1] extended with the Mobisaic client library. The X Mosaic client has a relatively clean internal interface for loading documents and spawning windows, so most of the code changes were to add the file descriptor for the dynamic environment communication channel to the list of input descriptors, the callback to handle asynchronous input on the descriptor, and calls to the Mobisaic library filters on the input and output communication paths. Overall, our changes added less than 60 lines to the client.

7 FUTURE WORK

This section discusses some of the limitations of the current design and implementation of Mobisaic, and suggests possible approaches for overcoming these limitations in future work.

7.1 Callbacks

Integrating the subscribe mechanism of dynamic environments into Web clients effectively provides a callback mechanism for other applications to notify the client that the state of the document the client is viewing has changed in some respect. However, it is not a general callback mechanism for the World Wide Web since it requires the use of dynamic environments. The Internet Engineering Task Force (IETF) is in the process of designing a standard callback mechanism with which Web servers and other applications can communicate asynchronously with Web clients, and Mobisaic's use of dynamic environments should be compatible with this callback standard once it emerges.

7.2 Storing URLs

Uniform Resource Locators are stored in various collections, such as hotlists, session and global histories, indices, bookmarks, and even email, to aid the user in organizing access to interesting documents. Storing dynamic URLs raises an interesting issue about what actually is stored because the user or Web client now has the choice of storing either the dynamic URL itself or the result of evaluating the URL at the time it is stored.

One can imagine wanting to do both in different situations. For example, assume a user has selected the first location-based dynamic URL from section 3 and finds the loaded document interesting. If the user wanted to store this document in a hotlist, the user would want to store the evaluated dynamic URL. However, if the user wanted to store a reference to the WhereAmI document, the user would want to store the dynamic URL itself in the hotlist so that, the next time it is selected, the dynamic URL would be evaluated in the user's current context.

To address this problem, Mobisaic could store evaluated dynamic URLs by default, and allow users to explicitly override the default behavior with a menu option. However, such behavior would require more extensive changes to each Web client supporting the Mobisaic extensions since these changes need to be made to the user interface.

7.3 Disabling active documents

Active documents in Mobisaic are always active in the sense that they will react to notifications to their subscriptions whenever they receive them. This behavior may not always be desirable. For example, imagine that you are in a museum that used active documents to describe its exhibits and you have just walked up to a particularly interesting exhibit. It is so interesting, in fact, that you want to show your friend just down the hall the document that has appeared in your browser. If you walk down the hall to show your friend the document, though, it may no longer be in the browser, having since been replaced with the document that describes the exhibit in front of your friend.

Mobisaic should provide some mechanism for allowing the user to disable active documents, such as with a button at the bottom of the document window, although, again, this would require more client-specific changes.

7.4 Subscriptions

The current method of subscribing active documents to dynamic environment variables is inflexible in two ways. The ability to express interest in combinations of multiple dynamic environment variables is limited. Furthermore, users might want subscriptions in some active documents to continue to hold even after the document is no longer being viewed.

Variable expressions in subscriptions

Subscriptions to multiple dynamic environment variables implicitly form an *OR* clause of those variables since the client subscribes to all variables and receives notifications when any of them change. A more expressive syntax would allow authors of active documents to subscribe their documents to arbitrary logical expressions of dynamic environment variables combined with arithmetic and string operators.

Our implementation of dynamic environments could be extended to support such subscriptions so that, whenever any of the dynamic environment variables referenced in the expression change, the expression included in the subscription would be evaluated. If the expression evaluates to a non-zero value, the client would then be notified of the new value of the variable that changed and, possibly, the value of the evaluated expression. With such a syntax, an author might

subscribe a location-based active document using the following expression that disables the active document in, for example, the restroom:

```
<!-- (subscribe to ${Location} && (${Location}) != "Restroom") -->
```

As another example, the following subscription might be used to receive updates when the stock **AStock** falls below a certain threshold. The stock price is maintained in the dynamic environment **Stocks**, and the threshold **AStock-Threshold** is maintained in the user's dynamic environment:

```
<!-- (subscribe to ${Stocks.AStock} && (${Stocks.AStock) < $(AStockThreshold))) -->
```

Indefinite subscriptions

One feature of Mobisaic is that active document subscriptions only last as long as the document is being browsed. However, this feature is also a limitation. Consider the situation where a user is exploring a building while viewing the **WhereAmI** active document, and is using the **WhereAmI** document as a starting point for exploring other documents when entering a new room. Once the user loads another document instead of the **WhereAmI** active document, though, the location-based subscription is lost. Consequently, when the user moves to a different place, no notifications are sent to the client even though the user might still be interested in receiving them.

It is possible to address this problem with the current version of Mobisaic by having whatever process that updates the **Location** dynamic environment variable also send a **goto** command to the user's Web client subscription, **MOBISAIC-STREAM**, the variable to which the client is always subscribed. However, this approach to the problem is limited in the sense that whatever is updating **Location** now has to know about the Mobisaic application.

Another possibility is to have a button or command in the Web client toggle the scope of subscriptions between single documents or all documents, although this would seem to require too much user interaction. Alternatively, Mobisaic could support a **(subscribe indefinitely to)** syntax that would give active documents the ability to have the client make document-specific subscriptions, such as to **\$(Location) != "Restroom"**, and indefinite subscriptions, such

as to **\$Location**). A button or command could then be provided by the Web client to clear all indefinite subscriptions when the user finished browsing, for example, the set of documents comprising the WhereAmI application.

8 CONCLUSIONS

This paper has described a World Wide Web information system called Mobisaic that investigates information browsing in a mobile wireless computing environment. Mobisaic introduces two mechanisms, dynamic URLs and active documents, for incorporating contextual information from a user's mobile computing environment into the Web. Dynamic URLs allow a single URL to return different documents or execute different commands depending upon the values of the embedded variables at the time the URL is selected by the user. Active documents specify dynamic environment variables to which clients subscribe. With these subscriptions, clients respond to changes in the user's mobile context and update the active documents being browsed.

Minimal modifications are required to Web clients and the URL syntax to support the features of Mobisaic. Web servers need no modifications whatsoever. A library hides the details of the communication mechanism and provides filters for parsing and resolving dynamic environment variable references, making it straightforward to modify a Web client to take advantage of the features of the Mobisaic Web system.

The X Mosaic Web client has been modified to use the Mobisaic extensions, and is currently running on PC laptops running the Linux operating system. The laptops are mobile and communicate with the World Wide Web using Proxim RangeLan2 wireless ethernet PCMCIA cards. The WhereAmI application described in this paper is accessible via the first dynamic URL from section 3. An infrared receiver attached to each laptop's serial line detects transmissions from infrared beacons placed in rooms and hallways, providing the fine-grained location information needed to make the WhereAmI application useful. Work is ongoing on incorporating the Mobisaic extensions with a second Web client, the W* client running in the Wit [10] mobile environment.

Acknowledgements

We would like to thank Marc Fiuczynski, Ed Lazowska, Hank Levy, Stefan Savage, Terri Watson, George Forman, and John Zahorjan for their suggestions on Mobicasic. Bill Schilit at Xerox Parc has been incredibly helpful in discussing context-aware applications. Finally, special thanks are due to Xerox PARC for supplying us with the software and hardware that began our experiment in mobile computing.

REFERENCES

- [1] Marc Andreessen and Eric Bina. "NCSA Mosaic: A Global Hypermedia System" In *Internet Research*, 4(1):7–17, Spring 1994.
- [2] Joel F. Bartlett. "W4 — the Wireless World Wide Web." In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pp. 176–178, December, 1994.
- [3] Tim Berners-Lee, Robert Cailliau, Jean-Francois Groff, and Bernard Pollermann. "World-Wide Web: The Information Universe" In *Electronic Networking: Research, Applications, and Policy*, 2(1): 52–58, Spring 1992.
- [4] C. Anthony DellaFera, Mark W. Eichen, Robert S. French, David C. Jedinsky, John T. Kohl, and William E. Sommerfeld. "The Zephyr Notification Service." In *Proceedings of the USENIX 1988 Winter Conference*, Winter 1988.
- [5] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. "The Information Bus — An Architecture for Extensible Distributed Systems" In *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*, December 1993.
- [6] M. Frans Kaashoek, Tom Pinckney, and Joshua A. Tauber. "Dynamic Documents: Mobile Wireless Access to the WWW." In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pp. 179–184, December, 1994.
- [7] Bill N. Schilit and Marvin M. Theimer. "Disseminating Active Map Information to Mobile Hosts" *IEEE Network*, September, 1994.
- [8] Bill N. Schilit, Norman Adams, Rich Gold, Michael Tso, and Roy Want. "The ParcTab Mobile Computing System." In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pp. 34–39, October 1993.

- [9] Bill N. Schilit, Marvin Theimer, and Brent B. Welch. "Customizing Mobile Applications." In *Proceedings of the USENIX Symposium on Mobile & Location-Independent Computing*, pp. 129–139, August 1993.
- [10] Terri Watson. "Application Design for Wireless Computing." In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pp. 91–94, December, 1994.

PROVIDING LOCATION INFORMATION IN A UBIQUITOUS COMPUTING ENVIRONMENT

Mike Spreitzer and Marvin Theimer

*Xerox Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, California, 94304*

ABSTRACT

To take full advantage of the promise of ubiquitous computing requires the use of location information, yet people should have control over who may know their whereabouts. We present an architecture that achieves these goals for an interesting set of applications. Personal information is managed by User Agents, and a partially decentralized Location Query Service is used to facilitate location-based operations. This architecture gives users primary control over their location information, at the cost of making more expensive certain queries, such as those wherein location and identity closely interact. We also discuss various extensions to our architecture that offer users additional trade-offs between privacy and efficiency. Finally, we report some measurements of the unextended system in operation, focusing on how well the system is actually able to track people. Our system uses two kinds of location information, which turn out to provide partial and complementary coverage.

Appeared previously in Proceedings of 14th ACM Symposium on Operating System Principles, Dec 5-8 1993 and ACM Operating Systems Review Vol. 27, No. 5, Dec 1993. Copyright ©1993 by the Association for Computing Machinery, Inc. Reprinted by permission. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept, ACM Inc., fax +1 (212) 869-0481, or (permissions@acm.org).

1 INTRODUCTION

Mobile and ubiquitous computing requires and can exploit a variety of kinds of location information[9, 7, 4]. Just providing a person with access to their normal computing services on a continual basis requires that their location be known to a certain extent. In addition, if information is available about who and what is in the vicinity of a person, then that person's computing environment and applications can behave in a context-sensitive manner. Applications can reflect a user's current circumstances and can respond to changes that might occur in the user's environment.

While desiring to exploit location information, we consider unrestricted access to a person's location data to be an unacceptable invasion of privacy[5]. One way to address this fundamental tension between location-based functionality and privacy is to try to give each user control over their location information and over who may gain access to it. Unfortunately, guaranteeing that no one can gain unauthorized access to one's location information is, in general, very difficult and expensive.

Another important issue is the accuracy and temporal resolution of location information. The sensing facilities we have available to us are not perfect and hence it is important to determine how well they work in practice. Temporal resolution comes into play because it implicitly defines how small a movement the sensing facilities are able to distinguish, and hence how quickly they are likely to detect a change in someone's location. Providing useful location information to applications thus faces both the problems of limits of the location sensing technologies used as well as protection of users from abuse of those technologies.

A topic not covered in this paper is the spatial resolution provided by a system and the implications that has for the kinds of applications that can be implemented. We did not explore this topic because the only spatial resolution provided by our sensing technologies is "room-level" resolution. This enables applications such as migrating display windows from one's office to a conference room, but does not easily support finer-grained applications, such as "flicking" a window from one's portable notebook computer to that of a neighbor sitting in the next chair.

In order to understand the issues of providing location information to a system we have chosen a suite of location-based applications to focus on and have de-

signed and built a location infrastructure in support of them. The applications we have built or prototyped include the following:

Visitor guidance : Guide a person to a designated location.

Migrating windows : Migrate a user's windows to a designated location.

Note distribution : Send a message to all persons at a given location or set of locations.

Ubiquitous Message Delivery (UMD) : A message submitted for delivery is delivered at the soonest "acceptable" time via the most "appropriate" terminal near the recipient. Acceptable delivery time depends on the context of the recipient. For example, the recipient's profile may specify that messages below a certain priority level should not be delivered when the recipient is in a meeting with other people. Similarly, the most appropriate terminal to use will depend on which devices are available at the recipient's current location.

Media Call : A user can request to be connected to one or more other users—wherever they currently are—by the "best" means available. Choices include video, audio, and teletype "talk" connections. As with UMD, users may specify policy constraints that control which kinds of connections may be established under various circumstances.

Scoreboard : This application is an information-oriented "screen saver". When a display is not being used for anything else, it displays information of general interest, but tailored to the interests of the people nearby.

Responsive environment : A "smart building" can optimize its energy usage by exploiting knowledge about which rooms are occupied. It can also control the environmental settings of each room according to the preferences of the people in them[3].

FindNearest : Find the nearest resource or person matching a given specification, such as "color printer" or "Unix wizard".

Locations : Display the current locations of various persons, printers, copiers, etc. A common variant is to show the locations of all nearby persons, printers, etc. (see Figure 1).

Of these applications UMD and the Locations program are deployed and in use in our lab; for the other applications we have initial prototypes running.

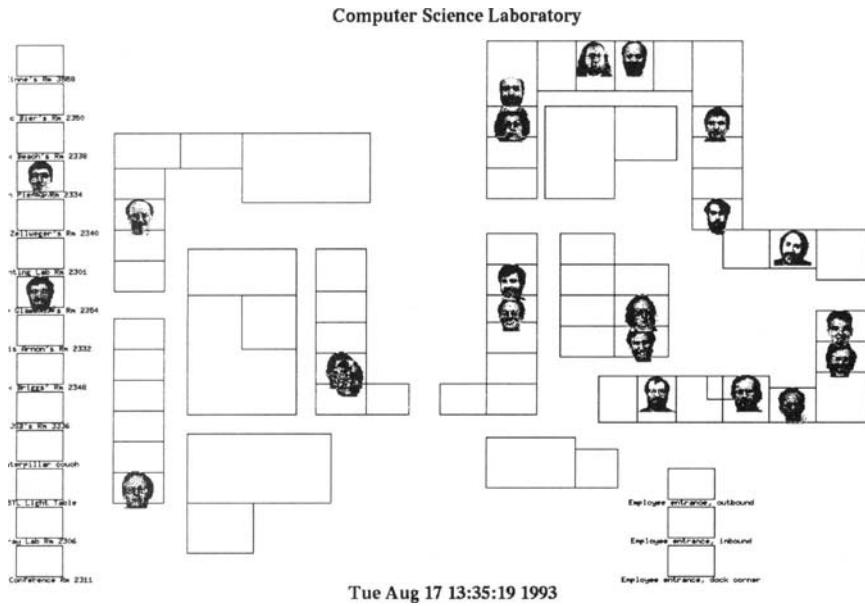


Figure 1 Output of one variant of the Locations program: displays the location of all nearby people willing to be publicly visible.

In the remainder of this paper we describe the location architecture we have designed and built, the design rationales behind it, various extensions one could add to offer users additional privacy/efficiency trade-offs, and the current status of our implementation. We also report some measurements of the system in operation, focusing on how well the system is actually able to track people. We conclude with a discussion of the insights we have gained from our work.

2 ARCHITECTURE

2.1 Key Issues

The design of a location infrastructure must concern itself with a variety of fundamental issues. In order to motivate the design of our architecture we start by presenting the key issues that we wanted to address. Examples of how

applications use our architecture and more detailed design considerations are presented in later sections, after the description of the architecture itself.

Perhaps the most important assumption we make is that our system will span more than one administrative domain. This being the case, we cannot trust all parts of the system with equal measure. In particular, designs that require one to indiscriminately trust the services and servers of foreign administrative domains seem unacceptable to us. The main consequence is that we cannot simply keep everyone's location information in a federation of centralized databases, which would otherwise be the simplest means of providing a location infrastructure.

A second consequence of multiple administrative domains is that we must assume the possibility of sophisticated attempts at traffic analysis occurring in some or all parts of a system. As a result, "perfect" privacy guarantees are, in general, very hard (and expensive) to provide.

An important observation for our design is that most peoples' privacy and functionality requirements differ according to the context they are in. Many situations in which functionality is most desired are also situations in which strict privacy guarantees are not so important or where greater trust of system components is warranted. For example, coworkers in a company with benevolent management might be perfectly willing to have their whereabouts known to each other while at work, but might insist on exercising far greater control over who may know their movements when off the job. Furthermore, if the building they work in is physically secure, they may also be willing to accept a more centralized implementation of the location infrastructure in exchange for greater functionality or efficiency.

Our canonical example of an untrusted, or partially trusted, environment is a shopping mall. It would be undesirable to allow just anyone (such as junk mail senders) to have access to all one's movements within the mall, yet one might wish to be visible to, or reachable by, a select set of friends and family.

An important consideration to keep in mind is that in many circumstances having one's privacy compromised (e.g. while at the mall) is an inconvenience rather than a real problem. Hence, providing guaranteed privacy all the time at a high price—say, in the form of too little functionality and/or too high a performance cost—will not reflect users' true needs. On the other hand, there are clearly circumstances when very strong privacy guarantees are a requirement. The conclusion we draw from these examples is that we need an architecture that provides user-controllable trade-offs between privacy guarantees and both

functionality and efficiency. Much of our design focuses on how to selectively regain the efficiency achievable by centralized designs when users are willing to risk trusting various system components to some degree.

2.2 Description

Figure 2 illustrates the architecture we have designed in response to the issues just discussed. The circles show programs, which run on a network of computers and communicate via RPC. Some of the computers are connected by a wired network; some may be portables that communicate wirelessly. The black arrows show the flow of location information while the gray arrows show the path of a ubiquitous message delivery. Not shown are various application-specific servers and our Name-and-Maintenance Service, which uses familiar techniques to enable lookup of servers by name and keep programs up and running.

There is one User Agent for each user. Access control for personal information is implemented primarily by a user's agent. That is, each User Agent collects and controls all the personal information pertaining to its user and applications can only get personal information from a user's agent—and only if that agent consents. In fact, a user's agent can lie about the user's personal information, because consumers of that information have no other authoritative way of determining that information. A user's agent is under the control of the user: he determines the policies implemented by his agent, chooses which implementation of the agent to run (or writes his own), and runs it on one or more computers he trusts.

A User Agent consists of several modules, some of which perform system infrastructure functions, and some of which are responsible for implementing the agent's responsibilities for specific applications. User Agents are the locus of control for customizing applications with respect to their environment. That is, knowledge about the user's environment and context and his preferences with respect to various circumstances are maintained in the User Agent and propagated from there to the user's various applications. Thus, the User Agent serves as a general policy coordinator for both the user's privacy concerns as well as his context-sensitive customization concerns.

Each User Agent collects location information from a variety of sources, examples of which might include:

1. infra-red-based active badges[8, 7],

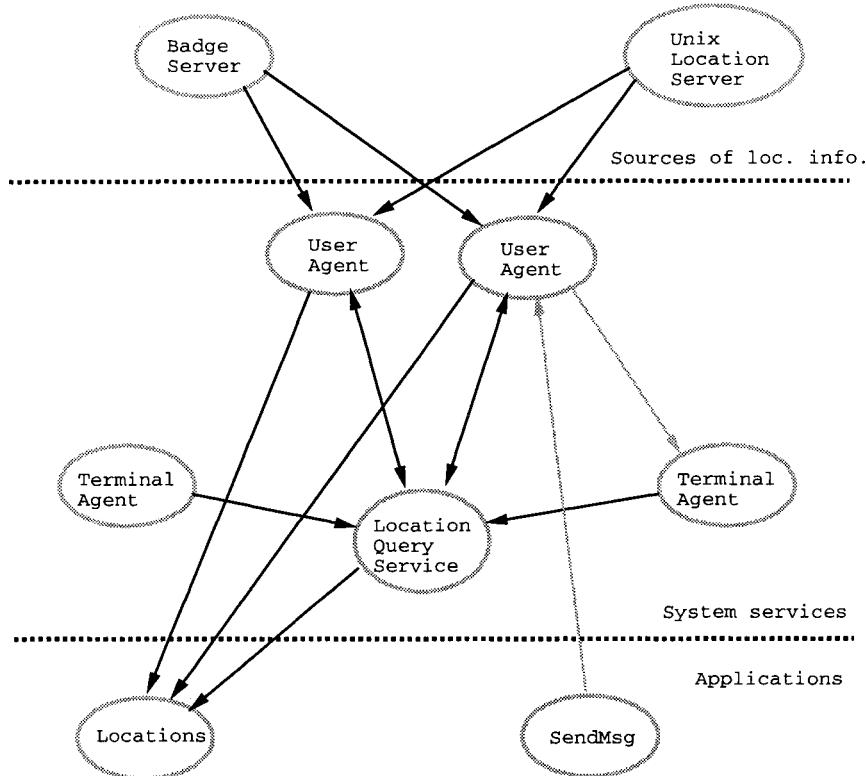


Figure 2 Basic system architecture, with an instance of UMD message delivery illustrated by the gray arrows.

2. wireless nano-cell communications activity[9],
3. global positioning system information[1],
4. device input activity from various computers[6],
5. motion sensors and cameras[3], and
6. explicitly specified information obtained directly from human beings.

Our existing system employs 1, 4, and 6. The badges in our system emit a unique id every 15 seconds and a Badge Server in each region passes data obtained from polling its badge sensors to interested parties. Each User Agent

registers for information about the badge id representing its user; the correspondence between a badge id and a user's identity is known only by the user's agent.

In a similar fashion, the Unix Location Server polls the **rusers** daemons on our Unix workstations and passes data on each user to his User Agent. Users may also inform their User Agent of their current location explicitly by running the **AtLocation** program. Each User Agent synthesizes the (possibly inconsistent) location hints it receives from the various sources into one opinion.

A User Agent is a well-known service that external clients can ask for a variety of information about the user it represents, such as the user's current location. The User Agent will either honor or reject any request depending on the policy its user has specified. Subject to the user's policy, the agent also makes its user findable by location using the facilities described later.

Some applications work by simply interacting with User Agents. For example, submitting a message for ubiquitous delivery consists of simply giving the message to the User Agents for the recipients; each agent then takes care of getting the message to its user. Other applications, like Scoreboard, Responsive Environment, and FindNearest, are primarily concerned with a given location, and must perform some kind of a query to find the agents for the people at or near that location. Sometimes User Agents themselves need to perform location-based queries—for example, to find nearby terminals through which to present a message for ubiquitous delivery.

One of the key problems addressed by our architecture is how to keep disclosure of the association between a person's identity and that person's location under the control of that person while supporting our applications with reasonable efficiency. Our architecture provides control through the User Agent: (1) an application starting from a person's identity can only discover that person's location by asking the person's agent, and (2) an application starting from a location must use the Location Query Service (see below) to discover the identities of the people at that location, and the Location Query Service is designed to give User Agents control over the circumstances of the release of that information.

The Location Query Service (LQS) provides a way of executing location queries that offers different trade-offs between efficiency and privacy. It is built around the idea of queries over *located objects*. A located object is represented by a tuple consisting of a location, an RPC handle, and an association-list describing the object's type and other information that the object chooses to make

available.¹ Examples of located objects include users (represented by User Agents) and terminals (represented by Terminal Agents). A query is a predicate over a location and an association-list; the result of a query is the set of tuples that satisfy the predicate.

A key feature of the LQS is that located objects can be anonymous. That is, a tuple's association-list may reveal only its type and its RPC handle may employ techniques such as indirection through trustworthy intermediaries to hide the true identity of the real server behind the handle. A client getting a query response listing a tuple with an anonymous RPC handle and no identity in the association-list would have to use the RPC handle to ask the object for its identity.² That object (e.g., a User Agent) can respond truthfully, falsely, or not at all, depending on its policy (which might, for example, require authenticating the caller).

Note that a located object could register several tuples for itself, in order to make traffic analysis more difficult.

The LQS is organized by regions, with a centralized server, called the LocationBroker, running in each region. Public objects whose identities and locations are not meant to be kept secret—such as printers and office display terminals—register a full description of themselves in the LocationBroker covering the region they inhabit. A private object—such as a User Agent—who is willing to reveal that *someone* (without revealing who) is at their current location, registers itself in the appropriate LocationBroker in an anonymous fashion.

Each region's LocationBroker also supports standing queries: a client can submit a query and a callback RPC handle, with the LocationBroker notifying the client of the change whenever the answer to the query changes. This is used, for example, by the Locations program to monitor a given area.

A final efficiency trade-off that LocationBrokers can provide is to implement access control on behalf of an object. This amounts to selectively returning a tuple, or portions of its association list, in the results of a query according to a policy specified by the object when it registers itself. An object using a region's LocationBroker thus has the choice of (1) registering minimal information (location, type, and anonymous RPC handle) with the LocationBroker and implementing access control entirely on its own, (2) using (and thus trusting) the access control functionality of the region's LocationBroker, or (3) any com-

¹Object type is used indicate which RPC interface to use with the RPC handle.

²In a similar way, a client can hide its identity by issuing its queries from an anonymous RPC handle.

bination of the previous two.³ Note that for regions where most User Agents are willing to entrust their access controls to the region’s LocationBroker, that region’s LQS has essentially become a centralized design, with all the efficiency benefits and privacy risks that implies.

The last piece of our architecture concerns I/O devices. There is one Terminal Agent for each “terminal”, or cluster of I/O devices that operate together (for example, a workstation’s keyboard, mouse, screen, and sound card comprise one terminal). As with the User Agent, the Terminal Agent consists of several modules, some infrastructure and some application-specific. The agent provides access through a device-independent interface, and manages the multiple demands on the terminal.

Because terminals have owners, and are dedicated (in some cases) to specified uses, there are also policy decisions to be made by Terminal Agents. Agents for non-mobile terminals register in the LocationBroker, so that they can be found by location. A mobile terminal may be dedicated to a particular user and might communicate directly with that user’s agent instead.

2.3 Application Examples

To illustrate how applications make use of our system, we describe how two representative applications are implemented: UMD and the `Locations` program. Someone wishing to send a message for ubiquitous delivery to a user can invoke the `SendMsg` program to submit a message to the user’s User Agent. The User Agent keeps track of which (personal) portable computing devices its user is currently using as well as what “public” terminals and people are near the user’s current location. The latter is achieved by registering for callbacks with the LQS for the user’s current location. When a message is submitted to the User Agent for delivery, it checks to see if the user’s current situation allows delivery of the message (for example, the user’s policy profile may specify that only priority messages should be delivered when the user is in the presence of other people) and if a suitable terminal is currently available. If so, it sends the message to the terminal’s Terminal Agent; otherwise it waits until the user’s circumstances and/or location change and then tries again.

More than one terminal may be available—for example, if the intended recipient is in their office, they might have access to both their workstation and a

³Since our interest is in exploring what happens when servers are not trusted, we have not implemented access controls in our LocationBroker.

portable paging device, if they are carrying one. In this case the User Agent picks the most appropriate one, where appropriateness depends on terminal characteristics as well as whether the message to deliver is marked private—and hence shouldn't be delivered to terminals whose display might be publicly visible (as is the case with workstations). Terminal characteristics are exported in the association-list that a Terminal Agent includes when it registers with the LocationBroker (or User Agent if it is dedicated to a particular user).

In a system with many users, the **Locations** program needs to be told which users to display information for. One way is to provide an explicit list of user names. In this case the **Locations** program contacts those users' agents and requests to be kept apprised of any changes in their users' locations (assuming they consent).

Another way to limit things is to have the **Locations** program show all users within a specified physical area. Consider what happens when the area fits entirely within an LQS region. The program issues a callback registration to the region's LocationBroker, asking to be notified of any changes in the area due to User Agents. All User Agents currently registered in the area with the LocationBroker will have their registrations returned in the LocationBroker's initial response to the callback registration. Any User Agents whose users enter the LQS region at some future point in time and register themselves in the area with the LocationBroker will have their registrations returned to the **Locations** program via callback notifications. Similarly, whenever a User Agent leaves the area or changes location within it, a callback will be made to notify the **Locations** program. An area that intersects multiple LQS regions can be handled by performing the above operations in each region.

3 DESIGN CONSIDERATIONS

3.1 Design Principles

As discussed in the previous section, perfect privacy guarantees are difficult to provide. Consequently we have designed a system that allows users to trade off increasing levels of privacy risks for increasing levels of location-based functionality and efficiency. The following three implementation principles guided our work:

- Structure the system so that users have both the ability and the choice to not be visible to the various sensor networks and servers in the system.
- Avoid requiring personal information to be placed in servers (which may be in untrusted administrative domains).
- Use encryption and anonymous handles to limit the kind of information being revealed.

Ideally, sensing systems such as active badges and activity-monitoring OS services would establish secret and authenticated communications channels with their users' agents. Unfortunately the technologies we currently use (Olivetti active badges and the SunOS **rusers** daemons) do not allow us to achieve this goal. Our active badges are simple fixed-signal, infra-red beacons whose emissions must be gathered by a centralized polling server. Querying **rusers** daemons suffers from the same problem and, even worse, from the fact that Unix makes this information available indiscriminately.

Use of these facilities is acceptable in a friendly environment, but would not be if we extended our system to a larger, more heterogeneous setting. Only the ability to remain silent—for example, by not wearing one's active badge—can ensure that corrupted servers and traffic analyzers will not be able to determine the identities of persons entering their domains.

An interesting unsolved problem is the question of knowing which sensor systems are actually present at a given location. For example, most people in our lab were originally unaware that the **rusers** daemon runs on their workstation by default. Similarly, most people do not think about the fact that many of their monetary transactions implicitly reveal their current locations to the merchants and banking agencies that are party to each transaction.

The potential for inadvertently revealing one's identity and location can be reduced by employing anonymity. Examples include the use of multiple, anonymous login ids and facilities such as anonymous electronic cash[2]. Similarly, applications such as **Scoreboard** only need profile information; they do not need explicit identity information. In our design we rely on anonymity to bridge the gap between wanting to keep location information hidden in a decentralized collection of User Agents and needing to provide some means of performing location queries. User Agents can also use anonymous handles to exercise control over which callers can discover any given piece of information (by choosing how to answer based on the caller's identity).

Queriers, which may themselves be User Agents, can also use anonymity. If both the querier and the responder are initially anonymous and unwilling to reveal themselves without further identification from the other then some additional mechanism is needed to negotiate what to do. We have not explored this topic yet.

3.2 Tradeoffs

Anonymity is not always sufficient to preserve privacy. Simply keeping track of how many people are at each location over time can reveal some strong hints about who has been where. The use of multiple, changing anonymous handles and background noise can obscure this information, but there is a long chain of measures and countermeasures that can be taken. One could even be concerned with detection of minute analog differences between individual devices.

A user must keep in mind is that there are actually several different ways that the privacy of his location can be compromised:

- Application-level operations (such as giving a message to a Terminal Agent for delivery) with untrustworthy parties can reveal location information.
- Location information directly published through the LQS is available to any querier.
- A LocationBroker might not faithfully implement the access controls it offers.
- The intermediaries used to implement anonymity might be corrupt, or not competent enough to foil the attacker.
- Traffic analysis of LQS queries or results might reveal the identity of otherwise anonymous queriers of, or objects in, the LQS.
- The various location sensing systems that gather location information might deliberately give it to other parties.
- The communications between the location sensing systems and the User Agent might not be secret and authenticated.
- The communications between the person's portable computers and his processes running at fixed locations (e.g., a User Agent) might not be secret and authenticated.

Our architecture gives users choices to limit which of the above potential exposures apply. Different potential exposures involve trusting different system services to different degrees. We must allow users to opt out at whatever level their trust is exceeded. Thus, (1) a User Agent might register a single anonymous tuple in the LocationBroker; (2) a User Agent might register multiple anonymous tuples in the LocationBroker; (3) a User Agent might not register in the LocationBroker at all; and (4) a user might disable transmissions from his portable devices (i.e., receive only—or turn them off if he's concerned about noise emissions) and refrain from identifying himself to fixed devices. Thus it is important for the system—including applications—to be tolerant of missing or inaccurate information. Uncertainty of location information (and other personal information) is now fundamentally a part of the system at every level.

It would not make sense (in a real system) for a user to give up a lot of efficiency or functionality protecting against one potential exposure while not protecting against another that applies in the same situation. For example, in a completely untrusted administrative domain, one has to assume that any of the domain's services (LocationBroker, Terminal Agents, location sensing units) could be corrupted. The unfortunate consequence of all this is that the users of a system must stay aware of who controls which aspects of the system they are currently using and must act in an accordingly consistent fashion.

Our architecture is not dependent on the exact nature of the location sensing technologies, nor the communications media, employed. For example, while the experimental system we've built to explore our design extends an inconsistent level of trust—it uses anonymous registrations in the LocationBroker, even though our active badges and Unix workstations reveal location information indiscriminantly—we were willing to accept this because known techniques could be used to provide more secure location sensing facilities.

As one possible replacement, consider using portables that have a GPS receiver and a cellular telephone. The portable could get GPS-resolution information to the User Agent, while exposing only cell-resolution location information to only the phone companies involved (if the user assumes nobody is going to use analog techniques to locate his transmitter more precisely). As another possible replacement, consider putting location beacons in each room, and having an active badge that relays a received location to its User Agent by sending an encrypted message through a chain of intermediaries. In this case the User Agent gets room-resolution location information, trusting the intermediaries to hide the link between location and identity, and revealing that *someone* is in the room to anyone that can see such an agent-bound message and discover its room of origin.

A User Agent trades privacy against functionality when choosing how much information to reveal to which other parties. A completely mistrustful User Agent cannot be found by FindNearest or Note Distribution, cannot customize a Scoreboard or a Responsive Environment, and cannot ubiquitously deliver a message or participate in a Media Call.

In addition to allowing trade-offs between privacy and functionality, our system allows users to make trade-offs between privacy and efficiency. The LQS offers a User Agent three choices in this regard. The first choice is how much information to include in the association list describing the agent. When all the information needed by an information-seeking application is found in the association list, the application need not contact the agent to complete its job; including this information thus increases efficiency, at the privacy cost of perhaps indiscriminately revealing that information.

The second choice offered by the LQS is between registering one or several tuples for a located object. Use of several tuples will cause the User Agent to appear in more query results and hence have to answer more follow-up questions from potentially interested clients.

The third choice offered by the LQS pertains to the use of access controls within the LocationBroker. If most User Agents participating in the LQS of a region are willing to completely trust the region's LocationBroker then certain kinds of queries can be made much more efficient. In particular, queries where the set of User Agents that will appear in the result cannot be well approximated on the basis of location alone will benefit from this optimization. For example, consider querying for the k users in some set of locations whose names are alphabetically nearest one given name (such as might appear in the middle of a scrolling list). Before the final k can be chosen, all agents registered anonymously at those locations must be queried individually for their names.

3.3 Alternatives

An interesting addition to our architecture to consider is the use of multicast. One could imagine having a multicast group instead of a LocationBroker for each LQS region and having clients of the LQS multicast their location queries to all members of a region's multicast group. Interested parties, such as the User Agents of all users currently in a region, would anonymously listen to the region's multicast group to hear location queries. They would answer a query if they matched it and if their current privacy policy allowed it.

The advantage of using multicast is that it only reveals the association between an RPC handle and the region, and that only to the multicast routing infrastructure. In contrast, using the LocationBroker reveals the association between an RPC handle and a specific location. The disadvantage of multicast is increased computation and communication: location queries go to, and must be processed by, all listening objects in the *region* instead of just the LocationBroker, and the cost of multicasting to User Agents located somewhere on the Internet may be substantially more than the cost of communicating just with the LocationBroker. Note also that we require *reliable* multicast for the design just described.

One way to address the inefficiency problems of multicast is to offer it as an option, in addition to the option of using a LocationBroker. Thus, each LQS region could maintain both a multicast group and a LocationBroker, with the LocationBroker listening to its region's multicast group and processing all location queries against the objects that are registered in it. User Agents would thus have a choice between listening to a region's multicast group for greater privacy or registering with a region's LocationBroker for greater efficiency.

Unfortunately, if multicast is unavailable then clients have no choice but to register with the LocationBroker (if they wish to be findable) and accept the increased risk that implies. Note also that a User Agent wishing to listen to a region's multicast group must be able to register in the multicast group from whatever address it (or the last intermediary of its anonymous indirection chain) has. Such functionality is not yet widely available in the Internet.

There are other, radically different, architectures one could design to protect one's privacy, but these do not support all the applications we are interested in. For example, if all we cared about were visitor guidance then a simple scheme whereby each location contains a broadcast location beacon that could be received by nearby portable devices would suffice. Such a scheme would provide strong privacy guarantees as long as no portable device tried to communicate with the rest of the world. Another alternative design could be constructed around "proximity-based" communications and sensing facilities. These could be used to enable communications and presence detection among objects at a particular location without requiring more wide-ranging communications that would be easier to monitor by external parties. Such facilities, combined with portable computer devices, could be used to implement things like Scoreboard, in-room note distribution, and in-room window migration. However, finding out about things beyond one's immediate proximity—as is needed by the *FindNearest* and *Locations* applications would not be possible.

4 STATUS AND EXPERIENCE

Our location infrastructure is built and currently deployed within part of our lab with 12 User Agents, 21 Terminal Agents, and one LocationBroker running. The active badge system includes about 120 infra-red sensors that are deployed in about 70 offices, 10 common areas and lab rooms, and the corridors interconnecting them.⁴ This represents about 10% of the entire floor plan of our building. Thus, the people participating in our system are still outside the system a considerable amount of time; both during the work day and outside of it.

As mentioned in the introduction, the two applications currently in use in our system are the **Locations** program and **UMD**. The more popular one is the **Locations** program, which people tend to keep running all the time in a background window on their workstations. Its primary use seems to be as a quick “hint” reference source to know if someone is currently in or where they might currently be. When run with the **-map** option, the program also provides a convenient map of the lab instead of just an alphabetically sorted list of (name, location) pairs. However, the map output option also takes up more screen space, making it less popular as a permanent background facility.

The **UMD** application manages to successfully deliver about 63% of all submitted messages to their intended recipients within one minute and about 73% within five minutes.⁵ Unfortunately, these statistics are not very informative because they are dominated by the fact that our users are not always near a display terminal and are frequently outside the range of our system altogether. As a consequence, messages may require a considerable time before being delivered, even when the basic location tracking system is functioning perfectly.

Our experience with **UMD** verified our expectation that recipient context is very important. Originally, we silently popped up a window when a message was delivered to someone. However, because our terminals run screen savers when they aren’t in active use, many offered messages didn’t get noticed. Unfortunately, when we added an audio beep to announce message delivery, we found that message delivery was perceived as being intrusive if the recipient was with other people. This was especially the case if an unimportant message was delivered to the electronic whiteboard of one of our conference rooms during a meeting. Although **UMD** provides mechanism for implementing context-sensitivity, it

⁴Our infrastructure is a follow-on to an earlier and simpler, but more widely deployed one.

⁵Successful delivery means that the recipient explicitly acknowledged receipt of the message.

is still unclear what policies are desirable to effect ubiquitous message delivery in both an effective and a socially desirable manner.

In the remainder of this section we describe the data we have gathered concerning how well our system is able to track people.

4.1 Active Badge Tracking System

Our badge system consists of strings of infra-red sensors, mounted in the ceilings of rooms and corridors, that are periodically polled by programs running on workstations. Offices and corridors typically have one sensor installed in them, while common areas and lab rooms have between two and four sensors installed. Our installation includes three separate strings of sensors; each attached to a different workstation. Each of the three poller programs feeds its raw data into the Badge Server, which then forwards the appropriate parts to each User Agent that has registered with it.

Our badges emit a fixed-id signal every 15 seconds and it takes roughly 2 to 3 seconds for each poller program to interrogate all the sensors on the sensor string it is responsible for; this implies that the minimum temporal resolution of our system is about 15 to 18 seconds. In order to get a handle on how reliably the infra-red sensors manage to detect badge emissions, we have structured the data presented in this section around the notion of a “sighting interval”, which is the time between subsequent sightings of the same badge (or a person’s input activity in the case of the `rusers` data presented later on). If badge emissions are reliably detected by the sensor system then the average sighting interval for a person, while they are in the area covered by the sensors, should be around 15 to 20 seconds. Longer sighting intervals will occur when a badge’s emissions are missed by the sensor system.

Because people are frequently not within the area that the badge sensor system covers, we have applied two heuristics in this paper to account for absences. To approximate the working day, we only consider badge sighting intervals between the first sighting of a day and the last sighting, with days considered to end at 2AM. While this heuristic does the wrong thing for people who work at 2AM, none of our subjects fall into that category. We have also tried to filter out periods when someone leaves the badge sensor area to go to another part of the building or to leave the building during the “work day”. This is done by excluding from consideration intervals longer than an upper bound;

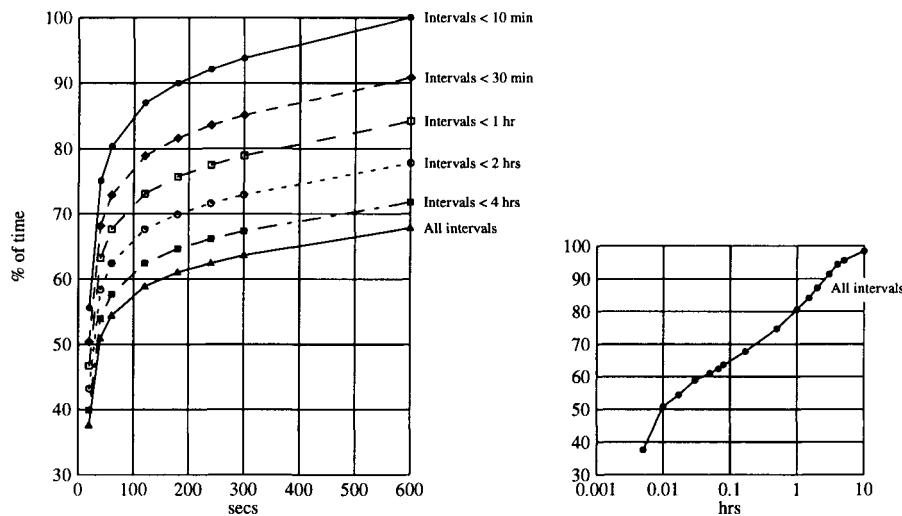


Figure 3 Cumulative graphs of badge sighting intervals by interval length, with various upper bounds on interval length considered.

we consider several different values for this upper bound because other effects (such as obstructing a badge's emissions) can also produce long intervals.

Figure 3 shows cumulative time graphs of badge sighting intervals by interval length, with various upper bounds on interval length considered. The curve for all intervals (within a "working day") is shown extended out to 10 hours in the side plot. Each point on a curve represents the total amount of time spent in intervals shorter in length than that point's x axis time value as a fraction of the time spent in all intervals considered for the curve. Note that over the region shown by the main figure, the curves are actually just scaled versions of each other because the sum of the interval values used for any x axis point is the same for each curve. The purpose of showing multiple curves is to give the reader some idea of how the (same) data looks as we apply ever-more-aggressive versions of our heuristic for filtering out intervals during which a person is outside the system.

When all intervals during the "working day" are included then short sighting intervals account for a distressingly small percentage of the time. Accounting for likely absences improves the numbers but still leaves significant periods of time during which a user is sighted only after a lengthy interval.

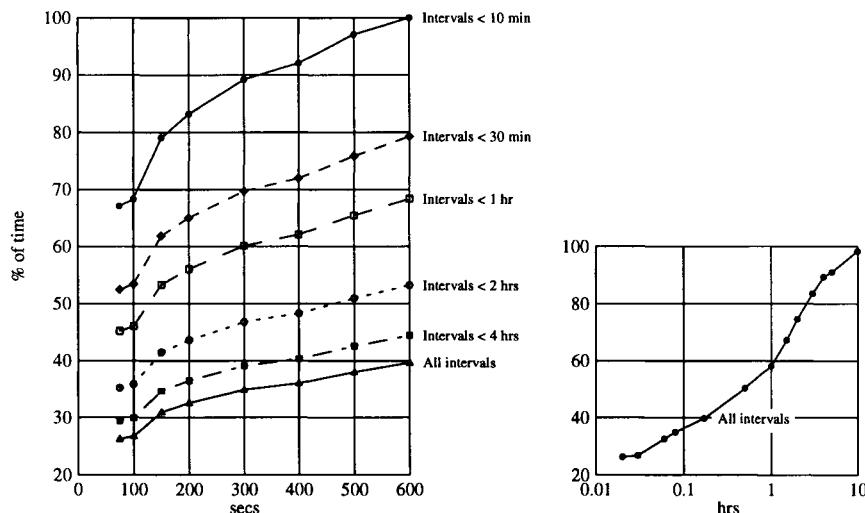


Figure 4 Cumulative graphs of computer input sighting intervals by interval length, with various upper bounds on interval length considered.

When the badge sighting data is broken down by person, considerable variation is seen between people. For example, the percentage of time spent in intervals less than 20 seconds long varied from 9% to 63% for the “all intervals” cases. When intervals greater than 1 hour are thrown out then the time spent in intervals of less than 20 seconds varied from 12% to 78%. These variations seem to be due to a variety of factors, such as time spent outside the badge system area during the day, whether or not a person wears their badge all the time, and how “visible” their badge is to sensors. The latter issue is problematic for several reasons:

- Both natural light and some of our ceiling lighting interfere with the sensitivity of our IR sensors.
- Many people prefer wearing their badge on their belt rather than pinned to their chest. Unfortunately a belt-worn badge is frequently obscured by a person’s arms or other objects; especially when they are seated.
- Our offices typically have only one sensor in them, yet people tend to face different directions when performing different activities. A common example is working at a computer versus talking with a colleague.

One of the questions we had with using an infra-red-based badge system was how often multiple sensors would see a badge at the same time. This can occur in large rooms containing multiple sensors, at corridor intersections, and for glass-walled offices that happen to have a corridor sensor outside them. Our system has multiple-location sightings about 0.2% of the time; with the bulk of them occurring in our meeting rooms and lab rooms containing multiple sensors. Note that multiple sightings are not a problem for our architecture since uncertainty is part of the system in any case.

4.2 Computer Input Tracking System

The basic design of the Unix Location Service is the same as that of the Badge Server: the Unix Location Server polls the **rusers** daemon on each workstation in our lab once every 60 seconds to find out when the most recent input activity occurred and which user login id it occurred for. The results are then forwarded to the appropriate User Agents. Figure 4 describes the data we have gathered for this service.

The **rusers** data displays similar characteristics to that of the badge system. That is, the time spent in intervals of small size represents only a small fraction of the total time spent in all sighting intervals, with the fraction significantly improving as larger intervals are excluded from the data. The breakdown by person again yields significant differences due to different people's work patterns.

4.3 Overlap Between Tracking Systems

One of the most interesting things we observed about our two tracking systems is that they tend to complement rather than overlap each other. Table 1 lists how often only a person's badge was seen, only a person's computer input activity was seen, both were seen, and neither were seen, as a fraction of the total time that the person is in the system. A person is considered to be "in the system" when they are not absent from the badge data and not absent from the input activity data. As before, we define a person to be absent from sensor data during intervals longer than various bounds. We define the notion of a person being "seen" during a sighting interval as meaning that the length of the interval is less than some cut-off value. The table gives overlap statistics for several different absence bounds and "seen interval" cut-off values, the smallest cut-off value being set at a value slightly larger than the minimum sighting

Seen interval cut-off size:	75 sec.	150 sec.	5 min.	10 min.
All working-day intervals:				
Only badge seen:	19%	20%	21%	21%
Only input activity seen:	17%	20%	21%	23%
Both seen:	7%	9%	10%	13%
Neither seen:	57%	51%	48%	43%
Only intervals less than 1 hr. considered:				
Only badge seen:	26%	26%	28%	29%
Only input activity seen:	23%	28%	29%	32%
Both seen:	9%	12%	14%	17%
Neither seen:	42%	34%	29%	22%
Only intervals less than 10 min. considered:				
Only badge seen:	32%	33%	35%	36%
Only input activity seen:	31%	36%	38%	43%
Both seen:	11%	15%	17%	21%
Neither seen:	26%	16%	10%	0%

Table 1 Table of badge and computer input activity sighting overlap statistics.

interval of either tracking system. The important thing to note is that the fraction of time during which both badge and input activity are seen is quite small, both in absolute terms and relative to the fractions of time during which only one or the other was seen.

We attribute this phenomenon to primarily two things: (1) people working at home will be seen by their computer input activity and not by the badge system, and (2) people who wear their badge on their belt and are typing at their workstation will tend to obscure their badge's emissions while having clearly visible computer input activity.

4.4 Tracking Moving Persons

People sitting within their office provide a different sighting profile to the active badge system than do people who are moving around. To get a handle on how well our active badge system could track moving persons—such as visitors—we performed several “walk-about” experiments to see how well the badge system could follow us.

As mentioned earlier, the basic badge sighting interval is 15 seconds; which is enough time to walk past about a half dozen offices and perhaps a corridor or two in our lab. We found that a person who randomly walked about our corridors was seen, on average, every 22 seconds, with a standard deviation of 17 seconds. We also tried the same experiment with the badge emission period changed to 10 seconds and obtained an average sighting interval of 17 seconds, with a standard deviation of 13 seconds.

Note that 17 seconds is still enough time to add noticeable inaccuracy to an application such as **Visitor Guidance**. Decreasing the badge emission interval to 5 seconds would presumably give us an average interval length somewhere between 5 and 10 seconds; but would cut the battery lifetime of our badges from its current value of about 3 months to about 1 month.

We did not have much trouble with people being sighted in offices while walking past, even though roughly half, on average, of the front wall of each office in our lab is open to IR. Only about 11% of the sightings from hall-walking experiments were in offices. While we recognize that the exact placement of a badge sensor within a room can greatly affect this result, we mention it because our sensors were placed in a fashion to optimize office coverage, without much concern about the hall “cross-talk”.

We infer from this that message delivery “chase” effects while people move about should probably not disturb the denizens of every office a person walks by. Anecdotal evidence has corroborated this — only one person has reported seeing an attempt at ubiquitous message delivery in his office for someone not present.

5 CONCLUSIONS

We have designed and built an infrastructure for providing location information and various applications that use that information. The architecture we advocate is a user-centric one in which the personal information for each user—including location information—is managed and controlled by that user’s User Agent. A Location Query Service consisting of a LocationBroker per region is provided to facilitate queries by location.

The principle assumptions behind our design were two-fold:

- The design should scale to use in multiple administrative domains.
- We did not rule out the existence of untrustworthy servers and sophisticated traffic analysis attacks in some domains.

The consequences of these assumptions were that we could not use strictly centralized designs that rely on trusted location databases and we had to accept the fact that strict privacy guarantees are in general difficult and expensive to provide.

The hybrid decentralized architecture we designed gives each user a range of privacy options that they may dynamically choose from. At one extreme is the ability to simply “opt out” of the system, exchanging any participation in (and benefit from) the system for a fairly strong guarantee of privacy. At the other extreme is the ability to convert any trusted region into an efficient centralized design by simply having everyone register themselves in that region’s LocationBroker with the appropriate access control specifications. In between, are two levels of anonymity that users can choose to assume, depending on the trust they place in a region’s servers and the level of risk aversion they wish to employ.

The price we paid for the decentralized nature of our architecture is increased communications and processing overhead. The worst case occurs for applications like the `FindNearest` and `Locations` programs, which cannot narrow the set of people they are interested in until *after* they have received query responses from many potential candidates. We believe that in practice the additional overhead will rarely be a problem: applications that are continually interested in changing location information can use the callback facilities to obtain incremental updates and “one-shot” applications, like `FindNearest`, are typically not run so frequently as to overwhelm the system’s resources. Our personal experience, to date, has borne this out.

Two important qualitative implications of our architecture are the following:

- Uncertainty is a *fundamental* aspect of our system that is visible at the applications level.
- Only certain kinds of location sensing technology can be deployed if users are to be able to hide from the system at will.

Uncertainty has strong implications for a variety of our applications. For context-sensitive applications, such as `UMD` and `Media Call`, it means that they

must always assume the possibility that invisible people are at any given location unless explicitly told otherwise. For applications such as `FindNearest` and `Locations`, uncertainty means that they must be viewed as “hint” services. Despite this, we have still found location information to be quite useful; with the `Locations` program being the most popular application in our running system.

Our requirement that users be able to completely hide from the system has strong implications for which sensing technologies may be deployed. Users can hide from our active badge system by simply taking off their badge. If cameras were deployed throughout our lab then it would be almost impossible to allow some people to hide from the system while still being able to track others. In general, any technology that can track some unremovable, unhidable aspect of people must be avoided if we wish to allow people to remain hidden from the system.

Our quantitative experience with providing location information has primarily covered the efficacy of the location sensing systems we deployed. We found that both our infra-red-based badge system and our `rusers` computer input monitoring service gave only partial coverage, even when time spent outside the system was taken into account. While we suspect that substantially greater numbers of badge sensors—probably several per office—could significantly improve our badge tracking performance, this would also substantially increase the cost of deployment of the system.

Interestingly, our two sensing systems tended to complement each other rather than being redundant; thus we obtained a real benefit from employing more than one tracking system. An informative extension to our system would be the introduction of radio-based nano-cells[10], which would suffer from a different set of problems than infra-red. It is unclear whether it would be more economical to improve tracking performance by beefing up one of our sensing systems or by trying to deploy additional different ones. We are currently deploying portable notebook computers using nano-cell radio communication, personal communication devices using more advanced infra-red communication, and additional public display devices. These should improve both the coverage and variety of our location information, as well as giving us a greater number of devices through which our applications can interact.

In addition to the question of what accuracy of location information is attainable, there are a variety of open issues that remain to be addressed by future work. Perhaps most important of these is the question of how accurate location information needs to be, given the fundamental uncertainty introduced by

peoples' desire for privacy. Our current coverage is sufficient to enable useful, though imperfect, versions of all our applications to be implemented. Until we have further actual usage experience with our system it will be difficult to tell how much users actually value various levels of privacy versus functionality and how important either accuracy or efficiency considerations will turn out to be.

We are also curious to see what kinds of privacy policies users actually deploy. Our architecture is designed to provide users with a great deal of flexibility and control, but it is not at all clear how much users will actually take advantage of all the options offered them. We suspect that in the long run most users will settle into a small number of usage "modes" that reflect common situations, such as maximum trust and functionality (e.g. being at home), a fair amount of trust and lots of functionality (e.g. most of the time at work), less trust and less functionality (e.g. being at a shopping mall), and no trust with no functionality (e.g. when privately negotiating with someone).

A related policy question to examine is that of how many intrusions users are willing to tolerate in order to improve application performance. For example, applications such as **UMD** and **Media Call** can initiate a preliminary dialogue with a user to verify the social context the user is in. Similarly, an application such as note distribution can employ external feedback mechanisms to verify its successful execution. Active participation by users allows an application to overcome inaccurate and incomplete location information by having users modify their behavior. The result is a system that may provide greater privacy safeguards, but is also more intrusive and less automated than it might be.

In addition to these policy issues there are at least two other infrastructure questions that deserve mention for future work. We listed the addition of multicast to our system as an interesting extension to consider. However, whereas the properties of local-area multicast are fairly well understood, it is unclear what the behavior and scaling properties of reliable Internet multicast are. Consequently, it is unclear what would happen if a large-scale deployment of a multicast-based location infrastructure ever happened.

The second infrastructure problem we mention is that of how two anonymous parties can agree to conditionally reveal information to each other. Although special cases of this problem are easy to solve and may represent the common usage case, it is unclear if there is a general solution that will be satisfactory in all cases.

Acknowledgements

We thank our colleagues with whom we have discussed many of the ideas in this paper, especially Dan Greene and David Nichols, who provided valuable insights and feedback to us in the early phases of the design.

REFERENCES

- [1] N. Ackroyd and R. Lorimer. *Global Navigation: A GPS User's Guide*. Lloyd's of London Press, 1990.
- [2] D. Chaum. Security without identification: transaction systems to make big brother obsolete. *CACM*, 28(10):1030–1044, October 1985.
- [3] S. Elrod, G. Hall, R. Costanza, M. Dixon, and J. desRivieres. The responsive environment: Using ubiquitous computing for office comfort and energy management. Technical Report CSL-93-5, Xerox Palo Alto Research Center, 1993.
- [4] B.N. Schilit, M.M. Theimer, and B.B. Welch. Customizing mobile application. In *Proceedings USENIX Symposium on Mobile & Location-Independent Computing*, pages 129–138. USENIX Association, August 1993.
- [5] M. Spreitzer and M.M. Theimer. Scalable, secure, mobile computing with location information. *CACM*, 36(7):27, July 1993.
- [6] ruser manual entry of the SUNOS UNIX manual.
- [7] R. Want and A. Hopper. Active badges and personal interactive computing objects. *Transactions on Consumer Electronics*, 38(1), February 1992.
- [8] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *Transactions on Information Systems*, 10(1), January 1992.
- [9] M. Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–104, September 1992.
- [10] M. Weiser. Some computer science problems in ubiquitous computing. *CACM*, 36(7):74–83, July 1993.

UNIX FOR NOMADS: MAKING UNIX SUPPORT MOBILE COMPUTING

Michael Bender, Alexander Davidson,
Clark Dong, Steven Drach, Anthony Glenning,
Karl Jacob, Jack Jia, James Kempf, Nachiappan
Periakaruppan, Gale Snow, Becky Wong

*Sun Microsystems, Inc.
Mountain View, California.*

ABSTRACT

Traditionally, the Unix operating system¹ has been associated with deskbound machines tethered to the wall by a power cord and an Ethernet cable. Making Unix support a more nomadic model of computing requires changes in the entire system, from the kernel level through the user command set of applications. In this paper, we present the results of an experimental prototype development effort targeted at supporting a nomadic computing model in Sun's Solaris 2 SVR4-based platform². The development involved enhancements in four primary areas: kernel changes to support power management and checkpointing of system state, drivers and other kernel support for the new PCMCIA bus standard, support for serial line networking, and a new electronic mail application designed specifically for accessing mail over slow serial connections. The paper discusses enhancements and modifications to the design of standard Solaris system components in each of these areas.

Permission has been granted by the USENIX Association to reprint this paper. This paper was originally published in the First USENIX Symposium on Location Dependent Computing, 1993. Copyright ©USENIX Association, 1993.

¹Unix is a registered trademark of Unix System Laboratory

²In this paper, SunOS 5.x refers to SunSoft's implementation of Unix System V Release 4 (SVR4). Solaris 2.x refers to SunSoft's SunOS 5.x-based application development and delivery environment, which includes the OpenWindows window environment and the Tooltalk application integration environment.

1 INTRODUCTION

An implicit assumption in the design of many Unix platform software components is that the workstation or server will remain tethered to a particular physical location by its need for power and an Ethernet connection. Newer, low power chip and system designs have enabled the manufacture of small form-factor, battery-powered portable computers. These computers are much more mobile than the machines on which Unix traditionally runs. Such machines may be used both in an office setting and while on the road, so that they become the primary machine for some users. A *nomadic computing model*, where users set up their computer and work somewhere then move the machine to a different physical location and continue working, is fundamentally different from the traditional deskbound model which Unix currently supports. Design modifications to a Unix platform for supporting nomadic computing require a system approach, since the nomadic usage model affects everything from the kernel through user applications.

The main factor driving kernel changes for nomadic use is power. Mobile computers tend to have tighter constraints on power consumption than desktop machines. These constraints derive primarily from the limited performance of existing batteries. Since battery technology is improving slowly relative to processor and memory technology, hardware and software for saving power in nomadic computers is essential to increasing the available compute time with limited battery life. In addition, nomadic usage patterns involve frequent power cycling, so that long start up and shut down times are less tolerable than in desktop machines. Mobile users want the ability to simply shut the machine off and later start it up again, having it restored to exactly the same state as they left off so they can begin work immediately, because their work is frequently interrupted by movement. While similar functionality has been added to MS-DOS³ and SunOS 4.1 [12], to our knowledge, our work is the first attempt to add power management and checkpoint/resume to a version of SVR4.

Nomadic users may have limited access to the standard Ethernet connections which many deskbound machines require. A wide variety of connectivity options, such as FAX/modem, ISDN, wired/wireless LAN/WAN, etc., may be available at the location where the user is working. In addition, the higher power requirements and large form factor of standard desktop bus extension peripherals are often inappropriate for mobile machines. A new bus standard for low power, small form factor (credit-card sized) hardware devices, called PCMCIA, has been designed specifically for portable computers [6]. Support

³MS-DOS is a trademark of Microsoft, Inc.

in the Unix operating system for PCMCIA is needed to make PCMCIA devices available for Unix nomads. In addition, since many of the connectivity options open to nomadic users are likely be serial lines with relatively low bandwidth and high latency, transparent access to an efficient serial line protocol is required.

Perhaps the canonical application for mobile users is electronic mail. A variety of new connectivity technologies, including mobile cellular and packet radio networks, are enabling wide area access to electronic mail beyond the desktop. A critical factor in providing mobile users with electronic mail service is the communication protocols used between the client and servers. Currently, they are optimized for high bandwidth, low latency connections. Most of the newer, wide area connectivity options are low bandwidth, high latency, with a relatively high cost per packet. New application level protocols are required to deal with the different characteristics of these connections.

This paper presents an overview of changes in various Solaris 2.x system components necessary to support nomadic computing. We discuss adding support for power management, kernel and application checkpointing, PCMCIA, serial line connectivity, and mobile electronic mail. The next section describes the power management framework. Section 3. discusses the design of checkpointing, and how it interacts with power management to provide the user with "instant on" capability. In Section 4. , we discuss the implementation of PCMCIA on SunOS 5.x. Section 5. provides an overview of the serial connectivity and the asynchronous serial line link manager. In Section 6. , we discuss electronic mail as an example of what changes may be required in applications to support mobile users. Finally, Section 7. summarizes the paper.

2 THE POWER MANAGEMENT FRAMEWORK

Limited power availability is the primary constraint on nomadic computer operation. Since a portable computer consists of a variety of hardware devices, conservation of power can be achieved by matching the power consumed to the activity level of each device. The objective of the power management framework is to match the power consumed by a device to its activity level. In its simplest form, the power management framework can turn devices off when they are idle and turn them on again when they are needed. If the device has special power management hardware, the power management framework can

match the power consumption of the device to its activity level in a more fine-grained manner. The result is increased battery life, and therefore increased operating time.

The goals of the power management design are the following:

- Provide a generic, portable power management framework within the SVR4 system design,
- Operate transparently to the user, but include user tunable parameters,

In pursuit of these goals, the framework is divided into three basic parts:

- The device driver mechanisms to support power management,
- The pseudo device driver, **pm-driver**, providing supervisory kernel mechanisms and policies,
- The DDI/DKI [11] changes required to support the framework.

Figure 1 contains a diagram of the power management framework, showing the architectural relationship between the parts, and the parts themselves are described in a separate subsection. In the figure, a light grey arrow indicates that the function is optional, a solid arrow indicates it is required, and a striped arrow indicates the functionality is provided by an implementation dependent mechanism.

2.1 Device Driver Mechanisms

Device drivers are really the key to power management. Device drivers control when a device is physically accessed, whether a device can be suspended, and whether any special power management hardware components are available to the kernel. The power management framework depends on the device drivers to manage the individual devices they control, which is the bulk of the power management. To remain compatible with existing drivers, the framework assumes that it should not try to power manage a device if the driver provides no power management.

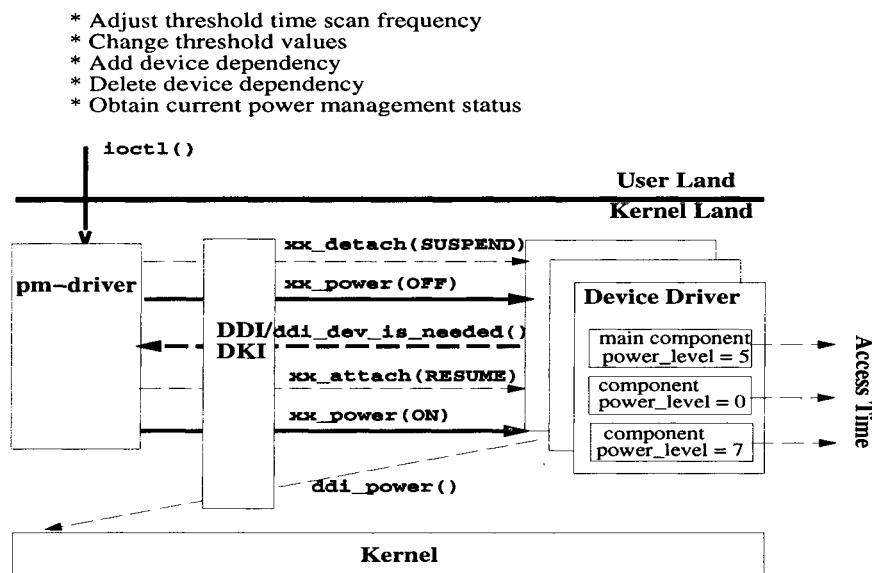


Figure 1 Power Management Framework

The design of the power management framework required adding a new state to device drivers, and rethinking how device driver software relates to the hardware it controls. The new state is device *suspension*. Suspension involves bringing the device to a state where there is no activity, although the power is still on. From a software point of view, suspending the device requires saving hardware registers and driver state to memory. Device suspension is clearly separated from power off, and new driver entry points were added for both suspension and power off. Powering a device off usually involves a writing specific value to a hardware register or issuing a device-specific command to the device.

From the power management point of view, hardware devices can consist of a variety of parts having differing power management capabilities. The extreme example is the entire computer itself, which consists of disk, CPU, display, etc. The power management frameworks models this structure by specifying that a device consists of a number of power manageable hardware units, called *components*. Each component is separately power controllable and has its own set of power parameters. One component is defined to be the main component and is logically where the state of the device is kept. The main component requires the driver to be suspended to save the hardware state before it can be turned off. The driver must also be resumed to restore state when the main

component is turned on. The other components must be stateless and do not require the driver to be suspended or resumed to be power managed.

Components collect together state and functionality relevant for power managing a device. Each component has a power level associated with it. A zero power level indicates that the device is turned off, while a device at a nonzero power level is on. The component is responsible for mapping between power states and integer power levels provided by clients to control power. Each component has two basic functions:

- It must export its last access time. A component indicates that it is busy by exporting a zero access time, allowing the framework to detect when the component does not need power management.
- It must notify the power management framework before it is used, if power has been reduced below the level needed for the operation. The component need not notify the framework if the power level is sufficient for the operation.

In some cases, power management within a driver could conflict with efficient device access. The granularity of power control is entirely up to the device driver writer. Each component of a device may be tuned differently, depending on where in the driver the timestamps are updated. Tuning of a component can assure that critical paths within drivers avoid any overhead at the possible expense of some power management granularity. The framework allows the driver writer to make the necessary trade-offs between device access efficiency and power manageability.

2.2 Power Management Pseudo Driver

The power management pseudo driver **pm-driver** is responsible for managing power reduction in the system. The **pm-driver** periodically scans all devices, querying each power manageable device about its usage, and informing the device to shut off if the device has been idle longer than a set amount of time. Additionally, a power manager interface is provided through **ioctl()** operations within the driver. The **ioctl()** operations allow the user to tune the parameters of the **pm-driver**'s power management policies. The **pm-driver** does not power manage any devices not exporting the correct power management information. The **pm-driver** initiates powering down of a component

due to inactivity, and powering up of a component through a request from the device driver. If a driver is in a state such that it cannot save all the necessary information, it may return failure. In that case, the **pm-driver** does not turn the main component off immediately, but it may try again later.

Through **pm-driver**, the power management framework recognizes two types of dependencies between devices: logical and physical. A power manageable device has a physical dependency if it has one or more devices attached below it on a set of interconnected buses. A device has a logical dependency if no physical interconnection exists, but a power management constraint exists nevertheless. For example, a graphics card driver that relies on keyboard and mouse inactivity to determine if the display can be turned off has a logical dependency. Prior to power removal, all dependent devices must be properly suspended and powered down before the dependee can be shut down. Physical dependency between devices is implicitly defined in the kernel device tree. The **pm-driver** must be informed of logical dependencies either through a driver configuration file or through an `ioctl()` call. If either a physical or logical dependency exists, then no component of the device is powered down unless the dependency is satisfied.

In order to decide if a device component is idle, the **pm-driver** obtains the time that the component was last accessed, which has been exported by the component. If the last access time is zero, the device is busy. If the difference between the last access time and the current time exceeds the management policy threshold time for the component, the component is deemed idle. If all logical dependents of the device are also idle, the **pm-driver** attempts to turn off the device. If an error occurs, the power down fails and the component is left on.

The **pm-driver** provides all kernel mechanisms and implements power management policy. It is accessed by the user through `ioctl()` operations. These operations allow the user to adjust the frequency of device scans, change component threshold values, or add and delete dependencies, and to obtain the current power management status of a device or of the **pm-driver** itself. The **pm-driver** only controls devices that have been configured (i.e. have at least their threshold time(s) set). Typically, at boot time, a script runs a configuration program that reads a system file and configures all power manageable devices.

2.3 DDI / DKI Changes

As shown in Figure 1, two standard operations in the DDI/DKI [11] have been changed to recognize two additional command parameters and three operations have been added experimentally to the DDI/DKI. The changes allow drivers to interact indirectly with the pm-driver, so different power management pseudo drivers can be provided for different hardware platforms, or no pseudo driver if power management is not desired. The changes also permit the implementation of other platform specific power management modules to provide ad-hoc power control for devices which were not initially designed with power control.

The standard DDI/DKI operations `xx_detach()` and `xx_resume()` have two additional command parameters added to their command set, SUSPEND and RESUME. The SUSPEND command parameter requests that the driver save its state, block any further IO, and allow all outstanding IO requests to complete. When it returns, there should be no outstanding requests for IO so that nothing is lost when the device is power cycled. The RESUME command parameter restores the driver's state for a power on. As indicated by the grey arrows in the figure, recognition of these commands by the driver is optional.

`ddi_dev_is_needed()` is a new operation called by the device driver when a component of a device is needed at a certain power level. It returns when the component is at the requested level. `xx_power()` is a new operation that is implemented by the device driver. It takes one command parameter, indicating whether the power should be turned on (ON) or off (OFF), and is called by pm-driver to request that the device driver turn the power on or off. `ddi_power()` is a new operation that is called by a device driver which doesn't want to handle power management itself to request that the kernel handle power management with the system default. The black arrows in Figure 1 indicate that the driver is required to implement these.

3 SYSTEM STATE CHECKPOINT AND RESUME

A major component of the nomadic enhancements is system checkpoint/resume (CPR). A system *checkpoint* saves the state of the entire system, including user-level processes, to nonvolatile storage. A system *resume* restores the system to the checkpointer state at a later time. The primary benefit of CPR for nomadic computing is its role in power conservation. Automatic checkpointing when the

battery is low protects the user from undesirable state loss. CPR also matches the people's expectations about how a nomadic machine should be used. With CPR, a machine can be moved from one place to another without going through the time-consuming bootstrap process. Resume times are typically on the order of a minute or less.

CPR has been available in MS-DOS and on a few portable Unix machines running SunOS 4.1 for some time. However, the SVR4 system is considerably more complex than MS-DOS or SunOS 4.1, so the design issues involved are somewhat more challenging. The design goals for CPR in SunOS 5.x are the following:

- CPR should provide extremely reliable service,
- System administration and maintenance should be as simple as possible,
- The design should remain flexible and extensible so it will easily work with new releases of SunOS 5.x,
- The checkpoint and resume times should be as short as possible, to encourage people to use the facility.

There are two possible design strategies for CPR:

- Forcefully shut off all subsystems and take a snapshot of all kernel and user-level process memory images,
- Ask each subsystem to stop and allow the system to save itself.

Most existing implementations of CPR take the first approach [12]. The first approach is faster but the outcome is less deterministic. For example, an Ethernet controller might be right in the middle of accepting a packet, and forcing it to shut off causes that packet to drop. For this reason, our design uses the second approach. Figure 2 schematically illustrates the checkpoint process, and the next two subsections examine CPR in more detail.

3.1 System Checkpointing

User processes are the first system component to be checkpointed. When a checkpoint is initiated, all user processes are requested to stop. This is accom-

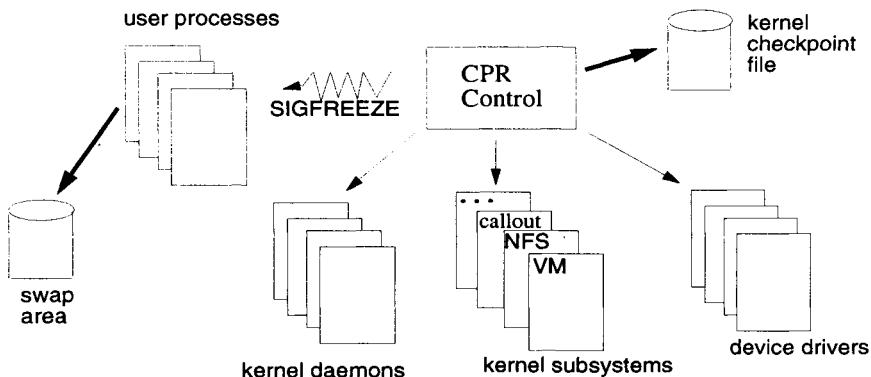


Figure 2 System Checkpoint Process

plished by sending a new user signal, **SIGFREEZE**, to each process. The default action of **SIGFREEZE** is to stop the process. Applications that need a chance to perform certain housekeeping tasks prior to the checkpoint (e.g. close down network connections, notify servers or clients of impending unavailability) can catch the signal. Once the user processes are frozen, all modified pages with backing store are paged out, and all such pages are invalidated and purged from the page cache. User-level processes are restored from these pages on resume.

After all user processes are stopped and their pages are pushed out to the swap device, the kernel daemons are requested to stop. Since each daemon is unique, it needs to stop at a different location. A callback mechanism allows each daemon to specify where it should be stopped. A callback handler can be registered with the CPR callback mechanism by the daemon at thread initialization time if the daemon requires special action prior to checkpointing. The installed handler is part of the daemon and uses the same synchronization mechanisms to stop the daemon.

After all the user processes and kernel daemons are stopped, no further base level activity is possible. The next step is to give all interested subsystems (e. g. VM, callout processing) a chance to prepare themselves for checkpoint. This is accomplished via the same callback mechanism used for stopping the kernel daemons. Only those subsystems that require special processing prior to checkpointing need register callbacks.

Finally, all device drivers are suspended. Although theoretically it would have been possible to design checkpointing so that IO in progress need not complete,

the probability that a restarted IO request would succeed on resume is rather low. Therefore, the checkpoint subsystem blocks all new requests and the kernel waits for all outstanding IO requests to complete. After the IO queue is empty, the driver saves the device hardware and software state to data structures in memory using the same suspension mechanisms as power management suspension. Once all the device drivers are stopped, interrupts are turned off and the system enters a completely dormant state. All valid kernel pages are written out to the kernel state file on the root device (e.g. disk or network) and the checkpoint process is complete.

3.2 System Resume

On the resume side, the kernel loading program started by the PROM detects a resumable system by an implementation dependent mechanism and loads in a special resume module. This resume module reads the kernel state file, and restores the kernel memory image and the MMU back to the checkpointed state. Control is then transferred back to the kernel, and the reverse of the checkpoint sequence is executed. When user level processes are ready to run, a SIGTHAW signal is sent, in case they need to initialize before beginning normal execution.

4 PCMCIA ON UNIX

Unlike desktop machines, portable computers have a limited amount of cabinet space for expansion slots and a limited amount of power for peripheral devices. In response to the need for a smaller form factor and a lower power bus, portable computer manufacturers formed the Portable Computer Manufacturers' Card Interface Association (PCMCIA), and developed a new bus standard, having the same name [6]. The PCMCIA standard defines the physical form factor, electrical bus interface (8 or 16 bit), and a software API for small, credit-card-sized devices. The standard allows mass storage, IO and memory peripherals to be used on multiple different hardware system architectures and operating systems. Although the software interface in the standard was developed specifically with implementation on the Intel x86 architecture and the MS-DOS operating system in mind, any computer which implements a bus interface conforming to the PCMCIA mechanical and electrical specifications can utilize a PCMCIA card. Since the existence of the standard is catalyzing volume manufacture, implementing software support for PCMCIA

on Unix would provide the Unix user with access to a wide variety of previously unavailable low power peripherals.

The goals of the PCMCIA implementation are the following:

- Provide a software architecture and API that adheres closely to the PCMCIA standard without violating the architectural assumptions of the SVR4 device driver architecture,
- Supply enhancements in areas where the standard is currently underspecified or lacking,
- Assure that the implementation supports maximum portability of PCMCIA card drivers between SunOS on different hardware platforms (e.g. SPARC and Intel x86),
- Feed the results back into the PCMCIA standards process to grow the current software interface beyond its focus on MS-DOS.

In the following subsections, we describe the implementation of the PCMCIA 2.01 standard API on SunOS 5.x.

4.1 The Card and Socket Services Design in Unix

The PCMCIA standard specifies the *Card Services* API as an interface between a device driver for a card and the system software layer managing the PCMCIA bus controller hardware, or *adapter*. The adaptor provides the physical slot, or *socket*, for the PCMCIA card and a hardware interface between the PCMCIA bus and the system bus. The Card Services API allows a card device driver to control the adapter independently from the specific adapter hardware. The *Socket Services* API is a hardware-independent interface between Card Services and the software layer that controls a particular adapter. It is possible to have multiple Socket Services layers to support multiple different types of adapter hardware.

Figure 3 illustrates the PCMCIA software driver architecture in SunOS 5.x and its mapping onto the hardware. In the figure, grey rectangles are hardware modules and grey arrows indicate interaction between software components and

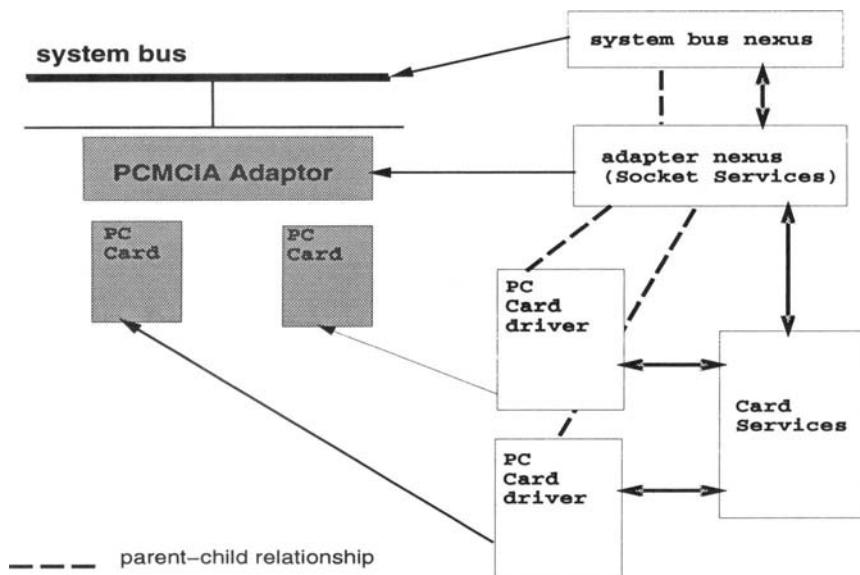


Figure 3 PCMCIA Hardware and Driver Software Architecture

hardware. Black arrows indicate interaction between software components. In the SVR4 architecture, device drivers for bus controllers such as a PCMCIA adaptor are called *nexus drivers* [11]. Drivers for devices on the bus are children of the nexus. Figure 3 indicates parent-child relationships with a striped line. The Socket Services layer corresponds to a nexus driver in the SVR4 architecture, with a separate nexus driver required for each different type of adapter hardware. A particular PCMCIA nexus driver is the parent of all card drivers for cards in that adapter's sockets, and, in turn, the PCMCIA nexus is the child of the system bus nexus. Unlike other device drivers, a PCMCIA card driver does not interact directly with its parent nexus. Rather, as specified by the standard, the card driver calls into the Card Services layer, and the Card Services layer deals with the nexus.

The PCMCIA standard also specifies calls in the Card Services API allowing a device driver to load and unload a Socket Services implementation. The mapping of Socket Services to a parent nexus driver in SunOS 5.x means that this feature cannot be provided on the SunOS 5.x implementation of PCMCIA. Allowing a child to replace its parent nexus is not permitted in the SVR4 device driver architecture.

The SunOS 5.x implementation of Card Services is a collection of function entry points in the kernel. These functions are located in a miscellaneous kernel module that is not part of the device driver hierarchy. The module handles all Card Services requests from all PCMCIA device drivers and makes the calls into the appropriate parent nexus. The Card Services functions require the driver to pass a client handle that uniquely identifies the requesting driver instance. SunOS 5.x Card Services implements this client handle as a pointer to a structure containing the driver's perinstance information. Card Services uses this handle to locate the appropriate parent nexus in an implementation specific way. In the PCMCIA standard, the interface between the parent and the child is publicly specified in the Socket Services interface. In the SunOS 5.x implementation of PCMCIA, this interface is private and not available for use by the device driver itself. However, the interface is available for developers of PCMCIA bus adapter hardware who need to develop a custom nexus.

4.2 Modifications to the Card Services Interface

The specification of the current PCMCIA Card Services API is highly tuned for the MS-DOS operating system. The standard specifies that Card Services has a single function entry point through which all Card Services requests are vectored. This single entry point is specified to require the same number of arguments no matter which Card Services function is being requested. The arguments specified in the standard are of fixed type, and different type arguments for different Card Services calls must be packed as untyped bytes into a variable length byte array. In the current PCMCIA standard, each processor type, processor mode (protected, real, etc.), subroutine call method (near, far) and operating system on which Card Services is implemented specifies an implementation-specific binding that dictates the details required to make the call. The model is very similar to an MS-DOS INT 21 function call.

This design hinders driver portability for operating systems, such as SunOS 5.x, that run on multiple system architectures and multiple processors. As part of Sun's efforts in the PCMCIA standard process, a C language calling convention is being proposed. The proposal implements a single Card Services function entry point with a variable argument list of pointers to function-specific structures and/or opaque data types, somewhat like an `ioctl()`. Details such as the sizes of various address and data types, byte ordering for shared structure members, and generic system resources managed by the kernel are kept hidden from the driver developer. Selective data hiding allows a device driver to be

source compatible between SunOS 5.x on the SPARC architecture and SunOS 5.x on the Intel x86 architecture. Certain differences between the SPARC and Intel x86 architectures, such as card IO port accesses, are hidden using register access macros supplied by the PCMCIA header files. On SPARC, card IO registers are mapped into the device driver's virtual memory address space, while on the Intel x86, the registers are accessed using privileged I/O instructions, but the macros hide this detail from the driver writer.

4.3 The Card Information Structure (CIS) Interface

One of the most important features of the PCMCIA standard is the specification of a mechanism making every PCMCIA card selfidentifying. Every PCMCIA card has an area in which card information is stored, called the *Card Information Structure* (CIS). The CIS is a singly-linked-list of variable-length tuples. Each tuple has a one byte code describing the tuple type, and a one byte link which is the offset to the start of the next tuple in the list. The CIS provides a wide variety of information about the PC card. Each tuple can contain subtuples that elaborate on the information provided by the parent tuple. Although the PCMCIA standard originally only specified enough bits for 256 tuples, as the standard grew, the need for more arose. Since no standard extension mechanism has been proposed, extensions have arisen in an ad-hoc manner.

The only method specified by the PCMCIA standard for dealing with CIS structures is a set of Card Services functions that return the raw contents of a tuple. The device driver is required to take care of tuple parsing itself, even for tuples whose structure is fixed by the standard. The SunOS 5.x Card Services implementation provides the standard Card Services functions, however, it also includes a tuple parser that parses all tuples specified by the standard into structures defined in the PCMCIA header files. A device driver need not include any tuple parsing code itself unless information from a nonstandard tuple is required.

PCMCIA CIS structures are also used by the PROM during booting to build the kernel device tree on SPARC systems. SPARC systems contain a rudimentary CIS interpreter and tuple parser in the body of the boot PROM. This CIS interpreter knows about certain common tuples that provide device identification and configuration information, and the interpreter and parser build simple nodes in the kernel device tree for each card found at boot time. In addition,

card developers can include Fcode, the programming language used by Sun's boot PROM for selfidentifying devices, in the CIS of their cards, allowing the boot PROM to configure PCMCIA cards just as Sbus cards are configured currently. For cards that appear to the system as mass storage or network devices, the PROM can load boot images or establish network connections, allowing the PROM to boot the SunOS 5.x kernel from files read off PCMCIA mass storage or network devices. In SunOS 5.x for Intel x86 systems, using a PCMCIA card as a boot device is not yet supported, since the boot PROM on Intel x86 machines is not programmable.

5 SERIAL WIDE-AREA CONNECTIVITY AND LINK MANAGEMENT

Traditionally on the desktop, the primary connectivity option available to users has been 10 Mbps Ethernet. The most common form of network connectivity available to nomadic users currently is a simple serial modem connection (2400 to 14.4K bps, faster with compression) via an analog telephone line. More recent innovations in mobile connectivity have made alternate options available, such as spread-spectrum and packet radio, cellular, and satellite. One characteristic common to most nomadic connectivity options regardless of their bandwidth and latency characteristics is point to point serial transfer. Support for efficient serial connectivity is therefore essential for nomadic users.

The goals of the serial connectivity design are the following:

- The serial connectivity protocol should slip easily in at the ISO data link layer, and use standard higher level and lower level interfaces,
- Performance of the protocol over low bandwidth, high latency serial networks, such as analog telephone, should be adequate,
- The serial protocol should be accommodated by existing network infrastructure (e.g. routers),
- Connection management should optimize connection time, so that expensive or unreliable connections need not remain active when they are not in use,
- The protocol should contain proper support for security.

These goals lead to the incorporation of the Point to Point Protocol (PPP) [9] [10] [5] [4] into SunOS 5.x, and the development of the asynchronous link manager. The next two subsections describe these new system components.

5.1 Point to Point Protocol (PPP)

Most network applications built for the desktop environment have been engineered for the Ethernet, and specifically for the TCP/IP or UDP/IP protocols. For nomadic users to be able to employ standard network applications, like the Network File System (NFS) over a serial line, the same kind of higher level services available on Ethernet, e.g. TCP/IP and UDP/IP, are required over a serial connection. PPP is a standard protocol developed by the Internet Engineering Task Force Network Working Group for point-to-point links such as dial-up modem servers [9] [10] [5] [4]. With PPP, applications can access the wide-area serial connection through the same system interfaces and higher level protocols used for the local area network.

The PPP protocol consists of three components:

- An encapsulation method for datagrams over serial links,
- A Link Control Protocol (LCP) for the establishment and configuration phases of starting a connection, and for testing,
- A family of Network Control Protocols (NCP's) for connecting over different network layer protocols.

The PPP design specifies an encapsulation protocol over bit-oriented synchronous links and asynchronous links with 8 bits of data and no parity. The links must be full duplex, but can be either dedicated or circuit-switched. An escape mechanism is specified to allow control data such as software flow control (XON/XOFF) to be sent transparently, and to remove control data interjected by intervening software or hardware. Only 8 additional octets are necessary for encapsulation, and the encapsulation can be shortened to 2 octets if bandwidth is at a premium, supporting lower bandwidth connections. The encapsulation scheme provides for multiplexing different network layer protocols over the link at the same time. As a result, PPP can provide a common solution for easy connection with a wide variety of existing bridges and routers. Finally, PPP provides two protocols for authentication: a password authentication protocol and a challenge-handshake authentication protocol.

5.2 The Link Manager

The asynchronous serial line link manager is a user level daemon that takes care of maintaining serial line connections. It automates the process of connecting to a remote host when PPP service is established. The connection can be initiated either by simply sending a IP datagram to a disconnected host, or by receiving notification from a remote host that a connection is desired. In this way, network services such as NFS can be provided without requiring a continuous, physical connection. In Figure 4, the position of the link manager in an experimental serial line architecture is shown. The link manager is normally started at boot time by `init`. It reads a configuration file, builds internal data structures, opens up a connection to the kernel IP-Dialup module (`ipdcm` in the figure), creates and opens the FIFO on which it listens for incoming (dial-in) requests, and then enters an idle state waiting until a message appears on either of the opened connections.

An outbound connection is initiated when the link manager receives a connection request message from IP-Dialup. It checks to see if the connection request corresponds to a configured "path", consults the UUCP data base files for modem and destination system information, and then places a phone call to the destination host. After the physical link is established (i.e. the two hosts are connected over the phone line), the link manager configures and initiates PPP. Once the data link layer is established and the PPP modules on the peer hosts are communicating with each other, the link manager starts IP and, if specified, modifies the local route table to indicate that the newly created path is to be used as the default route. The link manager then passively monitors the connection until an event such as idle time out, line disconnect, or an error condition occurs. When this happens, it will disconnect from the peer, clean up external data structures like the route table, and return to the idle state.

An inbound connection is initiated when the link manager receives a notification that the `login` service has opened the named pipe FIFO. The `login` service is a separate process that is invoked by the `login` program. When the `login` service is invoked, it opens the named pipe FIFO and passes the `login` name and the file descriptor for its standard input to the link manager listening on the other end of the named pipe. It then waits until the pipe between it and the link manager is closed, indicating a disconnect, then exits. This allows `login` to regain control over the serial port again and display the `login` prompts. When the link manager is notified that the `login` service has connected to the named pipe FIFO, it reads the information written to the pipe, checks to see if the `login`

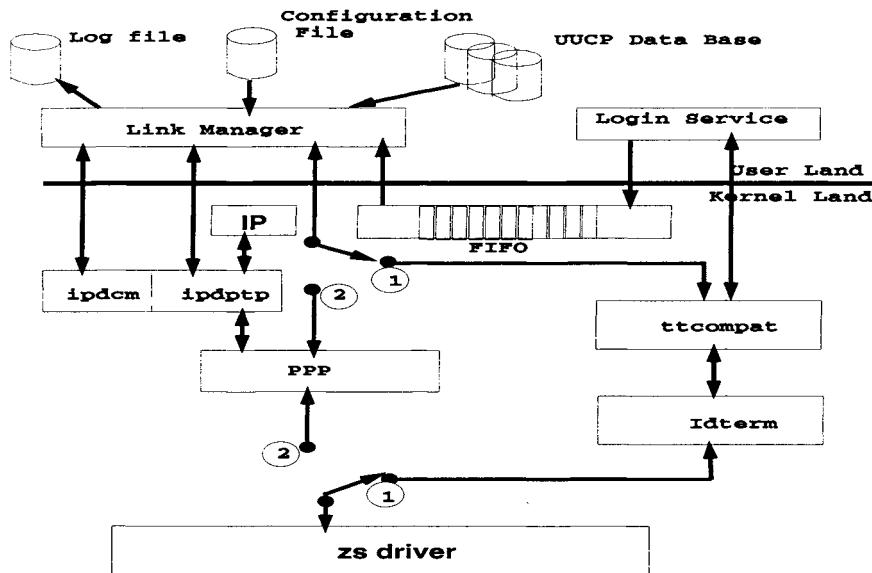


Figure 4 Components and Interfaces in the Link Manager

name corresponds to a configured path, and then configures and initiates PPP. The rest of the scenario is the same as that described for outbound connections.

6 NOMADIC ELECTRONIC MAIL

Electronic mail is the canonical example of an attractive mobile application. The asynchronous nature of electronic mail allows a mobile user to send email from one location and get the reply at another. Mobile users require the same level of electronic mail service as their deskbound counterparts. In addition, since a wide variety of wide-area and local-area connectivity options of varying cost and performance are likely to be available to a mobile user, nomadic access to electronic mail should reduce the need for connectivity unless it is absolutely required, and allow good access even over high latency, low bandwidth connections. For many nomadic users, remote mail may be the single most convenient way of maintaining electronic connectivity, supplanting terminal emulators and file transfer which are the current most widely used options.

Based on these considerations, the following goals for the nomadic electronic mail application were defined:

- Access to electronic mail should be possible over any wide-area or local-area connection,
- It should be possible to manipulate mail state while disconnected from the network, so a user can read mail without requiring a potentially costly wide-area network connection,
- The mail application should be usable even after an unanticipated disconnection from the network or a server failure,
- A user should be able to indicate that they need a certain level of functionality when they are disconnected and the mail application should accommodate this request,
- The nomadic electronic mail application's user interface should differ from the standard desktop UI only in those areas where specific nomadic problems must be addressed.

The following two subsections describe design solutions for these goals.

6.1 IMAP Mail Protocol

The underlying mail protocol used by the electronic mail system plays an important part in determining the connectivity characteristics of the electronic mail application. A variety of currently popular mail protocols and other ways of remotely viewing mail (POP [7], PCMail [3], remote X [8]) were examined and the Interactive Mail Access Protocol (IMAP) [1] [2] was selected. The primary reason for selecting IMAP is because it was designed as a interactive protocol rather than a protocol for simply moving messages from one message store to another. The IMAP protocol makes efficient use of the underlying connection bandwidth by breaking the message into logical chunks, allowing the mail reader to request only the parts of the mail message that the user really wishes to see. For example, if a mail message is in a multimedia format such as MIME, an IMAP-based mail reader could only request the text of the mail message and allow the user to make a choice as to whether or not they wanted to wait for a multimedia attachment to be downloaded.

Although IMAP was a good starting point, its main use to date has been over standard Ethernet and thus its implementation and definition required improvement to support mobility. For example, IMAP sends an envelope structure over the connection for each message, even though the envelope structure is unneeded unless the user actually indicates a desire to reply. To increase the efficiency of IMAP over low bandwidth, high latency wide-area connections, the protocol was modified to reduce the amount of such unnecessary data sent across the connection. In addition, IMAP originally did not support disconnection. Disconnected operation is handled by calculating a checksum on the server to determine if some other mail reader has modified the section of the remotely cached file. Although the checksum should be updated every time an operation affecting mail state is executed, transmitting the checksum separately could cause much higher packet traffic. By piggybacking the checksum on the completion reply packet, the update information is sent as part of the standard request/reply protocol, eliminating the need for a separate request/reply to handle the checksum. At the moment, the mobile IMAP does not support full functionality while disconnected, however. For example, full support of searches for text in the body of a message is impossible unless the body of the message is local or the connection is up.

6.2 Nomadic User Interface Enhancements

For the most part, the user interface of a nomadic mail reader application need differ little from one for desktop use. Indeed, close user interface compatibility between the mail reader for nomads and for the deskbound means that users need not learn a new command interface when they take their machine on the road. *ROAM* is a prototype nomadic mail reader similar in basic function to the standard Sun reader *Mailtool* but with a Motif interface. Besides those places where Motif and OpenLook differ, the command set is almost the same as *Mailtool*.

The exceptions are where additional command support for remote or disconnected operation is required. A command interface allows messages to be downloaded for caching on the local machine, so that users can view mail while disconnected. Users may also want to know which messages are local and which are remote, to avoid an unpleasant surprise while disconnected when they attempt to read a message which is not there. Headers of messages that are not locally cached appear differently from those that are only available over the network, and message bodies that are not local are displayed as half full icons. Locally cached messages have their headers displayed in roman font, while re-

mote messages are in italic. Another user command allows the mail queue to be manipulated, so that unsent messages can be deleted if desired. Finally, users who are connected over wide area serial links need the ability to specify more stringent searches to reduce the amount of traffic and number of misses when trying to find a particular message. Detailed search capability was added including search by dates, text in message body, status of messages and many others. Using this search capability, a user could search for a message received this week concerning a certain topic eliminating the large number of messages that would appear if all messages received for this topic were searched. The search is then run on the server, rather than downloading all message bodies to the client and running the search locally.

7 SUMMARY

Fostering a more nomadic usage model in Unix requires changes throughout the entire system, from the kernel to the commands of user applications. The primary and most important change in the kernel is automation of power monitoring and control, to reduce the drain on batteries. Power management increases battery life, allowing users to work longer without having to plug into wall power or change batteries. System checkpoint/resume provides ease of use in such areas as quick start and reliability. With system checkpoint/resume, a mobile user can quickly shut down operation at one physical location and start up at another without requiring a lengthy initialization.

The new PCMCIA standard presents an opportunity for Unix-based machines to leverage off of a wide variety of new communication, IO, and storage devices. Implementing the standard in Unix has required a variety of changes, since PCMCIA was originally developed for MS-DOS. The changes map concepts such as Socket Services onto components of the existing SVR4 device system design. Mostly, however, the PCMCIA Card Services layer has been maintained intact, providing high fidelity to the standard in the part of the API which matters most to device driver writers.

Serial line connectivity is and will remain the primary means for nomadic users to tap into the Internet. The PPP protocol, developed by the Internet Engineering Task Force, is an efficient serial line protocol allowing connection with existing network infrastructure. Since serial connections are generally more expensive and less reliable than Ethernet, connection management is required. The asynchronous serial line link manager hides the details of making serial

connections, so that a serial connection looks exactly like a standard TCP/IP connection to an application.

While many existing network applications can be used unchanged in the nomadic environment, others may require modification. Electronic mail is an example of the canonical nomadic application, since email is so useful to nomadic users. The IMAP interactive mail protocol, modified for nomadic use, provides the basis for efficient support of email over a serial line and for disconnected operation. The changes made to IMAP are a useful prototype for how application-specific protocols can be changed to support a nomadic model. In addition, modifications to the user interface allow users to specify caching upon disconnection, and provide a variety of other command features specific to the nomadic environment.

REFERENCES

- [1] Crispin, M. "Interactive Mail Access Protocol - Version 2," *Internet Engineering Task Force RFC 1176*, 1990.
- [2] Crispin, M., "IMAP2BIS - Extensions to the IMAP2 Protocol," *Internet Engineering Task Force RFC 1176*, 1992.
- [3] Lambert, M., "PCMail," *Internet Engineering Task Force RFC 1056*, 1988.
- [4] Lloyd, B., and Simpson, W., "PPP Authentication Protocols," *Internet Engineering Task Force RFC 1334*, 1992.
- [5] McGregor, G., "The PPP Internet Control Protocol (IPCP)," *Internet Engineering Task Force RFC 1332*, 1992.
- [6] PCMCIA Standard Release 2.01, Portable Computer Manufacturers' *Card Interface Association*, 1993.
- [7] Rose, M., "Post Office Protocol - Version 3," *Internet Engineering Task Force RFC 1081*, 1991.
- [8] Scheifler, R.W., and Gettys, J., "X Window System", *Digital Press*, 1992.
- [9] Simpson, W., "The Point-to-Point Protocol (PPP) for the Transmission of Multi-protocol Datagrams over Point-to-Point Links," *Internet Engineering Task Force RFC 1331*, 1992.

- [10] Simpson, W., "PPP Link Quality Monitoring," *Internet Engineering Task Force RFC 1333*, 1992.
- [11] SunOS 5.1 Writing Device Drivers, *SunSoft, Part No. 801-2871-10*, 1992.
- [12] "The Nomadic Computing Environment", *Tadpole Corp.*, 1993.

SCHEDULING FOR REDUCED CPU ENERGY

Mark Weiser,
Brent Welch*, Alan Demers, and Scott Shenker

Xerox Palo Alto Research Center,
3333 Coyote Hill Road
Palo Alto, CA 94304

*Sun Microsystems Laboratory
2600 Garcia Ave
Mountain View, CA 94043

ABSTRACT

The energy usage of computer systems is becoming more important, especially for battery operated systems. Displays, disks, and cpus, in that order, use the most energy. Reducing the energy used by displays and disks has been studied elsewhere; this paper considers a new method for reducing the energy used by the cpu. We introduce a new metric for cpu energy performance, millions-of-instructions-per-joule (MIPJ). We examine a class of methods to reduce MIPJ that are characterized by dynamic control of system clock speed by the operating system scheduler. Reducing clock speed alone does not reduce MIPJ, since to do the same work the system must run longer. However, a number of methods are available for reducing energy with reduced clock-speed, such as reducing the voltage [2][5] or using reversible [7] or adiabatic logic [1].

What are the right scheduling algorithms for taking advantage of reduced clock-speed, especially in the presence of applications demanding ever more instructions-per-second? We consider several methods for varying the clock speed dynamically under control of the operating system, and examine the performance of these methods against workstation traces. The primary result is that by adjusting the clock speed at a fine grain, substantial CPU energy can be saved with a limited impact on performance.

Permission has been granted by the USENIX Association to reprint this paper. This paper was originally published in the USENIX Association Conference Proceedings, 1994.
Copyright ©USENIX Association, 1994.

1 INTRODUCTION

The energy use of a typical laptop computer is dominated by the backlight and display, and secondarily by the disk. Laptops use a number of techniques to reduce the energy consumed by disk and display, primarily by turning them off after a period of no use [6][4]. We expect slow but steady progress in the energy consumption of these devices. Smaller computing devices often have no disk at all, and eliminate the display backlight that consumes much of the display-related power. Power consumed by the CPU is significant; the Apple Newton designers sought to maximize MIPS per WATT [3]. This paper considers some methods of reducing the energy used for executing instructions. Our results go beyond the simple power-down-when-idle techniques used in today's laptops.

We consider the opportunities for dynamically varying chip speed and so energy consumption. One would like to give users the appearance of a 100MIPS cpu at peak moments, while drawing much less than 100MIPS energy when users are active but would not notice a reduction in clock rate. Knowing when to use full power and when not requires the cooperation of the operating system scheduler. We consider a number of algorithms by which the operating system scheduler could attempt to optimize system power by monitoring idle time and reducing clock speed to reduce idle time to a minimum. We simulate their performance on some traces of process scheduling and compare these results to the theoretical optimum schedules.

2 AN ENERGY METRIC FOR CPUS

In this paper we use as our measure of the energy performance of a computer system the MIPJ, or millions of instructions per joule. MIPS/WATTS = MIPJ. (Of course MIPS have been superseded by better metrics, such as Specmark: we are using MIPS to stand for any such workload-per-time benchmark). MIPJ is not improving that much for high-end processors. For example, a 1984 2-MIPS 68020 consumed 2.0 watts (at 12.5Mhz), for a MIPJ of 1, and a 1994 200-MIPS Alpha chip consumes 40 watts, so has a MIPJ of 5. However, more recently lower speed processors used in laptops have been optimized to run at low power. For example, the Motorola 68349 is rated at 6 MIPS and consumes 300 mW for 20 MIPJ.

Other things being equal, MIPJ is unchanged by changes in clock speed. Reducing the clock speed causes a linear reduction in energy consumption, but a

similar reduction in MIPS. The two effects cancel. Similarly, turning the computer off, or reducing the clock to zero in the "idle-loop", does not effect MIPJ, since no instructions are being executed. However, a reduced clock speed creates the opportunity for quadratic energy savings; as the clock speed is reduced by n , energy per cycle can be reduced by n^2 . Three methods that achieve this are voltage reduction, reversible logic, and adiabatic switching. Our simulations assume n^2 savings, although it is really only important that the energy savings be greater than the amount by which the clock rate is reduced in order to achieve an increase in MIPJ.

Voltage reduction is currently the most promising way to save energy. Already chips are being manufactured to run at 3.3 or 2.2 volts instead of the 5.0 voltage levels commonly used. The intuition behind the power savings comes from the basic energy equation that is proportional to the square of the voltage.

$$E/\text{clock} \propto V^2 \quad (17.1)$$

The settling time for a gate is proportional to the voltage; the lower the voltage drop across the gate, the longer the gate takes to stabilize. To lower the voltage and still operate correctly, the cycle time must be lowered first. When raising the clock rate, the voltage must be increased first. Given that the voltage and the cycle time of a chip could be adjusted together, it should be clear now that the lower-voltage, slower-clock chip will dissipate less energy per cycle. If the voltage level can be reduced linearly as the clock rate is reduced, then the energy savings per instruction will be proportional to the square of the voltage reduction. Of course, for a real chip it may not be possible to reduce the voltage linear with the clock reduction. However, if it is possible to reduce the voltage at all by running slower, then there will be a net energy savings per cycle.

Currently manufacturers do not test and rate their chips across a smooth range of voltages. However, some data is available for chips at a set of voltage levels. For example, a Motorola CMOS 6805 microcontroller (cloned by SGS-Thomson) is rated at 6 Mhz at 5.0 Volts, 4.5 Mhz at 3.3 Volts, and 3 Mhz at 2.2 Volts. This is a close to linear relationship between voltage and clock rate.

The other important factor is the time it takes to change the voltage. The frequency for voltage regulators is on the order of 200 KHz, so we speculate that it will take a few tens of microseconds to boost the voltage on the chip.

Finally, why run slower? Suppose a task has a deadline in 100 milliseconds, but it will only take 50 milliseconds of CPU time when running at full speed to complete. A normal system would run at full speed for 50 milliseconds, and then idle for 50 milliseconds (assuming there were no other ready tasks).

During the idle time the CPU can be stopped altogether by putting it into a mode that wakes up upon an interrupt, such as from a periodic clock or from an I/O completion. Now, compare this to a system that runs the task at half speed so that it completes just before its deadline. If it can also reduce the voltage by half, then the task will consume 1/4 the energy of the normal system, even taking into account stopping the CPU during the idle time. This is because the same number of cycles are executed in both systems, but the modified system reduces energy use by reducing the operating voltage. Another way to view this is that idle time represents wasted energy, even if the CPU is stopped!

3 APPROACH OF THIS PAPER

This paper evaluates the fine grain control of CPU clock speed and its effect on energy use by means of trace-driven simulation. The trace data shows the context switching activity of the scheduler and the time spent in the idle loop. The goals of the simulation are to evaluate the energy savings possible by running slower (and at reduced voltage), and to measure the adverse affects of running too slow to meet the supplied demand. No simulation is perfect, however, and a true evaluation will require experiments with real hardware.

Trace data was taken from UNIX workstations over many hours of use by a variety of users. The trace data is described in Section 4 of the paper. The assumptions made by the simulations are described in Section 5. The speed adjustment algorithms are presented in Section 6. Section 7 evaluates the different algorithms on the basis of energy savings and a delay penalty function. Section 8 discusses future work, including some things we traced but did not fully utilize in our simulations. Finally, Section 9 provides our conclusions.

4 TRACE DATA

Trace data from the UNIX scheduler was taken from a number of workstations over periods of up to several hours during the working day. During these times the workloads included software development, documentation, e-mail, simulation, and other typical activities of engineering workstations. In addition, a few short traces were taken during specific workloads such as typing and scrolling through documents. Appendix I has a summary of the different traces we used.

Trace Points	Description
SCHED	Context switch away from a process
IDLE'ON	Enter the idle loop
IDLE'OFF	Leave idle loop to run a process
FORK	Create a new process
EXEC	Overlay a (new) process with another program
EXIT	Process termination
SLEEP	Wait on an event
WAKEUP	Notify a sleeping process

Table 1 These trace points were used to generate the trace data.

The trace points we took are summarized in Table 1. The idle loop events provide a view on how busy the machine is. The process information is used to classify different programs into foreground and background types. The sleep and wakeup events are used to deduce job ordering constraints.

In addition, the program counter of the call to sleep was recorded and kernel sources were examined to determine the nature of the sleep. The sleep events were classified into waits on "hard" and "soft" events. A hard event is something like a disk wait, in which a sleep is done in the kernel's biowait() routine. A soft event is something like a select that is done awaiting user input or a network request packet. The goal of this classification is to distinguish between idle time that can be eliminated by rescheduling (soft idle) and idle that is mandated by a wait on a device (hard idle).

Each trace record has a microsecond resolution time stamp. The trace buffer is periodically copied out of the kernel, compressed, and sent over the network to a central collection site. We used the trace data to measure the tracing overhead, and found it to range from 1.5% to 7% of the traced machine.

5 ASSUMPTIONS OF THE SIMULATIONS

The basic approach of the simulations was to lengthen the runtime of individually scheduled segments of the trace in order to eliminate idle time. The trace period was divided into intervals of various lengths, and the runtime and idletime during that interval were used to make a speed adjustment decision.

If there were excess cycles left over at the end of an interval because the speed was too slow, they were carried over into the next interval. This carry-over is used as a measure of the penalty from using the speed adjustment.

The ability to stretch runtime into idle periods was refined by classifying sleep events into "hard" and "soft" events. The point of the classification is to be fair about what idle time can be squeezed out of the simulated schedule by slowing down the processor. Obviously, running slower should not allow a disk request to be postponed until just before the request completes in the trace. However, it is reasonable to slow down the response to a keystroke in an editor such that the processing of one keystroke finishes just before the next.

Our simulations did not reorder trace data events. We justify this by noting that only if the offered load is far beyond the capacity of the CPU will speed changes affect job ordering significantly. Furthermore, the CPU speed is ramped up to full speed as the offered load increases, so in times of high demand the CPU is running at the speed that matches the trace data.

In addition, we made the following assumptions:

- The machine was considered to use no energy when idle, and to use energy/instruction in proportion to n^2 when running at a speed n , where n varies between 1.0 and a minimum relative speed. This is a bit optimistic because a chip will draw a small amount of power while in standby mode, and we might not get a one-to-one reduction in voltage to clock speed. However, the baseline power usages from running at full speed (reported as 1.0 in the graphs) also assume that the CPU is off during idle times.
- It takes no time to switch speeds. This is also optimistic. In practice, raising the speed will require a delay to wait for the voltage to rise first, although we speculate that the delay is on the order of 10s of instructions (not 1000s).
- After any 30 second period of greater than 90% idle we assumed that any laptop would have been turned off, and skipped simulating until the next 30 second period with less than 90% idle. This models the typical power saving features already present in portables. The energy savings reported below does not count these off periods.
- There was assumed to be a lower bound to practical speed, either 0.2, 0.44 or 0.66, where 1.0 represents full speed. In 5V logic using voltage reduction for power savings, these correspond to 1.0 V, 2.2 V and 3.3V

minimum voltage levels, respectively. The 1.0 V level is optimistic, while the 2.2 V and 3.3V levels are based on several existing low power chips. In the graphs presented in section 7, the minimum voltage of the system is indicated, meaning that the voltage can vary between 5.0 V and the minimum, and the speed will be adjusted linearly with voltage.

6 SCHEDULING ALGORITHMS

We simulated three types of scheduling algorithms: unbounded-delay perfect-future (OPT), bounded-delay limited-future (FUTURE), and bounded-delay limited-past (PAST). Each of these algorithms adjust the CPU clock speed at the same time that scheduling decisions are made, with the goal of decreasing time wasted in the idle loop while retaining interactive response.

OPT takes the entire trace, and stretches all the run times to fill all the idle times. Periods when the machine was "off" (more than 90% idle over 30 seconds) were not considered available for stretching runtimes into. This is a kind of batch approach to the work seen in the trace period: as long as all that work is done in that period, any piece can take arbitrarily long. OPT power savings were almost always limited by the minimum speed, achieving the maximum possible savings over the period. This algorithm is both impractical and undesirable. It is impractical because it requires perfect future knowledge of the work to be done over the interval. It also assumes that all idle time can be filled by stretching runlengths and reordering jobs. It is undesirable because it produces large delays in runtimes of individual jobs without regard to the need for effective response to real-time events like user keystrokes or network packets.

FUTURE is like OPT, except it peers into the future only a small window, and optimizes energy over that window, while never delaying work past the window. Again, it is assumed that all idle time in the next interval can be eliminated, unless the minimum speed of the CPU is reached. We simulated windows as small as 1 millisecond, where savings are usually small, and as large as 400 seconds, where FUTURE generally approaches OPT in energy savings. FUTURE is impractical, because it uses future knowledge, but desirable, because no realtime response is ever delayed longer than the window.

By setting a window of 10 to 50 milliseconds, user interactive response will remain high. In addition, a window this size will not substantially reduce a

very long idle time, one that would trigger the spin down of a disk or the blanking of a display. Those decisions are based on idle times of many seconds or a few minutes, so stretching a computation out by a few tens of milliseconds will not affect them.

PAST is a practical version of FUTURE. Instead of looking a fixed window into the future it looks a fixed window into the past, and assumes the next window will be like the previous one. The PAST speed setting algorithm is shown in Figure 1.

```
IdleCycles = HardIdle + SoftIdle;  
RunCycles += ExcessCycles;  
RunPercent = RunCycles / (IdleCycles + RunCycles);  
NextExcess = RunCycles -  
    Speed * (RunCycles + SoftIdle)  
IF ExcessCycles < 0. THEN  
    ExcessCycles = 0.  
Energy = (RunCycles - ExcessCycles) * Speed * Speed;  
IF ExcessCycles > IdleCycles THEN  
    NewSpeed = 1.0;  
ELSEIF RunPercent > 0.7 THEN  
    NewSpeed = Speed + 0.2;  
ELSEIF RunPercent < 0.5 THEN  
    NewSpeed = Speed - (0.6 - RunPercent);  
IF NewSpeed > 1.0 THEN  
    NewSpeed = 1.0;  
IF NewSpeed < MinSpeed THEN  
    NewSpeed = MinSpeed;  
Speed = NewSpeed;  
ExcessCycles = NextExcess;
```

Figure 1 Speed Setting Algorithm (PAST). *RunCycles* is the number of non-idle CPU cycles in the last interval. *IdleCycles* is the idle CPU cycles, split between hard and soft idle time. *ExcessCycles* is the cycles left over from the previous interval because we ran too slow. All these cycles are measured in time units.

There are four parts to the code. The first part computes the percent of time during the interval when the CPU was running. The *RunCycles* come from two

sources, the runtime in the trace data for the interval, and the *ExcessCycles* from the simulation of the previous interval.

The *ExcessCycles* represents a carry over from the previous interval because the CPU speed was set too slow to accommodate all the load that was supplied during the interval. Consider:

$$\text{NextExcess} = \text{RunCycles} - \text{speed} * (\text{RunCycles} + \text{SoftIdle}) \quad (17.2)$$

The *RunCycles* is the sum of the cycles presented by the trace data and the previous value of *ExcessCycles*. This initial value is reduced by the soft idle time and the number of cycles actually performed at the current speed. This calculation represents the ability to squeeze out idle time by lengthening the runtimes in the interval. Only "soft" idle, such as waiting for keyboard events, is available for elimination of idle. As the soft idle time during an interval approaches zero, the excess cycles approach:

$$\text{RunCycles} * (1 - \text{OldSpeed}) \quad (17.3)$$

The energy used during the interval is computed based on an n^2 relationship between speed and power consumption per cycle. The cycles that could not be serviced during the interval have to be subtracted out first. They will be accounted for in the next interval, probably at a higher CPU speed.

The last section represents the speed setting policy. The adjustment of the clock rate is a simple heuristic that attempts to smooth the transitions from fast to slow processing. If the system was more busy than idle, then the speed is ramped up. If it was mostly idle, then it is slowed down. We simulated several variations on the code shown here to come up with the constants shown here.

7 EVALUATING THE ALGORITHMS

Figure 2 on page 459 compares the results of these three algorithms on a single trace (Kestrel March 1) as the adjustment interval is varied. The OPT energy is unaffected by the interval, but is shown for comparison. The vertical access shows relative power used by the scheduling algorithms, with 1.0 being full power. Three sets of three lines are shown, corresponding to three voltage levels which determine the minimum speed, and the three algorithms, OPT, FUTURE, and PAST. The PAST and FUTURE algorithms approach OPT as the interval is lengthened. (Note that the log scale for the X axis.) For the

same interval PAST actually does better than FUTURE because it is allowed to defer excess cycles into the next interval, effectively lengthening the interval. The intervals from 10 msec to 50 msec are considered in more detail in other figures.

Figure 3 on page 460 shows the excess cycles that result from setting the speed too slow in PAST when using a 20 msec adjustment interval and the same trace data as Figure 2. Note that the graph uses log-log scales. Cycles are measured in the time it would take to execute them at full speed. The data was taken as a histogram, so a given point counts all the excess cycles that were less than or equal that point on the X axis, but greater than the previous bucket value in the histogram. Lines are used to connect the points so that the spike at zero is evident. The large spike at zero indicates that most intervals have no excess cycles at all. There is a smaller peak near the interval length, and then the values drop off.

As the minimum speed is lowered, there are more cases where excess cycles build up, and they can accumulate in longer intervals. This is evident Figure 3 where the points for 1.0 V are above the others, which indicates more frequent intervals with excess cycles, and the peak extends to the right, which indicates longer excess cycle intervals.

Figure 4 on 461 shows the relationship between the interval length and the peak in excess cycle length. It compares the excess cycles with the same minimum voltage (2.2 V) while the interval length varies. This is from the same trace data as Figures 2 and 3. The main result here is that the peak in excess cycle lengths shifts right as the interval length increases. All this means is that as a longer scheduling interval is chosen, there can be more excess cycles built up.

Figure 5 on page 463 compares the energy savings for the bounded delay limited past (PAST) algorithm with a 20 msec adjustment interval and with three different minimum voltage limits. In this plot each position on the X access represents a different set of trace data. The position corresponding to the trace data used in Figures 2 to 4 is indicated with the arrow.

While there is a lot of information in the graph, there are two overall points to get from the figure: the relative savings for picking different minimum voltages, and the overall possible savings across all traces.

The first thing to look for in Figure 5 is that for any given trace the three points show the relative possible energy savings for picking the three different minimum voltages. Interestingly, the 1.0 V minimum does not always result in

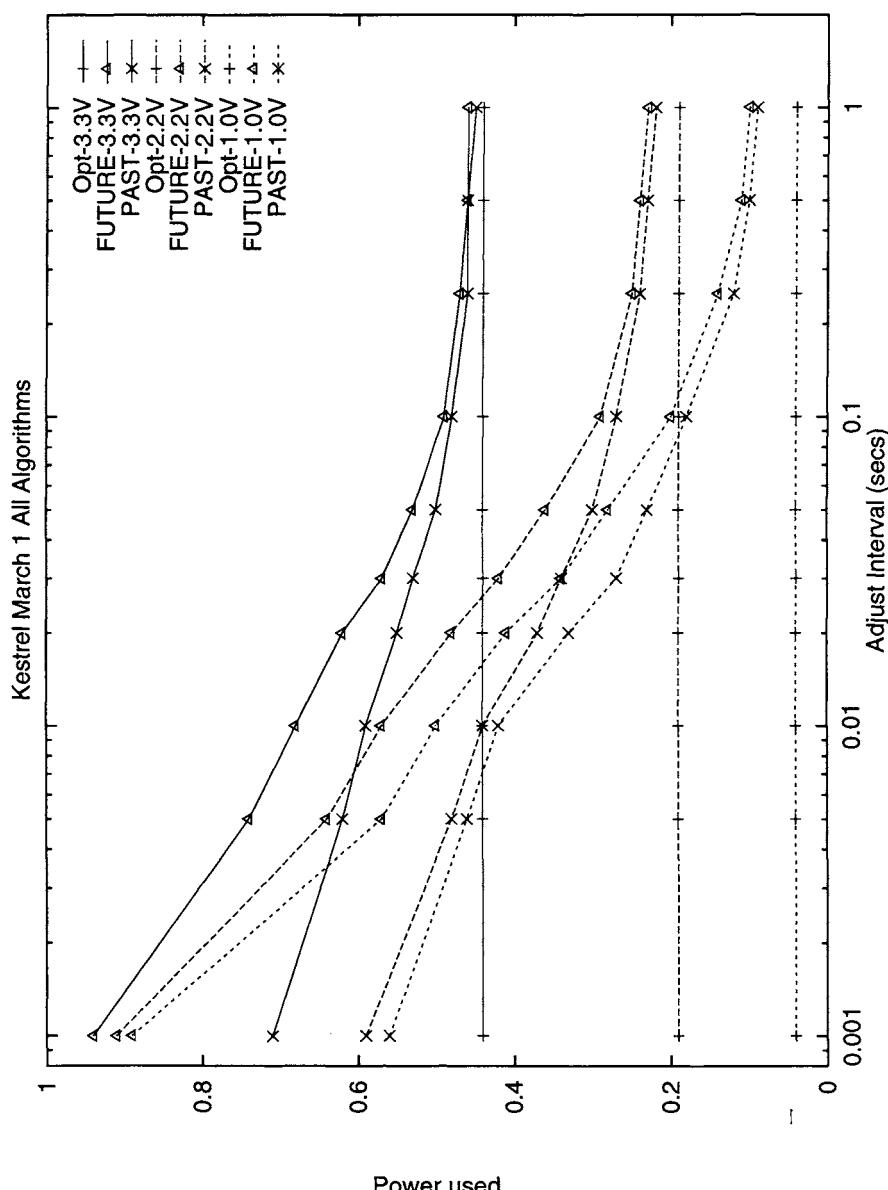


Figure 2 All algorithms compared with three different minimum voltages. As the adjustment interval lengthens, more power is saved in the FUTURE and PAST algorithms. The savings is always the same with OPT, which is shown for comparison.

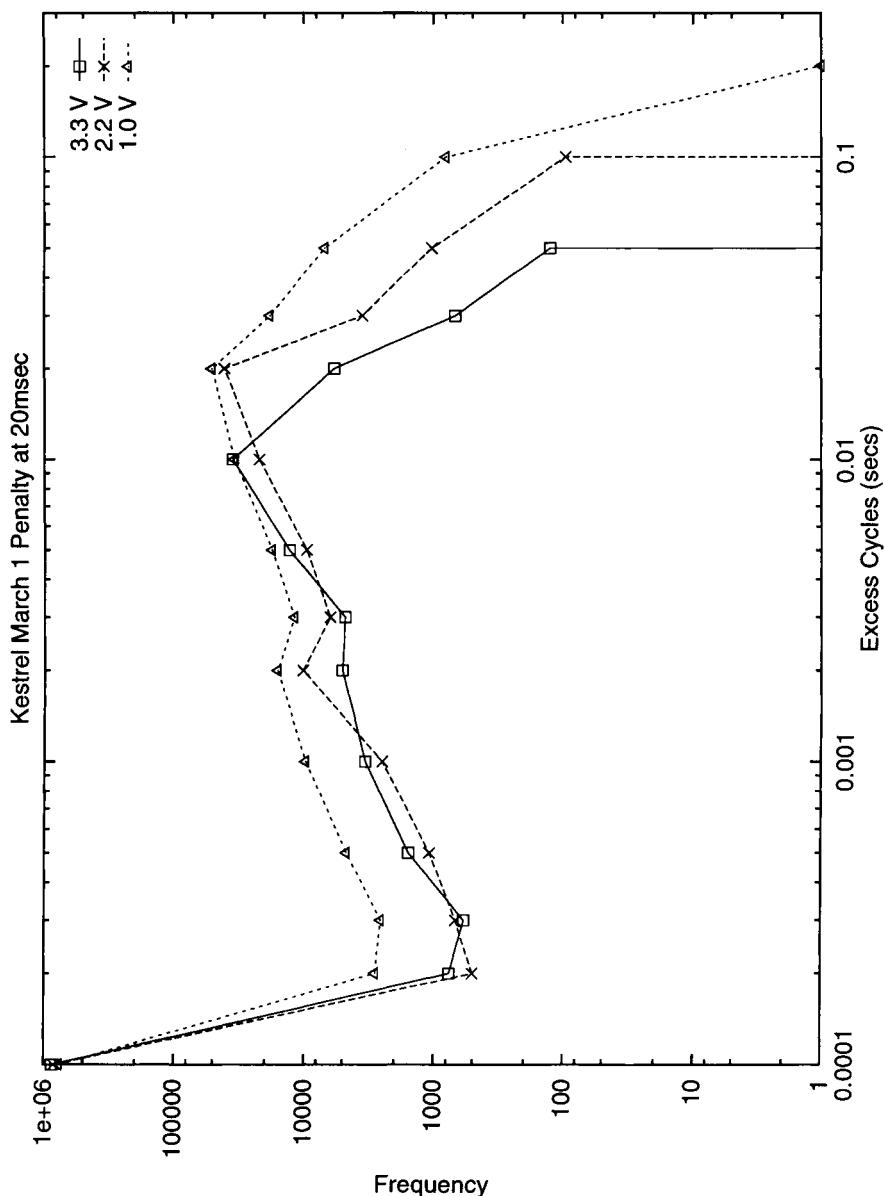


Figure 3 Excess cycle penalty at 20 msec for three different minimum voltages. The frequency is the number of occurrences of the penalty during the simulation.

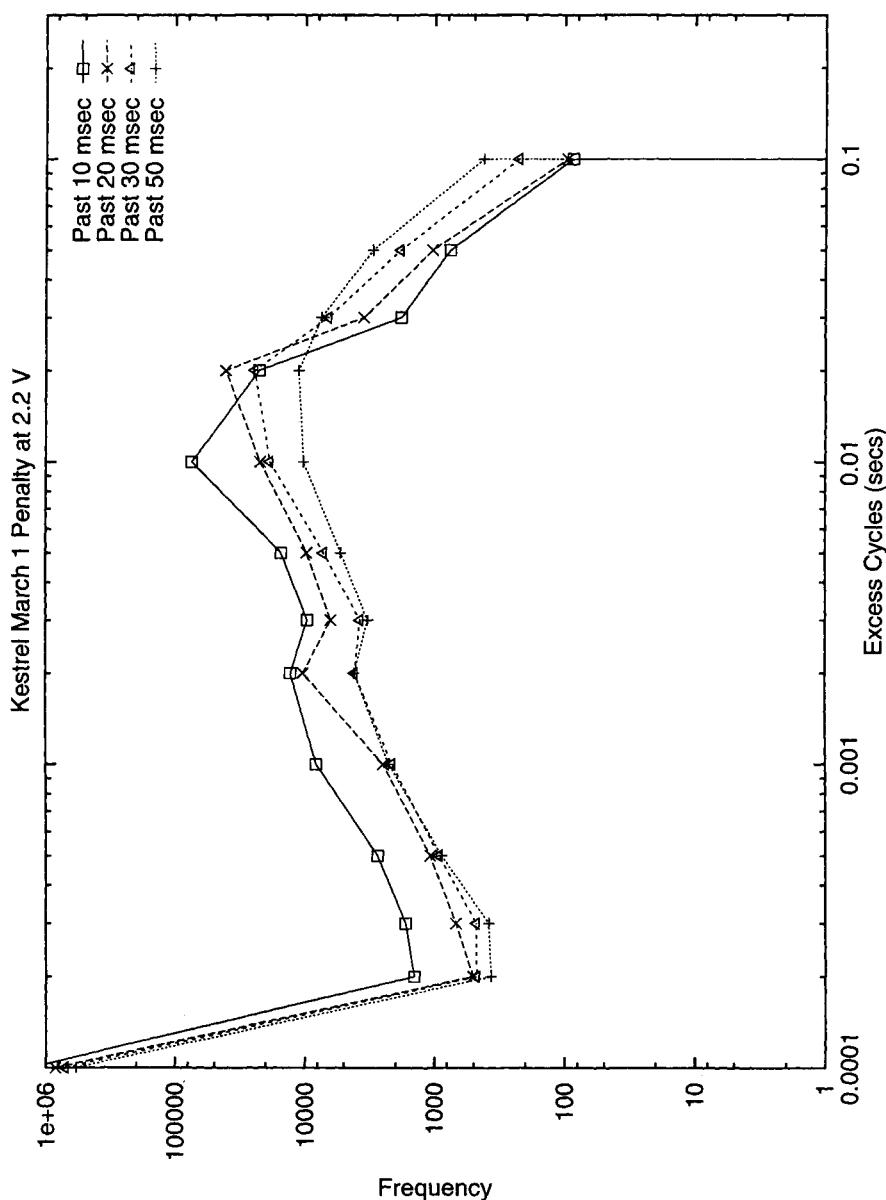


Figure 4 Excess cycle penalty at 2.2 minimum volts for different scheduling intervals. The frequency is the number of occurrences of the penalty during the simulation.

the minimum energy. This is because it has more of a tendency to fall behind (more excess cycles), so its speed varies more and the power consumption is less efficient. Even when 1.0 V does provide the minimum energy, the 2.2 V minimum is almost as good.

The other main point conveyed by Figure 5 is that in most of the traces the potential for energy savings is good. The savings range from about 5% to about 75%, with most data points falling between 25% to 65% savings.

Figure 6 on page 464 fixes the minimum voltage at 2.2 V and shows the effect of changing the interval length. The OPT energy savings for 2.2 V is plotted for comparison. Again, each position on the X axis represents a different trace. The position corresponding to the trace data used in Figures 2 to 4 is indicated with the arrow.

In this figure the main message to get is the difference in relative savings for a given trace as the interval is varied. This is represented by the spread in the points plotted for each trace. A longer adjustment period results in more savings, which is consistent with Figure 2.

Figures 7 and 8 on pages 465 and 466 show the average excess cycles for all trace runs. These averages do not count intervals with zero excess cycles. Figure 7 shows the excess cycles at a given adjustment interval (20 msec) and different minimum voltages. Figure 8 shows the excess cycles at a given minimum voltage (2.2 V) and different intervals. Again, the lower minimum voltage results show more excess cycles, and the longer intervals accumulate more excess cycles.

There is a trade off between the excess cycles penalty and the energy savings that is a function of the interval size. As the interval decreases, the CPU speed is adjusted at a finer grain and so it matches the offered load better. This results in fewer excess cycles, but it also does not save as much energy. This is consistent with the motivating observation that it is better to execute at an average speed than to alternate between full speed and full idle.

8 DISCUSSION AND FUTURE WORK

The primary source of feedback we used for the speed adjustment was the percent idle time of the system. Another approach is to classify jobs into background, periodic, and foreground classes. This is similar to what Wilkes

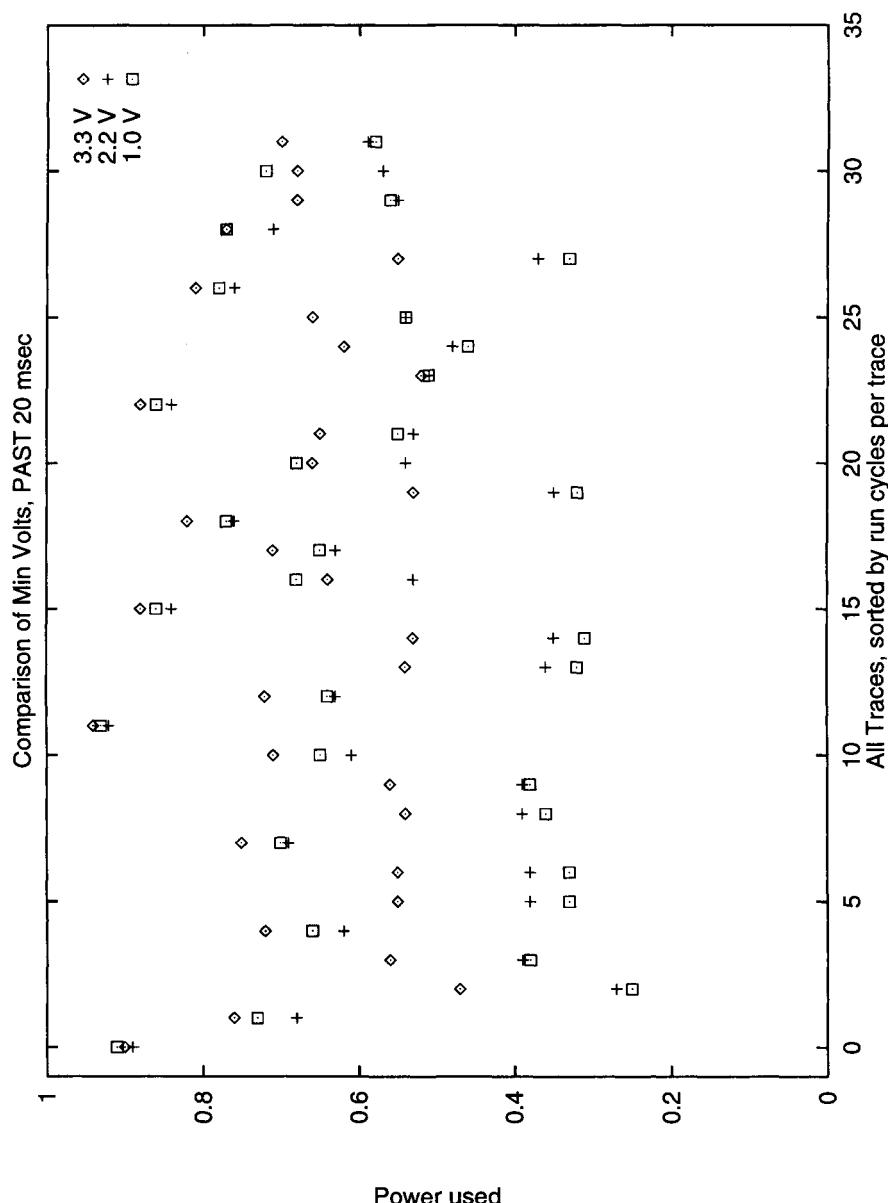


Figure 5 A comparison of energy savings in all traces with a 20 msec scheduling interface and three different minimum voltages.

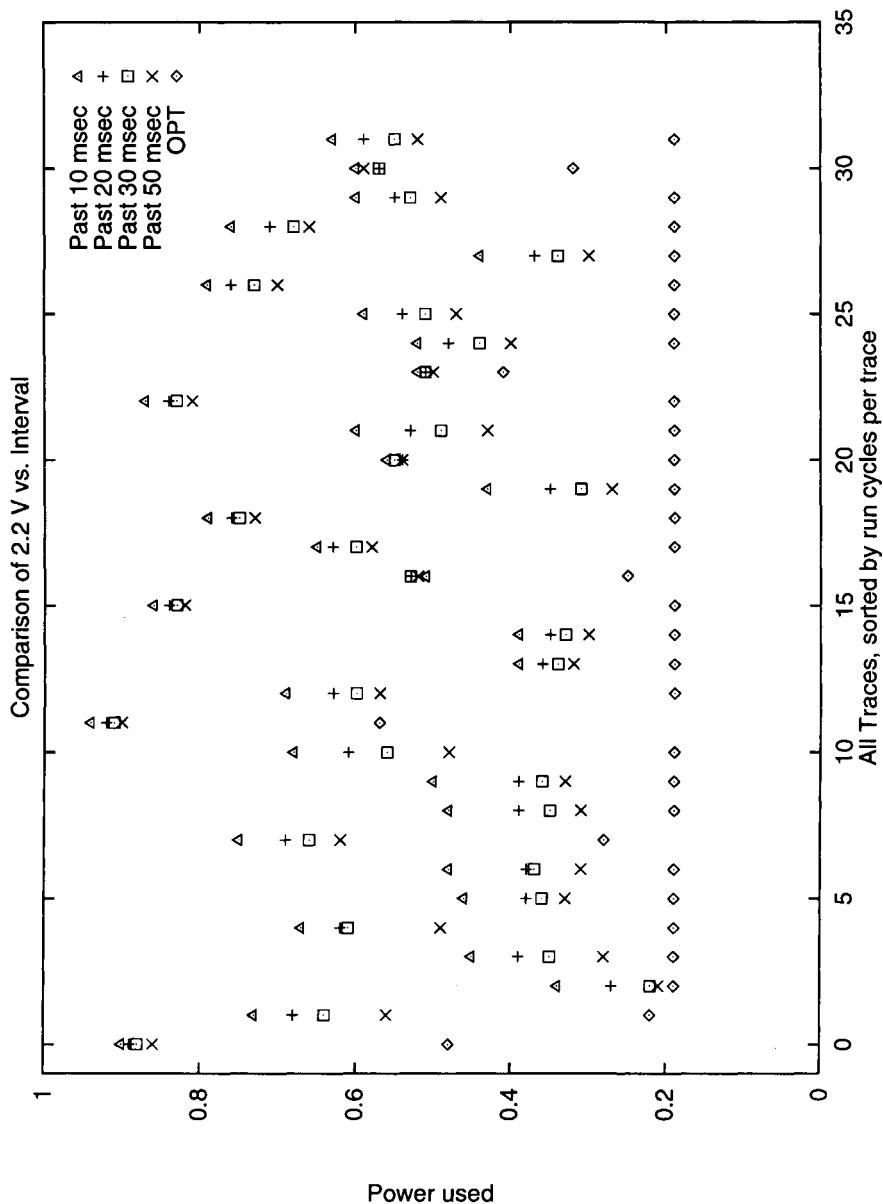


Figure 6 A comparison of energy savings in all traces with a 2.2 V minimum voltage and different scheduling intervals.

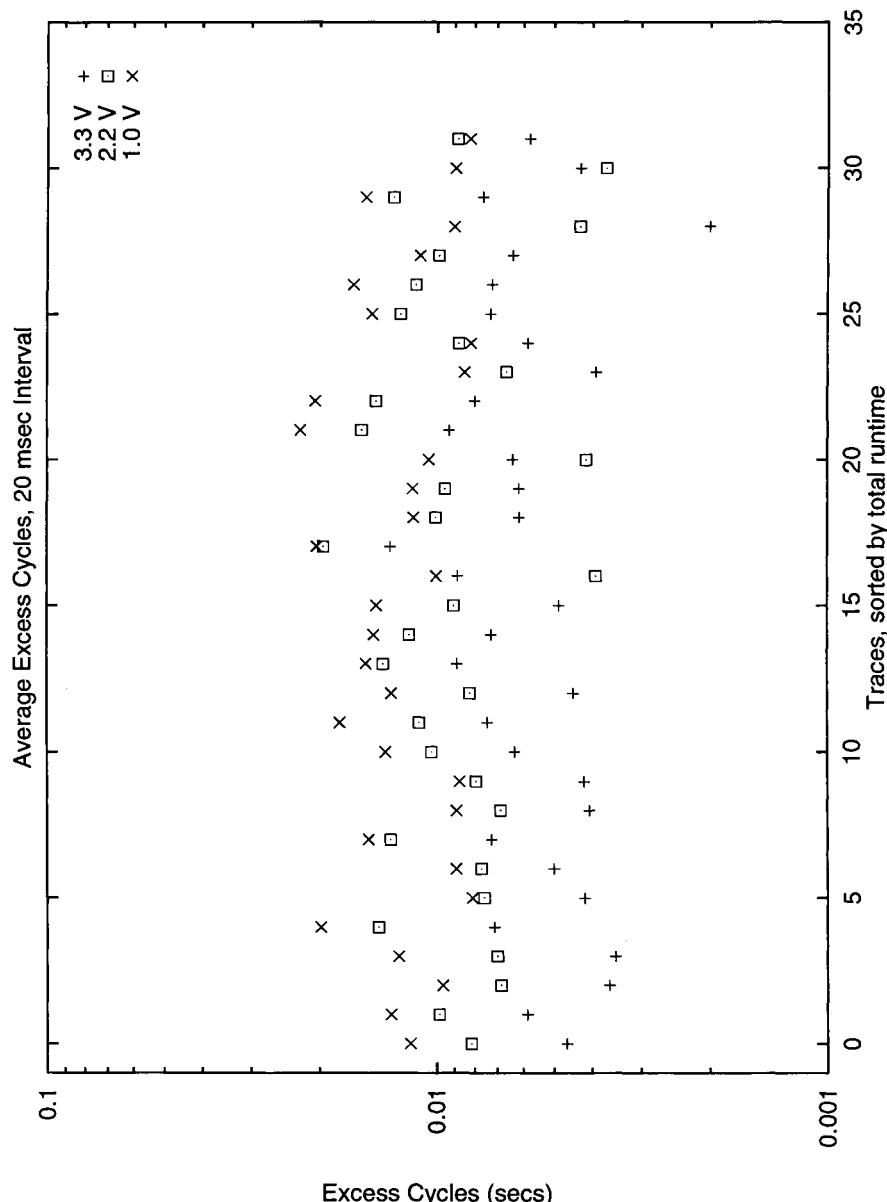


Figure 7 A comparison of excess cycle penalty in all traces with a 20 msec scheduling interval and different minimum voltages.

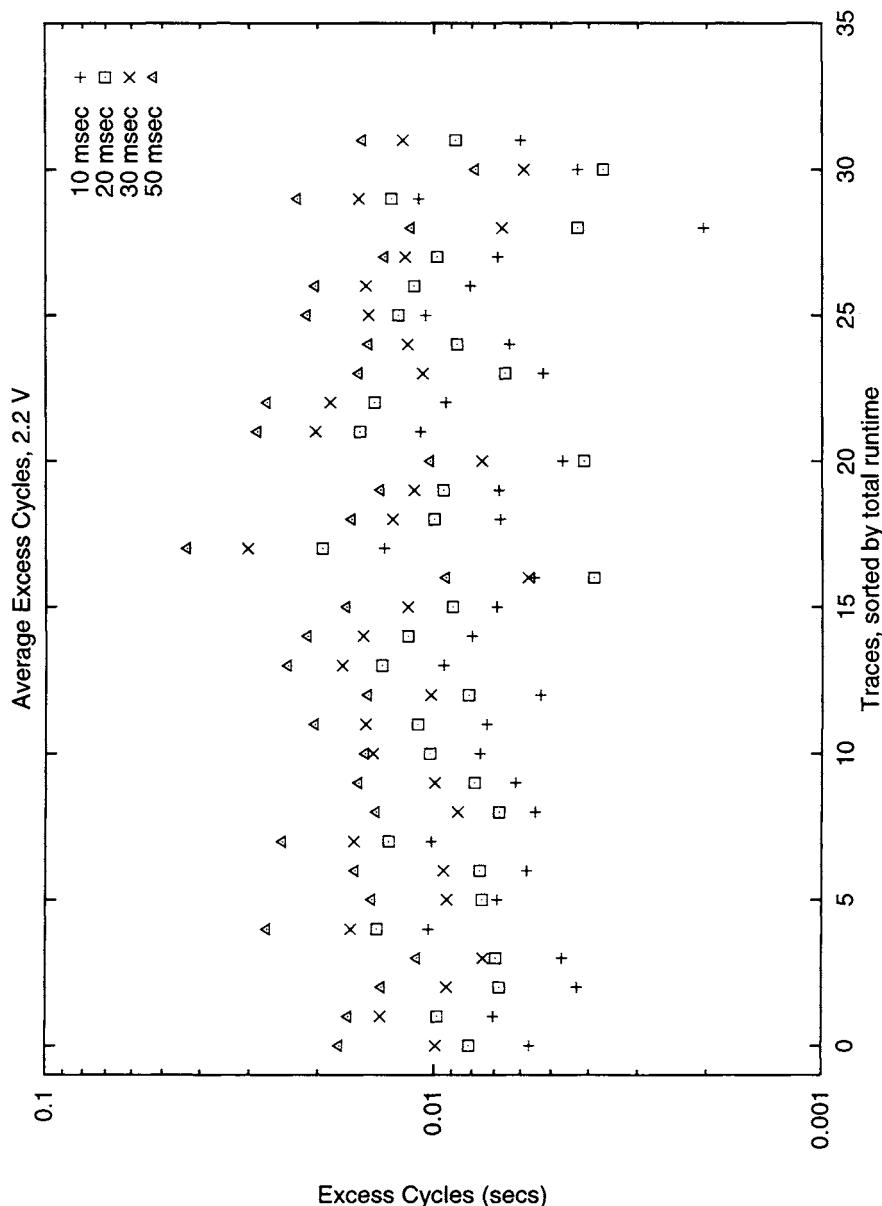


Figure 8 A comparison of excess cycle penalty in all traces with a 2.2 V minimum voltage and different scheduling intervals.

proposes in his schemes to utilize idle time [8]. With this sort of classification the speed need not be ramped up when executing background tasks. Periodic tasks impose a constant, measurable load. They typically run for a short burst and then sleep for a relatively long time. With these tasks there is a well defined notion of "fast enough", and the CPU speed can be adjusted to finish these tasks just in time. When there is a combination of background, periodic, and foreground tasks, then the standard approach is to schedule the periodic tasks first, then fit in the foreground tasks, and lastly fit in the background tasks. In this case there would be a minimum speed that would always execute the periodic tasks on time, and the system would increase the speed in response to the presence of foreground and background tasks.

The simulations we performed are simplified by not reordering scheduling events. In a real rate-adjusting scheduler, the change in processing rates will have an effect on when jobs are preempted due to time slicing and the order that ready jobs are scheduled. We argue that unless there is a large job mix, then the reordering will not be that significant. Our speed adjustment algorithm will ramp up to full speed during heavy loads, and during light loads the reordering should not have a significant effect on energy.

In order to evaluate more realistic scheduling algorithms, it would be interesting to generate an abstract load for the simulation. This load includes CPU runs with preemption points eliminated, pause times due to I/O delays preserved, and causal ordering among jobs preserved. Given an abstract load, it would be possible to simulate a scheduler in more detail, giving us the ability to reorder preemption events while still preserving the semantics of I/O delays and IPC dependencies.

We have attempted to model the I/O waits by classifying idle time into "hard" and "soft" idle. We think this approximation is valid, but it would be good to verify it with a much more detailed simulation.

9 CONCLUSIONS

This paper presents preliminary results on CPU scheduling to reduce CPU energy usage, beyond the simple approaches taken by today's laptops. The metric of interest is how many instructions are executed for a given amount of energy, or MIPJ. The observation that motivates the work is that reducing the cycle time of the CPU allows for power savings, primarily by allowing the

CPU to use a lower voltage. We examine the potential for saving power by scheduling jobs at different clock rates.

Trace driven simulation is used to compare three classes of schedules: OPT that spreads computation over the whole trace period to eliminate all idle time (regardless of deadlines), FUTURE that uses a limited future look ahead to determine the minimum clock rate, and PAST that uses the recent past as a predictor of the future. A PAST scheduler with a 50 msec window shows power savings of up to 50% for conservative circuit design assumptions (e.g., 3.3 V), and up to 70% for more aggressive assumptions (2.2 V). These savings are in addition to the obvious savings that come from stopping the processor in the idle loop, and powering off the machine all together after extended idle periods.

The energy savings depends on the interval between speed adjustments. If it is adjusted at too fine a grain, then less power is saved because CPU usage is bursty. If it is adjusted at too coarse a grain, then the excess cycles built up during a slow interval will adversely affect interactive response. An adjustment interval of 20 or 30 milliseconds seems to represent a good compromise between power savings and interactive response.

Interestingly, having too low a minimum speed results in less efficient schedules because there is more of a tendency to have excess cycles and therefore the need to speed up to catch up. In particular, a minimum voltage of 2.2 V seems to provide most of the savings of a minimum voltage of 1.0 V. The 1.0 V system, however, tends to have a larger delay penalty as measured by excess cycles.

In general, scheduling algorithms have the potential to provide significant power savings while respecting deadlines that arise from human factors considerations. If an effective way of predicting workload can be found, then significant power can be saved by adjusting the processor speed at a fine grain so it is just fast enough to accommodate the workload. Put simply, the tortoise is more efficient than the hare: it is better to spread work out by reducing cycle time (and voltage) than to run the CPU at full speed for short bursts and then idle. This stems from the non-linear relationship between CPU speed and power consumption.

Acknowledgements

This work was supported in part by Xerox, and by ARPA under contract DABT63-91-C-0027; funding does not imply endorsement. David Wood of the University of Wisconsin helped us get started in this research, and provided substantial assistance in understanding CPU architecture. The authors benefited from the stimulating and open environment of the Computer Science Lab at Xerox PARC.

REFERENCES

- [1] William C. Athas, Jeffrey G. Koller, and Lars "J." Svensson. "An Energy-Efficient CMOS Line Driver Using Adiabatic Switching", 1994 IEEE Fourth Great Lakes Symposium on VLSI, pp. 196-199, March 1994.
- [2] A. P. Chandrakasan and S. Sheng and R. W. Brodersen. "Low-Power CMOS Digital Design". JSSC, V27, N4, April 1992, pp 473-484.
- [3] Michael Culbert, "Low Power Hardware for a High Performance PDA", to appear Proc. of the 1994 Computer Conference, San Francisco.
- [4] Fred Dougulis, P. Krishnan, Brian Marsh, "Thwarting the Power-Hungry Disk", Proc. of Winter 1994 USENIX Conference, January 1994, pp 293-306
- [5] Mark A. Horowitz. "Self-Clocked Structures for Low Power Systems". ARPA semi-annual report, December 1993. Computer Systems Laboratory, Stanford University.
- [6] Kester Li, Roger Kumpf, Paul Horton, Thomas Anderson, "A Quantitative Analysis of Disk Drive Power Management in Portable Computers", Proc. of Winter 1994 USENIX Conference, January 1994, pp 279-292.
- [7] S. Younis and T. Knight. "Practical Implementation of Charge Recovering Asymptotically Zero Power CMOS." 1993 Symposium on Integrated Systems (C. Ebeling and G. Borriello, eds.), Univ. of Washington, 1993.
- [8] Wilkes, John "Idleness is not Sloth", to appear, proc. of the 1995 Winter USENIX Conf

APPENDIX A

A.1 DESCRIPTION OF TRACE DATA

Table A.1 on page 471 lists the characteristics of the 32 traces runs that are reported in the figures. The table is sorted from shortest to longest runtime to match the ordering in Figures 5 through 8. The elapsed time of each trace is broken down into time spent running the CPU on behalf of a process (Runtime), time spent in the idle loop (IdleTime), and time when the machine is considered so idle that it would be turned off by a typical laptop power manager (Offtime). The short traces labeled mx, emacs, and fm are of typing (runs 1 and 2) and scrolling (run 3) in various editors. The remaining runs are taken over several hours of everyday use.

I	Trace	Runtime	Idle	Elapsed	Offtime
0	feb28klono	0.906	29.094	9H 24M 20S	33828.9
1	idle1	1.509	28.653	39S	9.05
2	heur1	7.043	3.103	10S	0
3	emacs2	7.585	31.719	40S	0
4	emacs1	8.060	32.273	40S	0
5	mx2	8.362	30.916	39S	0
6	mx1	9.508	30.871	41S	0
7	fm1	9.544	10.594	20S	0
8	em3	11.669	27.580	40S	0
9	fm2	16.679	23.770	41S	0
10	mx3	20.738	18.642	39S	0
11	feb28dekanore	30.548	541.045	9H 24M 40S	33307.8
12	fm3	30.626	9.942	41S	0
13	mar1klono	41.822	1011.251	9H 55M 46S	34690.6
14	feb28mezzo	61.940	449.717	9H 24M 20S	33346.1
15	mar1cleonie	214.656	1321.591	9H 50S	30913.0
16	feb28kestrel	510.259	3362.222	1H 4M 33S	0
17	feb28corvina	524.248	768.857	9H 24M 41S	32588.0
18	mar1mezzo	686.340	673.204	9H 55M 36S	34375.7
19	marlegeus	695.409	4774.911	9H 55M 35S	30263.6
20	feb28ptarmigan	1497.908	2207.005	1H 1M 41S	0
21	feb28fandango	1703.037	3489.760	9H 24M 17S	28665.0
22	feb28zwilnik	4414.429	29448.058	9H 24M 21S	0
23	mar1zwilnik	4914.787	30823.917	9H 55M 38S	0
24	mar1kestrel	5135.297	30599.364	9H 55M 34S	0
25	feb28siria	6714.109	27146.678	9H 24M 20S	0
26	mar1siria	8873.114	26868.738	9H 55M 37S	0
27	feb28egeus	9065.477	13500.028	6H 16M 6S	0
28	mar1corvina	10898.545	24648.883	9H 55M 57S	210.202
29	mar1ptarmigan	12416.924	23319.178	9H 55M 34S	0
30	mar1fandango	20101.182	15638.594	9H 55M 38S	0
31	mar1dekanore	25614.651	14168.562	9H 55M 58S	7191.81

Table A.1 Summary of the Trace Data.

Storage Alternatives for Mobile Computers

Fred Douglis, Ramón Cáceres*,
M. Frans Kaashoek**, P. Krishnan*,
Kai Li***, Brian Marsh****,
and Joshua Tauber**

*AT&T Bell Laboratories
600 Mountain Ave., Room 2B-105
Murray Hill, NJ 07974*

* *AT&T Bell Laboratories, Holmdel, NJ*

** *Laboratory of Computer Science,*

Massachusetts Institute of Technology, Cambridge, MA

*** *Computer Science Department, Princeton University, Princeton, NJ*

**** *D.E. Shaw & Co., New York, NY*

ABSTRACT

Mobile computers such as notebooks, subnotebooks, and palmtops require low weight, low power consumption, and good interactive performance. These requirements impose many challenges on architectures and operating systems. This chapter investigates three alternative storage devices for mobile computers: magnetic hard disks, flash memory disk emulators, and flash memory cards.

We have used hardware measurements and trace-driven simulation to evaluate each of the alternative storage devices and their related design strategies. Hardware measurements on an HP OmniBook 300 highlight differences in the performance of the three devices as used on the Omnibook, especially the poor performance of version 2.00 of the Microsoft Flash File System [12] when accessing large files. The traces used

Permission has been granted by the USENIX Association to reprint the above paper. An earlier version of this appeared in the *Proceedings of the First Symposium on Operating Systems Design and Implementation*, USENIX Association, November, 1994, Copyright ©USENIX Association, 1994. Unlike that version, this chapter considers an SRAM buffer cache in addition to flash memory. It also uses updated parameters for one of the devices, and measures response time differently. A detailed description of the changes appears at the end of the chapter on page 503. This work was performed in part at the Matsushita Information Technology Laboratory of Panasonic Technologies, Inc.

in our study came from different environments, including mobile computers (Macintosh PowerBooks) and desktop computers (running Windows or HP-UX), as well as synthetic workloads. Our simulation study shows that flash memory can reduce energy consumption and mean read response time by up to two orders of magnitude, compared to magnetic disk, while providing acceptable write performance. These energy savings can translate into a 25–115% extension of battery life. We also find that the amount of unused memory in a flash memory card has a substantial impact on energy consumption, performance, and endurance: compared to low storage utilizations (40% full), running flash memory near its capacity (95% full) can increase energy consumption by 50–165%, degrade write response time by 24%, and decrease the lifetime of the memory card by up to 70%. For flash disks, asynchronous erasure can improve write response time by a factor of 2.5.

1 INTRODUCTION

Mobile computer environments are different from traditional workstations because they require light-weight, low-cost, and low-power components, while still needing to provide good interactive performance. A principal design challenge is to make the storage system meet these conflicting requirements.

Current storage technologies offer two alternatives for file storage on mobile computers: magnetic hard disks and flash memory. Hard disks provide large capacity at low cost, and have high throughput for large transfers. The main disadvantage of disks is that they consume a lot of energy and take seconds to spin up and down. Flash memory consumes relatively little energy, and has low latency and high throughput for read accesses. The main disadvantages of flash memory are that it costs more than disks—\$30–50/Mbyte, compared to \$1–5/Mbyte for magnetic disks—and that it requires erasing before it can be overwritten. It comes in two forms: flash memory cards (accessed as main memory) and flash disk emulators (accessed through a disk block interface).¹ These devices behave differently, having varying access times and bandwidths.

This chapter investigates three storage systems: magnetic disk, flash disk emulator, and directly accessed flash memory. All of these systems include a DRAM file cache and SRAM write buffer. Our study is based on both hardware measurements and trace-driven simulation. The measurements are “micro-

¹In this chapter, we use *flash disk* to refer to block-accessed flash disk emulators. We use *flash (memory) card* to refer to byte-accessible flash devices. When we wish to refer to the generic memory device or either of the above devices built with it, we refer to *flash memory* or a *flash device*. Note that the flash disk is actually a flash memory card as well, but with a different interface.

benchmarks” that compare the raw performance of three different devices: a typical mobile disk drive (Western Digital Caviar Ultralite CU140), a flash disk (SunDisk 10-Mbyte SDP10 PCMCIA flash disk [22], sold as the Hewlett-Packard F1013A 10-Mbyte/12-V Flash Disk Card [7]), and a flash memory card (Intel 10-Mbyte Series-2 flash memory card [9]). The measurements provide a baseline comparison of the different architectures and are used as device specifications within the simulator. They also point out specific performance issues, particularly with the Microsoft Flash File System (MFFS) version 2.00 [12].

Flash memory is significantly more expensive than magnetic disks, but our simulation results show that flash memory can offer energy reduction by an order of magnitude over disks—even with aggressive disk spin-down policies that save energy at the cost of performance [6, 14]. Since the storage subsystem can consume 20–54% of total system energy [14, 16], these energy savings can as much as double battery lifetime. Flash provides better read performance than disk, but worse average write performance. The maximum delay for magnetic disk reads or writes, however, is much higher than maximum flash latency due to the overhead of occasional disk spin-ups.

We also show that the key to file system support using flash memory is erasure management. With a flash card, keeping a significant portion of flash memory free is essential to energy conservation and performance. With a flash disk, decoupling write and erase latency can improve average write response by a factor of 2.5.

In total, our research uses both hardware measurements and simulation to contribute two key results: a quantitative comparison of the alternatives for storage on mobile computers, taking both energy and performance into account, and an analysis of techniques that improve on existing systems.

The rest of this chapter is organized as follows. The next section discusses the three storage architectures in greater detail. Section 3 describes the hardware micro-benchmarks. Section 4 describes our traces and the simulator used to perform additional studies. After that come the results of the simulations. Section 6 discusses related work, and Section 7 concludes. Appendix 8 compares this chapter to the results reported previously [5].

2 ARCHITECTURAL ALTERNATIVES

The three basic storage architectures we studied are magnetic disks, flash disk emulators, and flash memory cards. Their power consumption, cost, and performance are a function of the workload and the organization of the storage components. Each storage device is used in conjunction with a DRAM buffer cache. Though the buffer cache can in principle be write-back, in this chapter we consider a write-through buffer cache: this models the behavior of the Macintosh operating system and until recently the DOS file system. In most cases there is a nonvolatile SRAM buffer cache between DRAM and flash or disk, which reduces write latency dramatically.

An idle disk can consume 20–54% or more of total system energy [14, 16], so the file system must spin down the disk whenever it is idle. Misses in the buffer cache will cause a spun-down disk to spin up again, resulting in delays of up to a few seconds [6, 14]. Writes to the disk can be buffered in battery-backed SRAM, not only improving performance, but also allowing small writes to a spun-down disk to proceed without spinning it up. The Quantum Daytona is an example of a drive with this sort of buffering. In this chapter, we give magnetic disks the benefit of the doubt by simulating this deferred spin-up policy.

The flash disk organization replaces the hard disk with a flash memory card that has a conventional disk interface. With the SunDisk SDP series, one example of this type of device, transfers are in multiples of a sector (512 bytes). In contrast, the flash card organization removes the disk interface so that the memory can be accessed at byte-level. The flash card performs reads faster than the flash disk, so although the instantaneous power consumption of the two devices during a read is comparable, the flash card consumes less energy to perform the operation. Both flash devices can be used with an SRAM write buffer to reduce write latency and cleaning costs in most circumstances.

A fundamental problem introduced by flash memory is the need to *erase* an area before it can be overwritten. The flash memory manufacturer determines how much memory is erased in a single operation. The SunDisk devices erase a single 512-byte sector at a time, while the Intel Series-2 flash card erases one or two 64-Kbyte “segments.” There are two important aspects to erasure: *flash cleaning* and performance. When the segment size is larger than the transfer unit (i.e., for the flash card), any data in the segment that are still needed must be copied elsewhere. Cleaning flash memory is thus analogous to segment cleaning in Sprite LFS [20]. The cost and frequency of segment cleaning is

related in part to the cost of erasure, and in part to the segment size. The larger the segment, the more data that will likely have to be moved before erasure can take place. The system must define a policy for selecting the next segment for reclamation. One obvious discrimination metric is segment utilization: picking the next segment by finding the one with the lowest utilization (i.e., the highest amount of memory that is reusable). MFFS uses this approach [4]. More complicated metrics are possible; for example, eNVy considers both utilization and locality when cleaning flash memory [25].

The second aspect to erasure is performance. The SunDisk SDP5 flash disk couples erasure with writes, achieving a write bandwidth of 75 Kbytes/s. The time to erase and write a block is dominated by the erasure cost. The Intel flash card separates erasure from writing, and achieves a write bandwidth of 214 Kbytes/s—but only after a segment has been erased. Because erasure takes a large fixed time period (1.6s) regardless of the amount of data being erased [9], the cost of erasure is amortized over large erasure units. (The newer 16-Mbit Intel Series 2+ Flash Memory Cards erase blocks in 300ms [10], but these were not available to us during this study.) The two types of flash memory have comparable erasure bandwidth; to avoid delaying writes for erasure it is important to keep a pool of erased memory available. It becomes harder to meet this goal as more of the flash card is occupied by useful data, as discussed in Section 5.2.

Another fundamental problem with flash memory is its limited endurance. Manufacturers guarantee that a particular area within flash may be erased up to a certain number of times before defects are expected. The limit is 100,000 cycles for the devices we studied; the Intel Series 2+ Flash Memory Cards guarantee one million erasures per block [10]. While it is possible to spread the load over the flash memory to avoid “burning out” particular areas, it is still important to avoid unnecessary writes or situations that erase the same area repeatedly.

3 HARDWARE MEASUREMENTS

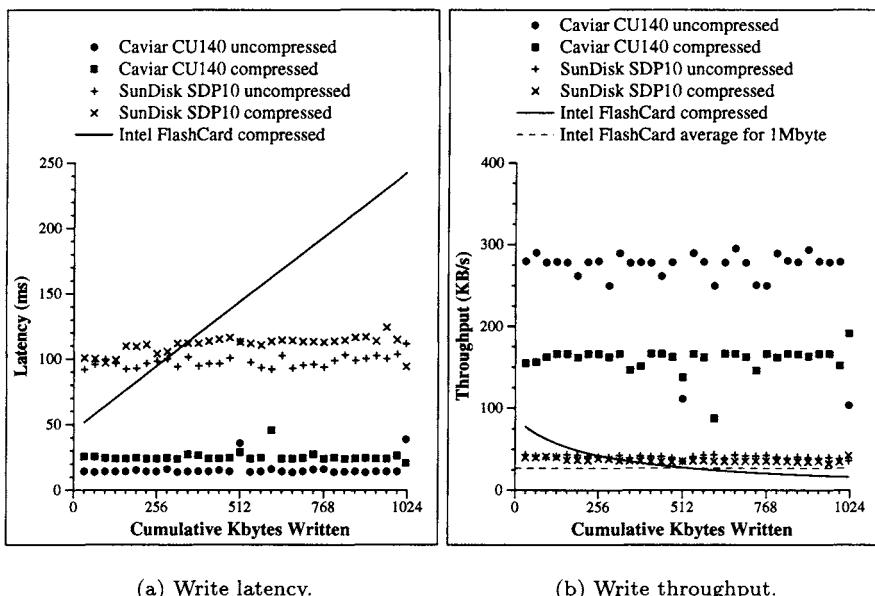
We measured the performance of the three storage organizations of interest on a Hewlett-Packard OmniBook 300. The OmniBook 300 is a 2.9-pound subnotebook computer that runs MS-DOS 5.0 and contains a 25-MHz 386SXLV processor and 2 Mbytes of DRAM. The system is equipped with several PCMCIA slots, one of which normally holds a removable ROM card containing Windows

Device	Operation	Throughput (KBytes/s)		Throughput (KBytes/s)	
		Uncompressed 4-KB file	1-MB file	Compressed 4-KB file	1-MB file
Caviar Ultralite CU140	Read	116	543	64	543
	Write	76	231	289	146
SunDisk SDP10	Read	280	410	218	246
	Write	39	40	225	35
Intel flash card	Read	645	37	345	34
	Write	43	21	83	27

Table 1: Measured performance of three storage devices on an HP OmniBook 300. The test read or wrote sequentially in units of 4 Kbytes, to either 4-Kbyte or 1-Mbyte files. For the CU140 and SDP10, we measured throughput with and without compression enabled; for the Intel card, compression was always enabled, but we distinguished between completely random data and compressible data.

and several applications. We used a 40-Mbyte Western Digital Caviar Ultralite CU140 and a 10-Mbyte SunDisk SDP10 flash disk, both of which are standard with the OmniBook, and a PCMCIA 10-Mbyte Intel Series 2 Flash Memory Card running the Microsoft Flash File System (MFFS) version 2.0 [12]. The Caviar Ultralite CU140 is compatible with PCMCIA Type III specifications, and weighs 2.7 ounces, while the flash devices are PCMCIA Type II cards weighing 1.3 ounces. Finally, the CU140 and SDP10 could be used directly or with compression, using DoubleSpace and Stacker, respectively. Compression is built into MFFS 2.00.

We constructed software benchmarks to measure the performance of the three storage devices. The benchmarks repeatedly read and wrote a sequence of files, and measured the throughput obtained. Both sequential and random accesses were performed, the former to measure maximum throughput and the latter to measure the overhead of seeks. For the CU140 and SDP10, we measured throughput with and without compression enabled; for the Intel card, compression was always enabled, but we distinguished between completely random data and compressible data. The compressible data consisted of the first 2 Kbytes of Herman Melville's well-known novel, *Moby-Dick*, repeated throughout each file (obtaining compression ratios around 50%). The Intel flash card was completely erased prior to each benchmark to ensure that writes from previous runs would not cause excess cleaning.



(a) Write latency.

(b) Write throughput.

Figure 1: Measured latency and instantaneous throughput for 4-Kbyte writes to a 1-Mbyte file. To smooth the latency when writing via DoubleSpace or Stacker, points were taken by averaging across 32-Kbytes of writes. Latency for an Intel flash card running the Microsoft Flash File System, as a function of cumulative data written, increases linearly. Though writes to the first part of the file are faster for the flash card than for the flash disk, the average throughput across the entire 1-Mbyte write is slightly worse for the flash card. The flash card was erased prior to each experiment. Also, because the cu140 was continuously accessed, the disk spun throughout the experiment.

Table 1 summarizes the measured performance for sequential 4-Kbyte reads and writes to 4-Kbyte and 1-Mbyte files, while Figure 1 graphs the average latency and instantaneous throughput for sequential 4-Kbyte writes to a 1-Mbyte file. These numbers all include DOS file system overhead. There are several interesting points to this data:

- Without compression, throughput for the magnetic disk increases with file size, as expected. With compression, small writes go quickly, because they are buffered and written to disk in batches. Large writes are compressed and then written synchronously.
- Compression similarly helps the performance of small file writes on the flash disk, resulting in write throughput greater than the theoretical limit of the SunDisk SDP10.
- Read throughput of the flash card is much better than the other devices for small files, with reads of uncompressible data obtaining about twice the bandwidth of reads of compressible data (since the software decompression step is avoided). Throughput is unexpectedly poor for reading or writing large files. This is due to an anomaly in MFFS 2.00 [12], whose performance degrades with file size. The latency of each write (Figure 1(a)) increases linearly as the file grows, apparently because data already written to the flash card are written again, even in the absence of cleaning. This results in the throughput curve in Figure 1(b).

Comparing the different devices, we see that the Caviar Ultralite cu140 provides the best sequential write throughput, since the disk is constantly spinning. Excluding the effects of compression, the flash card provides better performance than the flash disk for small files on an otherwise empty card, while its read and write performance are both worse than the flash disk for larger files. Finally, for 4-Kbyte uncompressed random I/Os (not shown), compared to sequential I/Os, the performance of the Caviar Ultralite cu140 drops substantially: it is reduced by a factor of 6 for reads and a factor of 3 for writes.

In Table 2 we include the raw performance of the devices, and power consumed, according to datasheets supplied by the manufacturers. As shown, the hard disk offers the best throughput of the three technologies, but consumes many times the power of the flash-based technologies. With regard to the two flash-based devices, the flash card offers better performance than the flash disk, while both devices offer comparable power consumption.

4 TRACE-DRIVEN SIMULATION

We used traces from several environments to do trace-driven simulation, in order to evaluate the performance and energy consumption of different storage

Device	Operation	Latency (ms)	Throughput (Kbytes/s)	Power (W)
Caviar Ultralite cu140	Read/Write	25.7	2125	1.75
	Idle	—	—	0.7
	Spin up	1000.0	—	3.0
SunDisk SDP10	Read	1.5	600	0.36
	Write	1.5	50	0.36
Intel flash card	Read	0	9765	0.47
	Write	0	214	0.47
	Erase	1600	70	0.47

Table 2: Manufacturers' specifications for three storage devices. Latency for read/write operations indicates the overhead from a random operation, excluding the transfer itself (i.e., controller overhead, seeking, or rotational latency). The Intel erasure cost refers to a separate operation that takes 1.6s to erase 64 or 128 Kbytes (in this case latency and throughput are analogous).

organizations and different storage management policies under realistic workloads. This section describes the traces and the simulator, while Section 5 describes the simulation results.

4.1 Traces

We used four workloads, MAC, DOS, HP, and SYNTH. For the MAC trace, we instrumented an Apple Macintosh PowerBook Duo 230 to capture a file system workload from a mobile computing environment. The trace is file-level: it reports which file is accessed, whether the operation is a read or write, the location within the file, the size of the transfer, and the time of the access. This trace did not record deletions. The trace was preprocessed to convert file-level accesses into disk-level operations, by associating a unique disk location with each file.

An interesting aspect of the MAC trace is the locality of write accesses: 36% of writes go to just one 1-Kbyte block and 24% of writes go to another.

We used a DOS trace collected by Kester Li at U.C. Berkeley [13], on IBM desktop PCs running Windows 3.1, also at file-level. It includes deletions. The trace was similarly preprocessed.

We used a disk-level trace collected by Ruemmler and Wilkes on an HP personal workstation running HP-UX [21]. This trace include metadata operations, which the file-level traces do not, but they are below the level of the buffer cache, so simulating a buffer cache would give misleading results (locality within the original trace has already been largely eliminated). Thus the buffer cache size was set to 0 for simulations of HP. The trace includes no deletions.

Finally, we created a synthetic workload, called SYNTH, based loosely on the *hot-and-cold* workload used in the evaluation of Sprite LFS cleaning policies [20]. The purpose of the synthetic workload was to provide both a “stress test” for the experimental testbed on the OmniBook, and a series of operations that could be executed against both the testbed and the simulator. (Unfortunately, none of our other traces accessed a small enough dataset to fit on a 10-Mbyte flash device.) The comparison between measured and simulated results appears in Section 5.1. The trace consists of 6 Mbytes of 32-Kbyte files, where $\frac{7}{8}$ of the accesses go to $\frac{1}{8}$ of the data. Operations are divided 60% reads, 35% writes, 5% erases. An erase operation deletes an entire file; the next write to the file writes an entire 32-Kbyte unit. Otherwise 40% of accesses are 0.5 Kbytes in size, 40% are between .5 Kbytes and 16 Kbytes, and 20% are between 16 Kbytes and 32 Kbytes. The interarrival time between operations was modeled as a bimodal distribution with 90% of accesses having a uniform distribution with a mean of 10ms and the remaining accesses taking 20ms plus a value that is exponentially distributed with a mean of 3s.

Though only the MAC trace comes from a mobile environment, the two desktop traces represent workloads similar to what would be used on mobile computers, and have been used in simulations of mobile computers in the past [13, 14, 15]. Table 3 lists additional statistics for the nonsynthetic traces.

4.2 Simulator

Our simulator models a storage hierarchy containing a buffer cache and non-volatile storage. The buffer cache is the first level searched on a read and is the target of all write operations. The cache is write-through to non-volatile storage, which is typical of Macintosh and some DOS environments². A write-back cache might avoid some erasures at the cost of occasional data loss. In addition, the buffer cache can have zero size, in which case reads and writes

²DOS supports a write-back cache, but after users complained about losing data, they were given a choice of write-through or write-back caching.

	MAC			DOS		
Applications	Finder, Excel, Newton Toolkit			Framemaker, Powerpoint, Word		
Duration	3.5 hours			1.5 hours		
Number of distinct Kbytes accessed	22000			16300		
Fraction of reads	0.50			0.24		
Block size (Kbytes)	1			0.5		
Mean read size (blocks)	1.3			3.8		
Mean write size (blocks)	1.2			3.4		
Inter-arrival time (s)	Mean	Max	σ	Mean	Max	σ
	0.078	90.8	0.57	0.528	713.0	10.8

	HP		
Applications	email, editing		
Duration	4.4 days		
Number of distinct Kbytes accessed	32000		
Fraction of reads	0.38		
Block size (Kbytes)	1		
Mean read size (blocks)	4.3		
Mean write size (blocks)	6.2		
Inter-arrival time (s)	Mean	Max	σ
	11.1	30min	112.3

Table 3: Summary of (non-synthetic) trace characteristics. The statistics apply to the 90% of each trace that is actually simulated after the warm start. Note that it is not appropriate to compare performance or energy consumption of simulations of different traces, because of the different mean transfer sizes and durations of each trace.

go directly to non-volatile storage. The HP-UX trace is simulated with a zero buffer cache, as the traces include the effects of a buffer cache.

An intermediate level containing battery-backed SRAM is used to buffer writes. The purpose of SRAM is primarily to hide the latency of the mass-storage devices (disk or flash) when doing a synchronous write. As mentioned above, it also serves to buffer small intermittent writes to a spun-down disk. Similarly, if there are “hot spots” that are repeatedly written, SRAM can reduce the number of operations that ultimately go to disk or flash. In addition, misses in DRAM may be satisfied by data in SRAM, though this scenario is unlikely if the size of DRAM is large relative to the SRAM buffer. SRAM is an integral part of many magnetic disk systems and is also a major component of the eNVy main-memory storage system [25].

We simulated the disk, flash disk, and flash card devices with parameters for existing hard disk, flash memory disk emulator, and flash memory card products, respectively. Each device is described by a set of parameters that include the power consumed in each operating mode (reading, writing, idle, or sleeping) and the time to perform an operation or switch modes. The power specifications came from datasheets; two different set of performance specifications were used, one from the measured performance and one from datasheets.

In addition to the products described in Section 3, we used the datasheet for the NEC μ PD4216160/L 16-Mbit DRAM chip and μ PD43256B 32Kx8-bit SRAM chip [18]. In the case of the SunDisk device, the simulation using raw (nonmeasured) performance numbers is based upon the SunDisk SDP5 and SDP5A devices, which are newer 5-volt devices [3]. Lastly, we also simulated the Hewlett-Packard Kittyhawk 20-Mbyte hard disk, which we refer to as KH, based on its datasheet [8]. In order to manage all the traces, we simulated flash devices larger than the 10-Mbyte PCMCIA flash devices we had for the OmniBook. Based on the characteristics of different-sized Intel flash cards, the variation in power and performance among flash cards of different size are insignificant.

For each trace, 10% of the trace was processed in order to “warm” the buffer cache, and statistics were generated based on the remainder of the trace.

The simulator accepts a number of additional parameters. Those relevant to this study are:

Flash size The total amount of flash memory available.

Flash segment size	The size of an erasure unit.
Flash utilization	The amount of data stored, relative to flash size. The data are preallocated in flash at the start of the simulation, and the amount of data accessed during the simulation must be no greater than this bound.
Cleaning policy	On-demand cleaning, as with the SunDisk SDP5, and asynchronous cleaning, as with the Flash File System running on the Intel flash card. Flash cleaning is discussed in greater detail below.
Disk spin-down policy	A set of parameters control how the disk spins down when idle and how it spins up again when the disk is accessed.
DRAM size	The amount of DRAM available for caching.
SRAM size	The amount of SRAM available for buffering writes.

We made a number of simplifying assumptions in the simulator:

- All operations and state transitions are assumed to take the average or “typical” time, either measured by us or specified by the manufacturer.
- Repeated accesses to the same file are assumed never to require a seek (if the transfer is large enough to require a seek even under optimal disk layout, the cost of the seek will be amortized); otherwise, an access incurs an average seek. Each transfer requires the average rotational latency as well. These assumptions are necessary because file-level accesses are converted to disk block numbers without the sophistication of a real file system that tries to optimize block placement.
- For flash file systems, while file data and metadata that would normally go on disk are stored in flash, the data structures for the flash memory itself are managed by the simulator but not explicitly stored in flash, DRAM, or SRAM. In the case of the SunDisk SDP5 flash device, there is no need for additional data structures beyond what the file system already maintains for a magnetic disk and the flash disk maintains internally for block remapping. For the Intel flash card, the flash metadata includes state that must be frequently rewritten, such as linked lists. These would be stored in SRAM in a real system; for comparison purposes, the SRAM needed to maintain page tables for flash memory in eNVy increased total system cost by 10% [25].

- For the flash card, the simulator attempts to keep at least one segment erased at all times, unless erasures are done on an as-needed basis. One segment is filled completely before data blocks are written to a new segment. Erasures take place in parallel with reads and writes, being suspended during the actual I/O operations, unless a write occurs when no segment has erased blocks.
- In general, I/Os do not take place in parallel. This assumption simplifies the simulator substantially, and is valid because we must assume that an operation in the trace *may* depend on any preceding operation. Operations such as SRAM or flash cleaning take place between operations dictated by the trace, with the exception of flash cleaning, which is interrupted by any other flash operations and subsequently resumed. Cleaning takes place only when it can conclude before the next I/O occurs or when SRAM or flash is full; in these latter cases, our simulated response-time results overestimate the impact of cleaning because in some cases the simulator will have missed an opportunity for parallelism.

5 RESULTS

We used the simulator to explore the architectural tradeoffs between disks, flash disks, and flash cards. We focussed on four issues: the basic energy and performance differences between the devices; the effect of storage utilization on flash energy consumption, performance, and endurance; the effect of combined writes and erasures on a flash disk; and the effect of buffer caches, both volatile and nonvolatile, on energy and performance.

5.1 Basic Comparisons

Tables 4(a)–(c) show for three traces and each device the energy consumed, and the average, mean, and standard deviations of the read and write response times. As mentioned in Section 4.2, the input parameters for each simulation were either based on measurements on the OmniBook (labeled “measured”) or manufacturers’ specifications (labeled “datasheet”). Note that it is not appropriate to compare response time numbers between the tables, because of the different mean transfer sizes of each trace. Simulations of the magnetic disks spun down the disk after 5s of inactivity, which is a good compromise between energy consumption and response time [6, 14]. Simulations using the flash card were done with the card 80% full.

Device	Energy	Read Response			Write Response		
		Mean	Max	σ	Mean	Max	σ
cu140 (M)	7,287	3.10	3535.3	55.1	6.50	3548.5	90.4
cu140 (D)	6,177	2.51	3519.5	52.9	6.35	3505.2	88.5
KH (D)	5,295	3.33	1673.1	58.6	8.83	1726.5	115.3
SDP10 (M)	306	0.48	1001.7	7.0	3.64	566.7	42.3
SDP5 (D)	247	0.34	619.9	4.3	2.24	340.0	25.5
Intel (M)	198	0.33	665.6	4.6	3.72	568.7	43.8
Intel (D)	84	0.11	95.8	0.7	0.86	117.0	9.0

(a) MAC trace

Device	Energy	Read Response			Write Response		
		Mean	Max	σ	Mean	Max	σ
cu140 (M)	4,556	15.37	2743.8	97.2	3.72	1004.5	53.4
cu140 (D)	3,323	12.23	2712.3	95.6	3.82	1005.0	53.4
KH (D)	1,698	16.41	1559.3	128.1	5.74	1536.8	81.9
SDP10 (M)	616	2.94	120.2	5.6	25.97	264.5	19.6
SDP5 (D)	522	1.98	77.5	3.6	15.53	160.2	11.8
Intel (M)	550	1.96	80.8	3.8	26.32	310.6	21.1
Intel (D)	339	0.51	17.0	0.8	5.39	65.9	4.2

(b) DOS trace

Device	Energy	Read Response			Write Response		
		Mean	Max	σ	Mean	Max	σ
cu140 (M)	93,478	106.87	3537.6	256.9	74.89	3581.1	237.3
cu140 (D)	63,285	83.23	3499.2	255.3	64.70	3514.8	233.8
KH (D)	21,785	106.48	1620.9	325.6	87.29	1667.3	315.1
SDP10 (M)	3,315	10.69	40.4	7.0	100.80	5763.2	116.9
SDP5 (D)	2,942	6.55	24.9	4.2	60.30	3441.3	69.8
Intel (M)	1,869	6.67	24.8	4.5	100.65	6044.8	121.0
Intel (D)	1,028	0.43	1.6	0.3	19.88	1226.9	24.5

(c) HP trace

Table 4: Comparison of energy consumption and response time for different devices, using the MAC, DOS, and HP traces. There was a 2-Mbyte DRAM buffer for MAC and DOS but no DRAM buffer cache in the HP simulations; each simulation had a 32-Kbyte SRAM write buffer. Disk simulations spun down the disk after 5s of inactivity. Flash simulations were done with flash memory 80% utilized. *Measured (M)* devices refer to performance characteristics that were derived from microbenchmarks using the devices, while *datasheet (D)* devices came directly from manufacturers' specifications. The datasheet parameters tend to underestimate overhead, but provide a more consistent base for comparison.

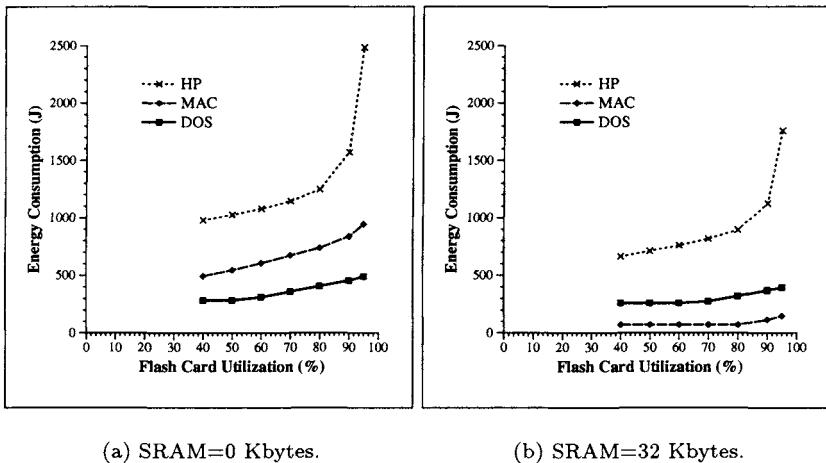
Based solely on the input parameters from the datasheets, one may conclude that the Intel flash card consumes significantly less energy than either the Caviar Ultralite CU140 or the SunDisk SDP5. Its mean read response time is 1–2 orders of magnitude faster than the Caviar Ultralite CU140 and 3–15 times faster than the SunDisk SDP5. The flash card’s mean write performance is about three times better than the SunDisk SDP5, and its write performance is much better than the Caviar Ultralite CU140 or KH for the MAC and HP traces. For the DOS trace, however, writing to the flash card is about 40% slower on average than the Caviar Ultralite CU140, and is just 6% faster than the KH on average.

When using the numbers for measured performance as input to the simulator, the flash card does not outperform the flash disk nearly as dramatically. The mean read performance of the flash card is 45–60% better than the flash disk, compared to being 3–15 times faster based on the datasheets. The mean write performance of the flash card is comparable to that of the flash disk.

We verified the simulator by running a 6-Mbyte synthetic trace both through the simulator and on the OmniBook, using each of the devices. The trace was smaller than the ones described above, in order to fit on the 10-Mbyte flash devices. We used the measured micro-benchmark performance to drive the simulator and then compared against actual performance. All simulated performance numbers were within a few percent of measured performance, with the exception of flash card reads, which was four times worse than the simulated performance. We believe this is due to overhead from cleaning and from decompression, which are more severe in practice than during the controlled experiments described in Section 3.

5.2 Flash Storage Utilization

For the Intel flash card, there is a substantial interaction between the storage utilization of flash memory and the behavior of the flash when the flash is frequently written. To examine this behavior, we simulated each trace with 40% to 95% of flash memory occupied by useful data. To do this, we set the size of the flash to be large relative to the size of the trace, then filled the flash with extra data blocks that reduced the amount of free space by an appropriate amount. Under low utilization, energy consumption and performance are fairly constant, but as the flash fills the system must copy increasing amounts of “live” data from one erasure unit to another to free up an entire erasure unit. Eventually flash cleaning consumes enough time that it can no longer be



(a) SRAM=0 Kbytes.

(b) SRAM=32 Kbytes.

Figure 2: Energy consumption as a function of flash storage utilization, simulated based on the datasheet for the Intel flash card, with a segment size of 128 Kbytes. The SRAM buffer was 0 or 32 Kbytes. Energy consumption increases steadily for each of the traces, due to increased cleaning overhead, but the energy consumed by the HP trace increases the most dramatically with high utilization. The size of the DRAM buffer cache was 2 Mbytes for MAC and DOS and no DRAM was used for HP.

done asynchronously with respect to new writes. Eventually the behavior of the flash degrades with increasing utilization, resulting in much greater energy consumption, worse performance, and more erasures per unit time (thus affecting flash endurance). By comparison, the flash disk is unaffected by utilization because it does not copy data within the flash.

Figures 2 and 3 graph simulated energy consumption and write response time as a function of storage utilization for each trace, using the specifications from the Intel flash card datasheet and a 2-Mbyte DRAM cache (no DRAM cache for the HP trace), and either no SRAM or 32 Kbytes of SRAM. With either amount of SRAM, at a utilization of 95%, compared to 40% utilization, the energy consumption rises by up to 165%. SRAM has a greater effect on write response time: without an SRAM write buffer, the average write time increases by up to 24%, but with a small buffer the mean write response time increases negligibly for the MAC and HP traces and by about 20% for the DOS trace. For

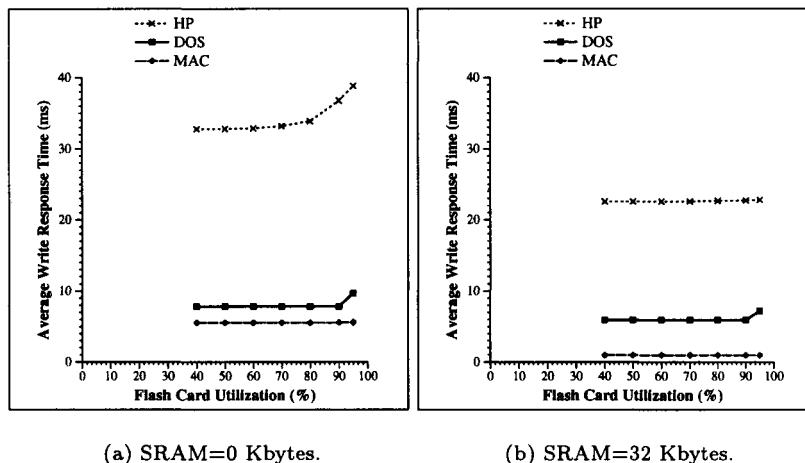


Figure 3: Mean write response time as a function of flash storage utilization, simulated based on the datasheet for the Intel flash card, with a segment size of 128 Kbytes. The SRAM buffer was 0 or 32 Kbytes. Performance holds steady until utilization is high enough for writes to be deferred while waiting for a clean segment; even so, the MAC trace has constant mean write response. It rewrites some blocks repeatedly, so the cleaner keeps up with writes more easily. The size of the DRAM buffer cache was 2 Mbytes for MAC and DOS and no DRAM was used for HP.

the HP trace (which writes the most data), even with the SRAM buffer, the mean number of times any given segment was erased increased from 0.7 to 2.5 (a 260% increase) and the maximum number of erasures for any one segment over the course of the simulation increased from 10 to 39. This effect was less dramatic for the other traces, especially with an SRAM buffer to absorb multiple writes to the same location. Still, higher storage utilizations can result in “burning out” the flash two to three times faster under a UNIX workload.

In addition, experiments on the OmniBook demonstrated significant reductions in write throughput as flash memory was increasingly full. Figure 4 graphs instantaneous throughput as a function of cumulative data written, with three amounts of “live” data in the file system: 1 Mbyte, 9 Mbytes, and 9.5 Mbytes. Each data point corresponds to 1 Mbyte of data being overwritten, randomly selected within the total amount of live data. The flash card was erased com-

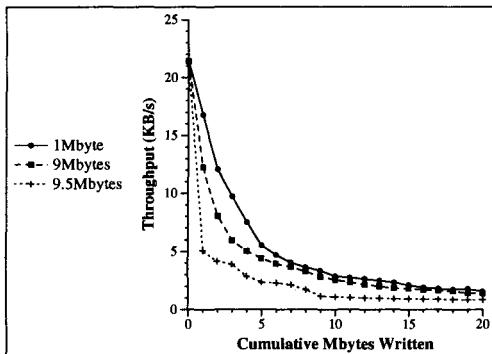


Figure 4: Measured throughput on an OmniBook using a 10-Mbyte Intel flash card, for each of 20 1-Mbyte writes (4 Kbytes at a time). Different curves show varying amounts of live data. Throughput drops both with more cumulative data and with more storage consumed.

pletely prior to each experiment, so that any cleaning overhead would be due only to writes from the current experiment and the experiments would not interfere with each other. The drop in throughput over the course of the experiment is apparent for all three configurations, even the one with only 10% space utilization, presumably because of MFFS 2.00 overhead. However, throughput decreased much faster with increased space utilization.

5.3 Asynchronous Cleaning

The next generation³ of SunDisk flash products, the SDP5A, has the ability to erase blocks prior to writing them, in order to get higher bandwidth during the write [3]. Erasure bandwidth is 150 Kbytes/s regardless of whether new data are written to the location being erased; however, if an area has been pre-erased, it can be written at 400 Kbytes/s (i.e., write latency is reduced by 62.5%).

We simulated to compare the SDP5A with and without asynchronous cleaning. Asynchronous cleaning has minimal impact on energy consumption, but it de-

³This device was not available when the majority of this study was performed. Therefore we generally consider the SunDisk SDP5 except in this context.

creases the mean write response time for each of the traces by 55–65%. This reduction applies both to cases where there is no SRAM buffer and to configurations with a small (32-Kbyte) buffer, indicating that a small amount of SRAM is not sufficient to hide the latency of cleaning. In fact, a larger SRAM buffer of 1 Mbyte results in asynchronous cleaning having an even greater benefit, reducing mean write latency by 80–90%. The difference in this case is the result of writes coming into the SRAM buffer faster than they can be flushed to the flash disk if the flash disk is erasing and writing each block in a single operation. With a large SRAM buffer and asynchronous erasure, the flash disk can keep up with the SRAM buffer.

The improvement experienced by asynchronous erasure on the SunDisk demonstrates the effect of small erasure units on performance. Considering again the simulated write response of the SunDisk SDP5 and Intel flash card shown in Tables 4(a)–(c), based on datasheets, if the SunDisk SDP5 write response decreased by 60% it would be comparable to the flash card. But as storage utilization increases, flash card write performance will degrade although the performance of the flash disk will remain constant.

5.4 DRAM Caching

Since flash provides better read performance than disk, the dynamics of using DRAM for caching file data change. DRAM provides better performance than flash but requires more power and is volatile. Unlike flash memory, DRAM consumes significant energy even when not being accessed. Thus, while extremely “hot” read-only data should be kept in DRAM to get the best read performance possible, other data can remain in flash rather than DRAM. One may therefore ask whether it is better to spend money on additional DRAM or additional flash. In order to evaluate these trade-offs, we simulated configurations with varying amounts of DRAM buffer cache and flash memory. (As is the case with all our simulations, they do not take into account DRAM that is used for other purposes such as program execution.) We began with the premise that a system stored 32 Mbytes of data, not all of which necessarily would be accessed, and considered hypothetical flash devices storing from 34–38 Mbytes of data. (Thus total storage utilization ranged from 94% with 34 Mbytes of storage down to 84% with 38 Mbytes.) In addition, the system could have from 0–4 Mbytes of DRAM for caching.

Figure 5 shows the results of these simulations, run against the DOS trace using specifications from the datasheets, and 32 Kbytes of SRAM. For the Intel flash

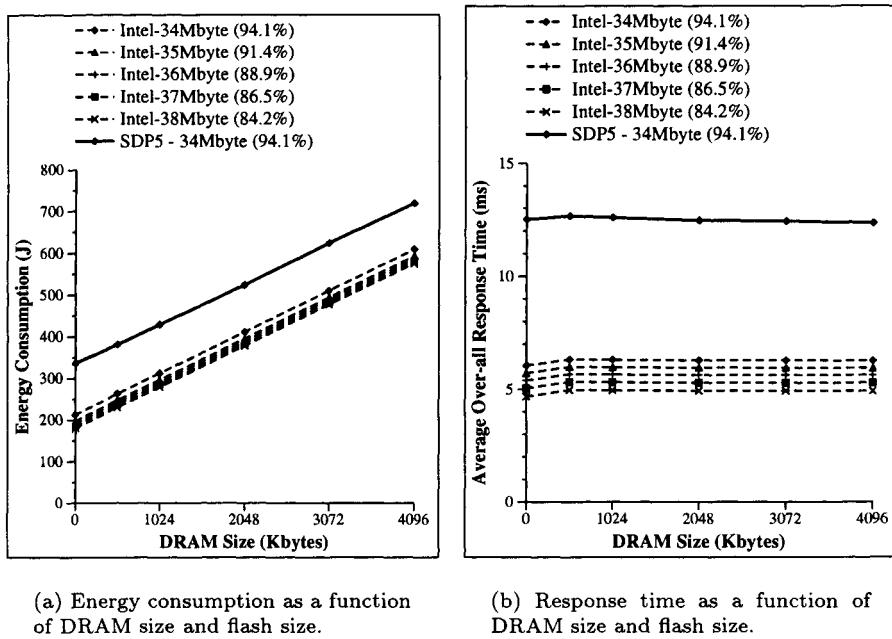


Figure 5: Energy consumption and average over-all response time as a function of DRAM size and flash size, simulated for the DOS trace with an SRAM buffer size of 32 Kbytes. We simulated multiple flash sizes for the Intel flash card, which shows a benefit once it gets below 90% utilization. Each line represents a 1-Mbyte differential in flash card size, similar to moving along the x-axis by 1 Mbyte of DRAM. Increasing the DRAM buffer size has no benefit for the Intel card; the slight increase in response time when moving from no DRAM cache to a 512-Kbyte cache comes from copying data multiple times between media with similar access times. The SunDisk has no benefit due to increased flash size (not shown), and here for this trace it shows little benefit from a larger buffer cache either.

card, increasing the amount of flash available by 1 Mbyte, thereby decreasing storage utilization from 94.1% to 91.4%, reduces energy consumption by 7% and average over-all response time by 6%. (The differences are greater when there is no SRAM cache [5].) The incremental benefit on energy consumption of additional flash beyond the first Mbyte is minimal, though adding flash does

help to reduce response time. Adding DRAM to the Intel flash card increases the energy used for DRAM without any appreciable benefits: the time to read a block from flash is barely more than the time to read it from DRAM.

Only one curve is shown for the SunDisk SDP5 because increasing the size of the flash disk has minimal effect on energy consumption or performance. In fact, for this trace, even a 500-Kbyte DRAM cache increases energy consumption for the SunDisk SDP5 without improving performance. With the MAC trace, which has a greater fraction of reads, a small DRAM cache improves energy consumption and performance for the SunDisk SDP5, while the Intel flash card utilization has a greater impact on energy consumption (a 17% reduction from 1 Mbyte of flash) and no impact at all on response time. Thus the trade-off between DRAM and flash size is dependent both on execution characteristics (read/write ratio) and hardware characteristics (any differences in performance between DRAM and the flash device).

5.5 SRAM Buffering

So far we have generally assumed that a system has a small amount of SRAM, specifically 32 Kbytes, to buffer writes to disk or flash. The quantity of SRAM is a tradeoff between cost and performance. A 32-Kbyte SRAM write buffer costs only a few dollars, which is a small part of the total cost of a storage system. Under light load, this buffer can make a significant difference in the average write response time, compared to a system that writes all data synchronously to secondary store. Although SRAM consumes significant energy itself, by reducing the number of times the disk spins up or flash memory is cleaned, the SRAM buffer can potentially conserve energy. However, if writes are large or are clustered in time, such that the write buffer frequently fills, then many writes will be delayed as they wait for the slower device. In this case, a larger SRAM buffer will be necessary to improve performance, and it will cost more money and consume more energy.

In practice, small SRAM write buffers for magnetic disks are relatively commonplace, though we are unaware of products other than the Quantum Daytona that use the write buffer to avoid spinning up an idle disk. Here we examine the impact of nonvolatile memory in a storage system. We base our results on a NEC 32Kx8-bit SRAM chip, part μ PD43256B, with a 55ns access time [19].

Figures 6–9 graph normalized energy consumption and write response time as a function of SRAM size for each of the traces and devices. The values are

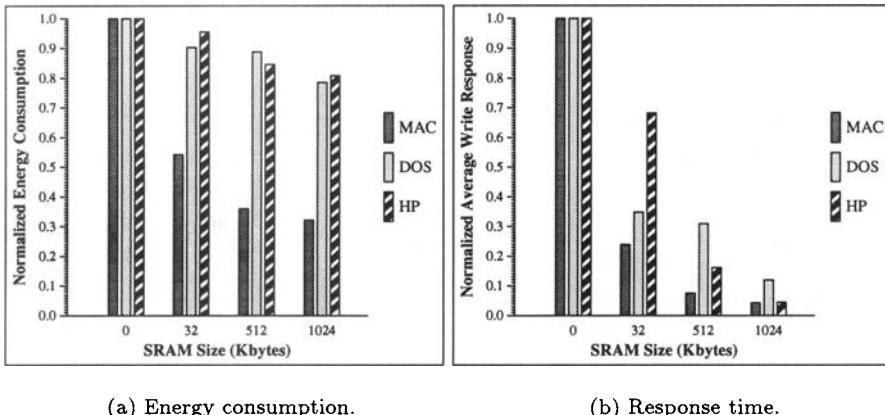


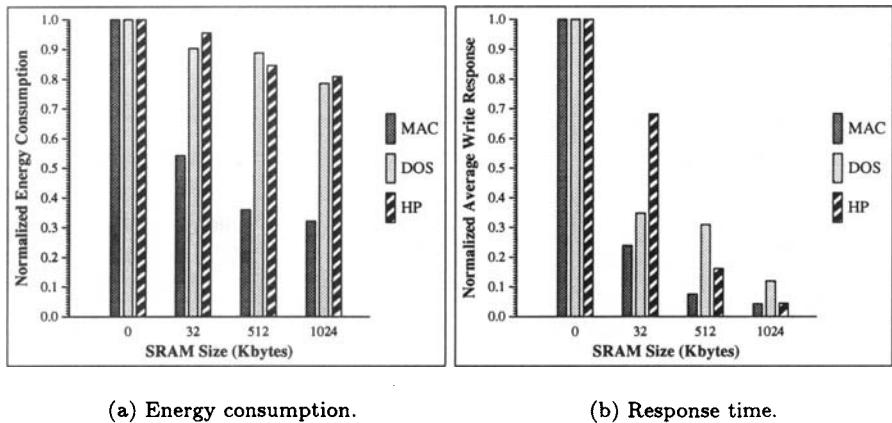
Figure 6: Normalized energy and write response time as a function of SRAM size for each trace, for the CU140. Results are normalized to the value corresponding to no SRAM. The MAC trace shows the greatest benefit in energy consumption, while all traces benefit substantially from a 32-Kbyte SRAM and even more from larger write buffers. The disk was spun down after 5s of inactivity. The size of the DRAM buffer cache was 2 Mbytes for MAC and DOS and no DRAM was used for HP.

normalized to the case without an SRAM buffer. As with the other experiments, DRAM was fixed at 2 Mbytes for MAC and DOS and not used for HP;⁴ the spin-down threshold was fixed at 5s. The results may be summarized as follows:

CU140 (Figure 6)

- Adding SRAM has minimal effect on energy consumption for the DOS and HP traces but even a 32-Kbyte buffer reduces energy consumption for the MAC trace by about half.

⁴For this experiment, one should discount the result from the HP trace by comparison to the other two traces. This is because the HP simulation has no DRAM cache, so reads cause the disk to spin up more than with the other simulations (except those reads that are serviced from recent writes to SRAM). The effect of SRAM on energy and response time in the HP environment bears further study.



(a) Energy consumption.

(b) Response time.

Figure 7: Normalized energy and write response time as a function of SRAM size for each trace, for the Intel flash card with a utilization of 80%. Results are normalized to the value corresponding to no SRAM. The MAC trace shows the greatest benefit in energy consumption and write response time, but all traces benefit in both respects from increasing SRAM. The size of the DRAM buffer cache was 2 Mbytes for MAC and DOS and no DRAM was used for HP.

- All the traces benefit with increasing amounts of SRAM as far as write response is concerned; a 1-Mbyte SRAM write buffer decreases mean write response time by about an order of magnitude for each trace, but even just 32 Kbytes has a large effect (30–80% reduction).

Intel flash card (Figure 7)

- A small write buffer reduces energy consumption for the MAC trace by 90%, and adding additional SRAM has no effect on energy consumption. Increasing SRAM sizes are beneficial for the other two traces, with a 1-Mbyte buffer reducing energy consumption by 50% for the DOS trace and 75% for the HP trace. By absorbing overwrites, SRAM reduces energy by dramatically reducing the number of flash erasures; this can be seen for example in Figure 8, which graphs normalized energy consumption and normalized erasures for the HP trace.

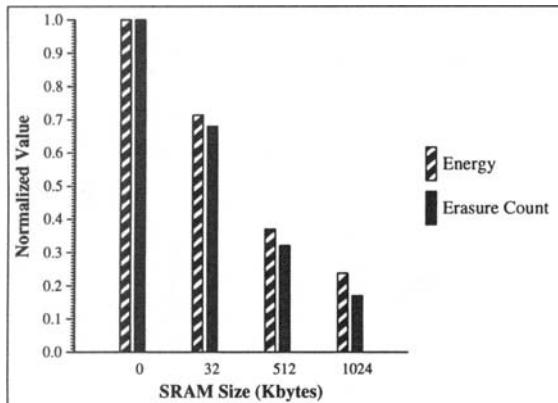


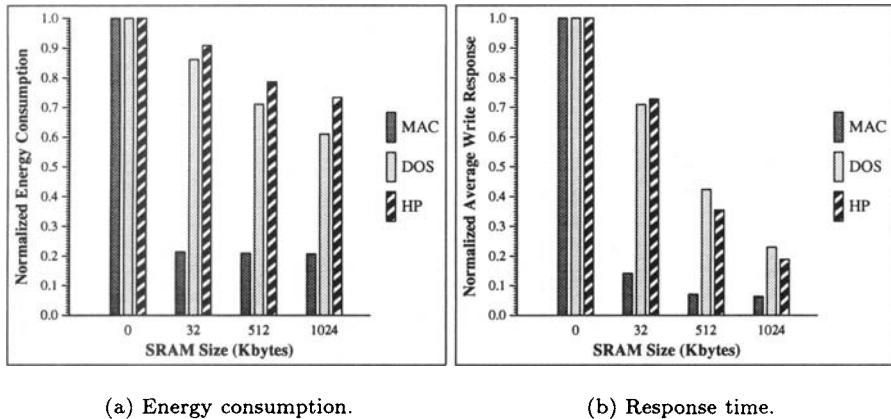
Figure 8: Normalized energy and erasure count as a function of SRAM size for the HP trace running on the Intel flash card with a utilization of 80%, and no DRAM cache. Results are normalized to the value corresponding to no SRAM. The two values follow similar curves, suggesting that the reduction in erasures has a direct impact on the reduction in overall energy consumption.

- Similarly, a small buffer has the biggest impact on write response time for the MAC trace. Again, a 1-Mbyte buffer shows reductions of about 80–95% in mean write response.

SDP5 (Figure 9)

The results for the SDP5 are similar to those for the Intel flash card:

- Energy consumption for the MAC trace with 32 Kbytes of SRAM drops to 20% of the configuration without SRAM, without further improvement from more SRAM. The HP trace shows less relative benefit than the DOS trace, unlike the flash card configuration, but both improve somewhat with increasing SRAM size.
- The mean write response times for each trace decrease with increasing SRAM size, up to a total reduction of about 75–95%.



(a) Energy consumption.

(b) Response time.

Figure 9: Normalized energy and write response time as a function of SRAM size for each trace, for the **SDP5**. Results are normalized to the value corresponding to no SRAM. Conclusions are similar to the preceding figure. The size of the DRAM buffer cache was 2 Mbytes for MAC and DOS and no DRAM was used for HP.

It is not surprising that the DOS trace would show the least benefit from increasing SRAM size with respect to energy consumption, since it has a small fraction of writes. The huge benefits from even a small SRAM buffer for the MAC trace are indicative of the extent of write locality in the trace.

6 RELATED WORK

In addition to the specific work on flash file systems mentioned previously, the research community has begun to explore the use of flash memory as a substitute for, or an addition to, magnetic disks. Cáceres et al. proposed operating system techniques for exploiting the superior read performance of flash memory while hiding its poor write performance, particularly in a portable computer where all of DRAM is battery-backed [2]. Wu and Zwaenepoel discussed how to implement and manage a large non-volatile storage system, called eNvY, composed of SRAM and flash memory for high-performance transaction processing. They simulated a system with Gbytes of flash memory and Mbytes

of battery-backed SRAM, showing it could support the I/O corresponding to 30,000 transactions per second using the TPC-A database benchmark [24]. They found that at a utilization of 80%, 45% of the time is spent erasing or copying data within flash, while performance was severely degraded at higher utilizations [25]. Marsh et al. examined the use of flash memory as a cache for disk blocks to avoid accessing the magnetic disk, thus allowing the disk to be spun down more of the time [15]. SunDisk recently performed a competitive analysis of several types of flash memory on an HP Omnibook 300 and found that the SunDisk SDP5-10 flash disk emulator was nearly an order of magnitude faster than an Intel Flash card using version 2 of the Flash File System [23]. They also found that performance of the Intel Flash card degraded by 40% as it filled with data, with the most noticeable degradation between 95% and 99% storage utilization.

Other researchers have explored the idea of using non-volatile memory to reduce write traffic to disk. Baker et al. found that some 78% of blocks written to disk were done so for reliability. They found that a small amount of battery-backed SRAM on each client was able to reduce client-server file write traffic by half, and SRAM on the file server could reduce writes to disk by 20% [1]. However, the benefits of SRAM for workstation clients did not justify its additional cost, which would be better applied toward additional DRAM. This contrasts with our results for a mobile environment, in which larger amounts of DRAM are not so cost effective, but a small amount of SRAM helps energy consumption and performance. Ruemmler and Wilkes also studied how well SRAM could absorb write traffic, finding that 4 Mbytes of SRAM was sufficient to absorb 95% of all write traffic in the systems they traced [21].

Segment cleaning in Rosenblum and Ousterhout's Log-Structured File System (LFS) [20] has a number of similarities to flash cleaning when the flash segment size is a large multiple of the smallest block size. The purpose of Sprite LFS is to amortize write overhead by writing large amounts of data at once; to do so requires that large amounts of contiguous disk space be emptied prior to a write. However, cleaning in LFS is intended to amortize the cost of seeking between segments anywhere on the disk, while flash cleaning is a requirement of the hardware.

Finally, Kawaguchi et al. [11] recently designed a flash memory file system for UNIX based on LFS, with performance comparable to the 4.4BSD Pageable Memory based File System [17]. They found that at low utilizations cleaning overhead did not significantly affect performance, but the throughput of random writes dropped from 222 Kbytes/s to 40 Kbytes/s when utilization increased from 30% to 90%. A reduction of 82% in write throughput is comparable to

the results we measured empirically with MFFS 2.00, but much worse than our simulated mean write response time. This is because our traces did not place a long sustained load on the system, instead allowing the flash cleaning process to keep up at least some of the time.

7 CONCLUSIONS

In this chapter we have examined three alternatives for file storage on mobile computers: a magnetic disk, a flash disk emulator, and a flash memory card. We have shown that either form of flash memory is an attractive alternative to magnetic disk for file storage on mobile computers. Flash offers low energy consumption, good read performance, and acceptable write performance. In addition, a nonvolatile SRAM write buffer is necessary with each configuration to hide write latency. SRAM is especially important for magnetic disks, which suffer delays due to spin-ups, and flash cards, which suffer delays due to cleaning and erasure.

The main disadvantage of using magnetic disk for file storage on mobile computers is its great energy consumption. To extend battery life, the power management of a disk file system spins down the disk when it is idle. But even with power management, a disk file system can consume an order of magnitude more energy than a file system using flash memory.

Our trace simulation results, using a SunDisk SDP5, a Caviar Ultralite CU140, and a 32-Kbyte SRAM write buffer, show that the flash disk file system can save 84–96% of the energy of the disk file system. It is 6–13 times faster for reads, while its mean write response varies from four times worse to three times better. The differences in write response time arise from the rate at which writes arrived in the traces: for the DOS trace, in which the SunDisk SDP5 fared the worst, many more writes to SRAM had to wait for blocks to be written to the flash disk than to the magnetic disk, because once the magnetic disk was spinning it could process writes faster.

The flash memory file system (using the Intel flash card) has the most attractive qualities with respect to energy and performance, though its price and capacity limitations are still drawbacks. Even in the presence of disk power management, the flash memory file system can reduce energy consumption by nearly two orders of magnitude compared to the magnetic disk file system, extending battery life by 25–115%. Furthermore, in theory the flash memory

file system can provide mean read response time that is up to two orders of magnitude faster than the disk file system. As with the SunDisk SDP5, the mean write response of the Intel flash card varies significantly compared to the Caviar Ultralite CU140: from just 14% of the disk's mean response using the MAC trace to 41% *more* than the disk using the DOS trace. Again, the differences among the traces can be attributed to raw throughput to a spinning disk versus waiting for a disk to spin up.

In practice, hardware measurements showed that there is a great discrepancy between the rated performance of each of the storage media and their performance in practice under DOS. This is especially true with the flash card using MFFS 2.00, whose write performance degrades linearly with the size of the file. Some of the differences in performance can be reduced with new technologies, in both hardware and software. One new technique is to separate the write and erase operations on a flash disk emulator, as the next generation of the SunDisk flash disk will allow. Another hardware technique is to allow erasure of more of a flash memory card in parallel, as the newer 16-Mbit Intel flash devices allow [10]. Newer versions of the Microsoft Flash File System should address the degradation imposed by large files, and in order to take advantage of asynchronous flash disk erasure, file systems for mobile computers must treat the flash disk more like a flash card than like a magnetic disk.

Finally, in our simulation study, we found that the erasure unit of flash memory, which is fixed by the hardware manufacturer, can significantly influence file system performance. Large erasure units require a low space utilization. At 90% utilization or above, an erasure unit that is much larger than the file system block size will result in unnecessary copying, degrading performance, wasting energy, and wearing out the flash device. In our simulations, energy consumption rose by as much as 165%, the average write response increased up to 20%, and the rate of erasure nearly tripled. Flash memory that is more like the flash disk emulator, with small erasure units that are immune to storage utilization effects, will likely grow in popularity despite being at a disadvantage in basic power and performance.

Acknowledgments

We thank W. Sproule and B. Zenel for their efforts in gathering trace data and/or hardware measurements. J. Wilkes at Hewlett-Packard and K. Li at U.C. Berkeley graciously made their file system traces available. Thanks to R.

Alonso, M. Dahlin, C. Dingman, B. Krishnamurthy, P. Krishnan, D. Milojičić, C. Northrup, J. Sandberg, D. Stodolsky, B. Zenel, W. Zwaenepoel, and anonymous reviewers for comments on previous drafts. We thank the following persons for helpful information about their products: A. Elliott and C. Mayes of Hewlett-Packard; B. Dipert and M. Levy of Intel; and J. Craig, S. Gross, and L. Seva of SunDisk.

REFERENCES

- [1] Mary Baker, Satoshi Asami, Etienne Deprit, John Ousterhout, and Margo Seltzer. Non-volatile memory for fast, reliable file systems. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 10–22, Boston, MA, October 1992. ACM.
- [2] Ramón Cáceres, Fred Douglis, Kai Li, and Brian Marsh. Operating Systems Implications of Solid-State Mobile Computers. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 21–27, Napa, CA, October 1993. IEEE.
- [3] Jeff Craig, March 1994. Personal communication.
- [4] Brian Dipert and Markus Levy. *Designing with Flash Memory*. Annabooks, 1993.
- [5] Fred Douglis, Ramón Cáceres, Brian Marsh, Frans Kaashoek, Kai Li, and Joshua Tauber. Storage alternatives for mobile computers. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 25–37. USENIX Association, November 1994.
- [6] Fred Douglis, P. Krishnan, and Brian Marsh. Thwarting the Power Hungry Disk. In *Proceedings of 1994 Winter USENIX Conference*, pages 293–306, San Francisco, CA, January 1994.
- [7] Hewlett-Packard. *HP 100 and OmniBook Flash Disk Card User's Guide*, 1993.
- [8] Hewlett-Packard. *Kittyhawk HP C3013A/C3014A Personal Storage Modules Technical Reference Manual*, March 1993. HP Part No. 5961-4343.
- [9] Intel. *Mobile Computer Products*, 1993.
- [10] Intel. *Flash Memory*, 1994.
- [11] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda. A flash-memory based file system. In *Proceedings of the USENIX 1995 Technical Conference on UNIX and Advanced Computing Systems*, pages 155–164, New Orleans, LA, January 1995.
- [12] Markus Levy. Interfacing Microsoft's Flash File System. In *Memory Products*, pages 4–318–4–325. Intel Corp., 1993.

- [13] Kester Li. Towards a low power file system. Technical Report UCB/CSD 94/814, University of California, Berkeley, CA, May 1994. Masters Thesis.
- [14] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *Proceedings of the 1994 Winter USENIX*, pages 279–291, San Francisco, CA, 1994.
- [15] Brian Marsh, Fred Douglis, and P. Krishnan. Flash Memory File Caching for Mobile Computers. In *Proceedings of the 27th Hawaii Conference on Systems Sciences*, pages 451–460, Maui, HI, 1994. IEEE.
- [16] Brian Marsh and Bruce Zenel. Power Measurements of Typical Notebook Computers. Technical Report 110-94, Matsushita Information Technology Laboratory, May 1994.
- [17] Marshall Kirk McKusick, Michael J. Karels, and Keith Bostic. A pageable memory based file system. In *USENIX Conference Proceedings*, pages 137–144, Anaheim, CA, Summer 1990. USENIX.
- [18] NEC. *Memory Products Data Book, Volume 1: DRAMS, DRAM Modules, Video RAMS*, 1993.
- [19] NEC. *Memory Products Data Book, Volume 2: SRAMs, ASMs, EEPROMs*, 1993.
- [20] Mendel Rosenblum and John Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, February 1992. Also appears in *Proceedings of the 13th Symposium on Operating Systems Principles*, October 1991.
- [21] Chris Ruemmler and John Wilkes. UNIX disk access patterns. In *Proceedings of the Winter 1993 USENIX Conference*, pages 405–420, San Diego, January 1993.
- [22] SunDisk Corporation. *SunDisk SDP Series OEM Manual*, 1993.
- [23] SunDisk Corporation, 3270 Jay Street, Santa Clara, CA 95054. *Competitive Analysis 80-40-00002 Rev. 1.0*, 1994.
- [24] Transaction Processing Performance Council. *TPC Benchmark A Standard Specification Rev 1.1*.
- [25] Michael Wu and Willy Zwaenepoel. eNVy: a Non-Volatile, main memory storage system. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 86–97, San Jose, CA, October 1994.

8 DIFFERENCES FROM THE PRECEDING VERSION

This chapter differs from the paper [5] that appeared in the *Proceedings of the First Symposium on Operating Systems Design and Implementation* in the following ways:

SRAM for flash memory Here we simulate an SRAM write buffer for flash memory as well as for disks. This greatly reduces write latency in many cases.

SRAM writes to a spun-down disk Although we attempted to buffer writes to a spun-down disk rather than spin it up again, previously there were circumstances when the disk would be spun up upon a write that fit in SRAM. The simulator now buffers writes in those cases. This results in lower energy consumption when writes are small and infrequent or hit a “hot spot.” However, more reads hit a spun-down disk, so reads that miss in the DRAM buffer cache may suffer an increased penalty due to disk spin-up. Furthermore, the effect of eliminating some accesses to disk is especially profound if the disk spins down just before the next operation: a read or write to a Caviar Ultralite CU140 may have to wait up to 3.5s in the worst case while the drive spins down and back up again.

Caviar Ultralite CU140 After the previous version appeared, we were told by a technical support representative at Western Digital that their Caviar Ultralite CU140 does not go into its low-power *sleep* mode unless the entire system is asleep. The least power consumed is therefore 125mW rather than 1.5mW as previously believed. Therefore, the total energy consumption for simulations using the Caviar Ultralite cu140 increased in proportion to the duration of the trace with long periods of inactivity. The HP trace was impacted by this change the most, with its total energy consumption increasing by factors of 3–4.

Write response time Until recently, the response time for an operation was measured from the time the operation started until the time it completed. The start time was

$$\max(\text{start_of_previous_operation} + \text{interarrival_time}, \\ \text{completion_of_pending_operations}).$$

Thus if flash cleaning caused a write to take 1.6s rather than 30ms, the response time for that write would be 1.6s but the next write would be considered to start upon the completion of the write that was delayed. Delays due to background operation were similarly excluded, so if a background operation (such as writing a block from SRAM to disk) completed after the time the next operation could have started, the start time was delayed. Upon reflection, we now consider background operations to have no effect on start time. As a result, and as one would expect, the delay due to flushing a block from SRAM to disk if SRAM fills up with modified data is more pronounced in small SRAM caches than in larger ones.

Table 5 summarizes the effects of these changes.

Change	Effect
SRAM for flash	Flash writes are much faster
SRAM writes to a spun-down disk	Less energy is consumed. Response time varies depending on the likelihood of the disk spinning down between operations, especially if the disk spins down just before the next operation
Caviar Ultralite cu140 specification	The Caviar Ultralite cu140 consumes substantially more energy over a long trace than before.
Write response time treatment	Mean write response increases when SRAM or flash fills and must be flushed between operations

Table 5: Summary of changes from [5].

DISCONNECTED OPERATION IN THE CODA FILE SYSTEM

James J. Kistler and M. Satyanarayanan

*School of Computer Science,
Carnegie Mellon University*

ABSTRACT

Disconnected operation is a mode of operation that enables a client to continue accessing critical data during temporary failures of a shared data repository. An important, though not exclusive, application of disconnected operation is in supporting portable computers. In this paper, we show that disconnected operation is feasible, efficient and usable by describing its design and implementation in the Coda File System. The central idea behind our work is that *caching of data*, now widely used for performance, can also be exploited to improve *availability*.

1 INTRODUCTION

Every serious user of a distributed system has faced situations where critical work has been impeded by a remote failure. His frustration is particularly acute when his workstation is powerful enough to be used standalone, but has

Previously appeared in ACM Transactions on Computer Systems, Vol. 10, No. 1, Feb 1992. Copyright ©1992 by the Association for Computing Machinery, Inc. Reprinted by permission. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept, ACM Inc., fax +1 (212) 869-0481, or (permissions@acm.org)

been configured to be dependent on remote resources. An important instance of such dependence is the use of data from a distributed file system.

Placing data in a distributed file system simplifies collaboration between users, and allows them to delegate the administration of that data. The growing popularity of distributed file systems such as NFS [15] and AFS [18] attests to the compelling nature of these considerations. Unfortunately, the users of these systems have to accept the fact that a remote failure at a critical juncture may seriously inconvenience them.

How can we improve this state of affairs? Ideally, we would like to enjoy the benefits of a shared data repository, but be able to continue critical work when that repository is inaccessible. We call the latter mode of operation *disconnected operation*, because it represents a temporary deviation from normal operation as a client of a shared repository.

In this paper we show that disconnected operation in a file system is indeed feasible, efficient and usable. The central idea behind our work is that *caching of data*, now widely used to improve performance, can also be exploited to enhance *availability*. We have implemented disconnected operation in the Coda File System at Carnegie Mellon University.

Our initial experience with Coda confirms the viability of disconnected operation. We have successfully operated disconnected for periods lasting four to five hours. For a disconnection of this duration, the process of reconnecting and propagating changes typically takes about a minute. A local disk of 100MB has been adequate for us during these periods of disconnection. Trace-driven simulations indicate that a disk of about half that size should be adequate for disconnections lasting a typical workday.

2 DESIGN OVERVIEW

Coda is designed for an environment consisting of a large collection of untrusted Unix¹ clients and a much smaller number of trusted Unix file servers. The design is optimized for the access and sharing patterns typical of academic and research environments. It is specifically not intended for applications that exhibit highly concurrent, fine granularity data access.

¹Unix is a trademark of AT&T.

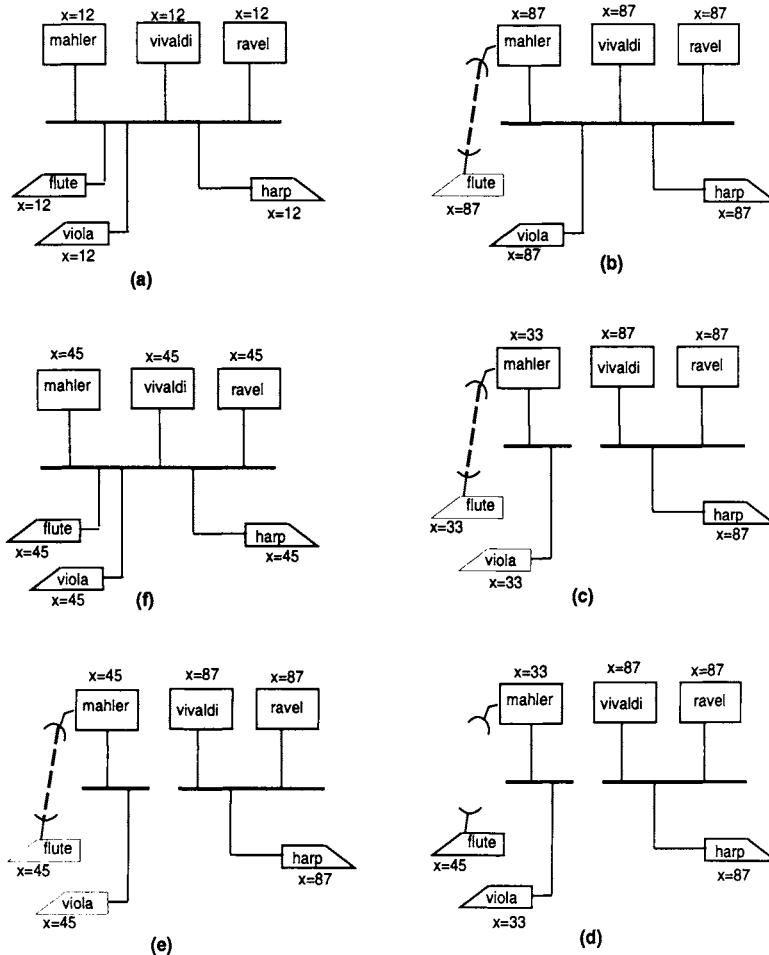
Each Coda client has a local disk and can communicate with the servers over a high bandwidth network. At certain times, a client may be temporarily unable to communicate with some or all of the servers. This may be due to a server or network failure, or due to the detachment of a *portable client* from the network.

Clients view Coda as a single, location-transparent shared Unix file system. The Coda namespace is mapped to individual file servers at the granularity of subtrees called *volumes*. At each client, a *cache manager* (*Venus*) dynamically obtains and caches volume mappings.

Coda uses two distinct, but complementary, mechanisms to achieve high availability. The first mechanism, *server replication*, allows volumes to have read-write replicas at more than one server. The set of replication sites for a volume is its *volume storage group* (*VSG*). The subset of a VSG that is currently accessible is a client's *accessible VSG* (*AVSG*). The performance cost of server replication is kept low by caching on disks at clients and through the use of parallel access protocols. Venus uses a cache coherence protocol based on *callbacks* [9] to guarantee that an open of a file yields its latest copy in the AVSG. This guarantee is provided by servers notifying clients when their cached copies are no longer valid, each notification being referred to as a “callback break.” Modifications in Coda are propagated in parallel to all AVSG sites, and eventually to missing VSG sites.

Disconnected operation, the second high availability mechanism used by Coda, takes effect when the AVSG becomes empty. While disconnected, Venus services file system requests by relying solely on the contents of its cache. Since cache misses cannot be serviced or masked, they appear as failures to application programs and users. When disconnection ends, Venus propagates modifications and reverts to server replication. Figure 1 depicts a typical scenario involving transitions between server replication and disconnected operation.

Earlier Coda papers [17], [18] have described server replication in depth. In contrast, this paper restricts its attention to disconnected operation. We discuss server replication only in those areas where its presence has significantly influenced our design for disconnected operation.



Three servers (mahler, vivaldi, and ravel) have replicas of the volume containing file x . This file is potentially of interest to users at three clients (flute, viola, and harp). Flute is capable of wireless communication (indicated by a dotted line) as well as regular network communication. Proceeding clockwise, the steps above show the value of x seen by each node as the connectivity of the system changes. Note that in step (d), flute is operating disconnected.

Figure 1 How Disconnected Operation Relates to Server Replication

3 DESIGN RATIONALE

At a high level, two factors influenced our strategy for high availability. First, we wanted to use conventional, off-the-shelf hardware throughout our system. Second, we wished to preserve *transparency* by seamlessly integrating the high availability mechanisms of Coda into a normal Unix environment.

At a more detailed level, other considerations influenced our design. These include the need to *scale* gracefully, the advent of *portable workstations*, the very different *resource*, *integrity*, and *security* assumptions made about clients and servers, and the need to strike a balance between *availability* and *consistency*. We examine each of these issues in the following sections.

3.1 Scalability

Successful distributed systems tend to grow in size. Our experience with Coda's ancestor, AFS, had impressed upon us the need to prepare for growth *a priori*, rather than treating it as an afterthought [16]. We brought this experience to bear upon Coda in two ways. First, we adopted certain mechanisms that enhance scalability. Second, we drew upon a set of general principles to guide our design choices.

An example of a mechanism we adopted for scalability is callback-based cache coherence. Another such mechanism, *whole-file caching*, offers the added advantage of a much simpler failure model: a cache miss can only occur on an *open*, never on a *read*, *write*, *seek*, or *close*. This, in turn, substantially simplifies the implementation of disconnected operation. A partial-file caching scheme such as that of AFS-4 [21], Echo [8] or MFS [1] would have complicated our implementation and made disconnected operation less transparent.

A scalability principle that has had considerable influence on our design is the *placing of functionality on clients* rather than servers. Only if integrity or security would have been compromised have we violated this principle. Another scalability principle we have adopted is the *avoidance of system-wide rapid change*. Consequently, we have rejected strategies that require election or agreement by large numbers of nodes. For example, we have avoided algorithms such as that used in Locus [22] that depend on nodes achieving consensus on the current partition state of the network.

3.2 Portable Workstations

Powerful, lightweight and compact laptop computers are commonplace today. It is instructive to observe how a person with data in a shared file system uses such a machine. Typically, he identifies files of interest and downloads them from the shared file system into the local name space for use while isolated. When he returns, he copies modified files back into the shared file system. Such a user is effectively performing manual caching, with write-back upon reconnection!

Early in the design of Coda we realized that disconnected operation could substantially simplify the use of portable clients. Users would not have to use a different name space while isolated, nor would they have to manually propagate changes upon reconnection. Thus portable machines are a champion application for disconnected operation.

The use of portable machines also gave us another insight. The fact that people are able to operate for extended periods in isolation indicates that they are quite good at predicting their future file access needs. This, in turn, suggests that it is reasonable to seek user assistance in augmenting the cache management policy for disconnected operation.

Functionally, *involuntary* disconnections caused by failures are no different from *voluntary* disconnections caused by unplugging portable computers. Hence Coda provides a single mechanism to cope with all disconnections. Of course, there may be qualitative differences: user expectations as well as the extent of user cooperation are likely to be different in the two cases.

3.3 First vs Second Class Replication

If disconnected operation is feasible, why is server replication needed at all? The answer to this question depends critically on the very different assumptions made about clients and servers in Coda.

Clients are like appliances: they can be turned off at will and may be unattended for long periods of time. They have limited disk storage capacity, their software and hardware may be tampered with, and their owners may not be diligent about backing up the local disks. Servers are like public utilities: they have much greater disk capacity, they are physically secure, and they are carefully monitored and administered by professional staff.

It is therefore appropriate to distinguish between *first class replicas* on servers, and *second class replicas* (i.e., cache copies) on clients. First class replicas are of higher quality: they are more persistent, widely known, secure, available, complete and accurate. Second class replicas, in contrast, are inferior along all these dimensions. Only by periodic revalidation with respect to a first class replica can a second class replica be useful.

The function of a cache coherence protocol is to combine the performance and scalability advantages of a second class replica with the quality of a first class replica. When disconnected, the quality of the second class replica may be degraded because the first class replica upon which it is contingent is inaccessible. The longer the duration of disconnection, the greater the potential for degradation. Whereas server replication preserves the quality of data in the face of failures, disconnected operation forsakes quality for availability. Hence server replication is important because it reduces the frequency and duration of disconnected operation, which is properly viewed as a measure of last resort.

Server replication is expensive because it requires additional hardware. Disconnected operation, in contrast, costs little. Whether to use server replication or not is thus a tradeoff between quality and cost. Coda does permit a volume to have a sole server replica. Therefore, an installation can rely exclusively on disconnected operation if it so chooses.

3.4 Optimistic vs Pessimistic Replica Control

By definition, a network partition exists between a disconnected second class replica and all its first class associates. The choice between two families of replica control strategies, *pessimistic* and *optimistic* [5], is therefore central to the design of disconnected operation. A pessimistic strategy avoids conflicting operations by disallowing all partitioned writes or by restricting reads and writes to a single partition. An optimistic strategy provides much higher availability by permitting reads and writes everywhere, and deals with the attendant danger of conflicts by detecting and resolving them after their occurrence.

A pessimistic approach towards disconnected operation would require a client to acquire shared or exclusive control of a cached object prior to disconnection, and to retain such control until reconnection. Possession of exclusive control by a disconnected client would preclude reading or writing at all other replicas. Possession of shared control would allow reading at other replicas, but writes would still be forbidden everywhere.

Acquiring control prior to voluntary disconnection is relatively simple. It is more difficult when disconnection is involuntary, because the system may have to arbitrate among multiple requestors. Unfortunately, the information needed to make a wise decision is not readily available. For example, the system cannot predict which requestors will actually use the object, when they will release control, or what the relative costs of denying them access would be.

Retaining control until reconnection is acceptable in the case of brief disconnections. But it is unacceptable in the case of extended disconnections. A disconnected client with shared control of an object would force the rest of the system to defer all updates until it reconnected. With exclusive control, it would even prevent other users from making a copy of the object. Coercing the client to reconnect may not be feasible, since its whereabouts may not be known. Thus, an entire user community could be at the mercy of a single errant client for an unbounded amount of time.

Placing a time bound on exclusive or shared control, as done in the case of leases [7], avoids this problem but introduces others. Once a lease expires, a disconnected client loses the ability to access a cached object, even if no else in the system is interested in it. This, in turn, defeats the purpose of disconnected operation which is to provide high availability. Worse, updates already made while disconnected have to be discarded.

An optimistic approach has its own disadvantages. An update made at one disconnected client may conflict with an update at another disconnected or connected client. For optimistic replication to be viable, the system has to be more sophisticated. There needs to be machinery in the system for detecting conflicts, for automating resolution when possible, and for confining damage and preserving evidence for manual repair. Having to repair conflicts manually violates transparency, is an annoyance to users, and reduces the usability of the system.

We chose optimistic replication because we felt that its strengths and weaknesses better matched our design goals. The dominant influence on our choice was the low degree of write-sharing typical of Unix. This implied that an optimistic strategy was likely to lead to relatively few conflicts. An optimistic strategy was also consistent with our overall goal of providing the highest possible availability of data.

In principle, we could have chosen a pessimistic strategy for server replication even after choosing an optimistic strategy for disconnected operation. But that would have reduced transparency, because a user would have faced the anomaly

of being able to update data when disconnected, but being unable to do so when connected to a subset of the servers. Further, many of the previous arguments in favor of an optimistic strategy also apply to server replication.

Using an optimistic strategy throughout presents a uniform model of the system from the user's perspective. At any time, he is able to read the latest data in his *accessible universe* and his updates are immediately visible to everyone else in that universe. His accessible universe is usually the entire set of servers and clients. When failures occur, his accessible universe shrinks to the set of servers he can contact, and the set of clients that they, in turn, can contact. In the limit, when he is operating disconnected, his accessible universe consists of just his machine. Upon reconnection, his updates become visible throughout his now-enlarged accessible universe.

4 DETAILED DESIGN AND IMPLEMENTATION

In describing our implementation of disconnected operation, we focus on the client since this is where much of the complexity lies. Section 4.1 describes the physical structure of a client, Section 4.2 introduces the major states of Venus, and Sections 4.3 to 4.5 discuss these states in detail. A description of the server support needed for disconnected operation is contained in Section 4.5.

4.1 Client Structure

Because of the complexity of Venus, we made it a user level process rather than part of the kernel. The latter approach may have yielded better performance, but would have been less portable and considerably more difficult to debug. Figure 2 illustrates the high-level structure of a Coda client.

Venus intercepts Unix file system calls via the widely-used Sun Vnode interface [10]. Since this interface imposes a heavy performance overhead on user-level cache managers, we use a tiny in-kernel *MiniCache* to filter out many kernel-Venus interactions. The MiniCache contains no support for remote access, disconnected operation or server replication; these functions are handled entirely by Venus.

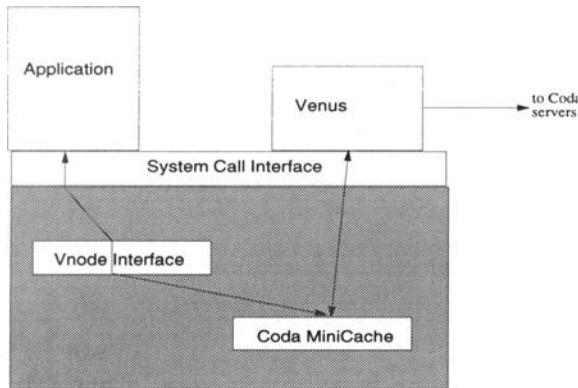
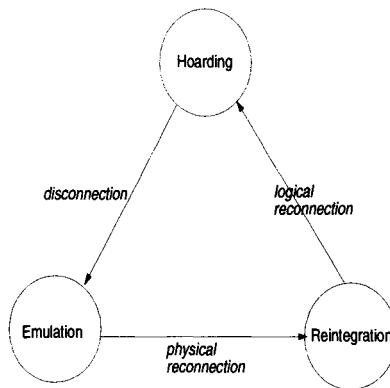


Figure 2 Structure of a Coda Client

A system call on a Coda object is forwarded by the Vnode interface to the MiniCache. If possible, the call is serviced by the MiniCache and control is returned to the application. Otherwise, the MiniCache contacts Venus to service the call. This, in turn, may involve contacting Coda servers. Control returns from Venus via the MiniCache to the application program, updating MiniCache state as a side effect. MiniCache state changes may also be initiated by Venus on events such as callback breaks from Coda servers. Measurements from our implementation confirm that the MiniCache is critical for good performance [20].

4.2 Venus States

Logically, Venus operates in one of three states: *hoarding*, *emulation*, and *reintegration*. Figure 3 depicts these states and the transitions between them. Venus is normally in the hoarding state, relying on server replication but always on the alert for possible disconnection. Upon disconnection, it enters the emulation state and remains there for the duration of disconnection. Upon reconnection, Venus enters the reintegration state, resynchronizes its cache with its AVSG, and then reverts to the hoarding state. Since all volumes may not be replicated across the same set of servers, Venus can be in different states with respect to different volumes, depending on failure conditions in the system.



When disconnected, Venus is in the emulation state. It transits to reintegration upon successful reconnection to an AVSG member, and thence to hoarding, where it resumes connected operation.

Figure 3 Venus States and Transitions

4.3 Hoarding

The hoarding state is so named because a key responsibility of Venus in this state is to hoard useful data in anticipation of disconnection. However, this is not its only responsibility. Rather, Venus must manage its cache in a manner that balances the needs of connected and disconnected operation. For instance, a user may have indicated that a certain set of files is critical but may currently be using other files. To provide good performance, Venus must cache the latter files. But to be prepared for disconnection, it must also cache the former set of files.

Many factors complicate the implementation of hoarding:

- File reference behavior, especially in the distant future, cannot be predicted with certainty.
- Disconnections and reconnections are often unpredictable.
- The true cost of a cache miss while disconnected is highly variable and hard to quantify.
- Activity at other clients must be accounted for, so that the latest version of an object is in the cache at disconnection.

```

# Personal files                                # X11 files          # Venus source files
a /coda/usr/jjk d+                            # (from X11 maintainer)   a /coda/project/coda/src/venus 100:c+
a /coda/usr/jjk 100:d+                         a /usr/X11/bin/X        a /coda/project/coda/include 100:c+
a /coda/usr/jjk/papers/sosp 1000:d+            a /usr/X11/bin/Xvga     a /coda/project/coda/lib c+
a /usr/X11/bin/mwmw                           a /usr/X11/bin/startx   .
a /usr/X11/bin/xclock                          a /usr/X11/bin/xinit    .
a /usr/X11/bin/xinit                           a /usr/X11/bin/xterm    .
a /usr/X11/include/X11/bitmaps c+             a /usr/X11/lib/app-defaults d+
a /usr/X11/lib/fontr/misc c+                  a /usr/X11/lib/fonts/misc c+
a /usr/X11/lib/system.mwmrc                   a /usr/X11/lib/system.mwmrc

# System files
a /usr/bin/100:d+
a /usr/etc 100:d+
a /usr/include 100:d+
a /usr/lib 100:d+
a /usr/local/gnu d+
a /usr/local/rca d+
a /usr/ucb d+

```

(a) (b) (c)

These are typical hoard profiles provided by a Coda user, an application maintainer, and a group of project developers. Each profile is interpreted separately by the HDB front-end program. The 'a' at the beginning of a line indicates an add-entry command. Other commands are delete an entry, clear all entries, and list entries. The modifiers following some pathnames specify non-default priorities (the default is 10) and/or meta-expansion for the entry. Note that the pathnames beginning with '/usr' are actually symbolic links into '/coda'.

Figure 4 Sample Hoard Profiles

- Since cache space is finite, the availability of less critical objects may have to be sacrificed in favor of more critical objects.

To address these concerns, we manage the cache using a *prioritized algorithm*, and periodically reevaluate which objects merit retention in the cache via a process known as *hoard walking*.

Prioritized Cache Management

Venus combines implicit and explicit sources of information in its priority-based cache management algorithm. The implicit information consists of recent reference history, as in traditional caching algorithms. Explicit information takes the form of a per-workstation *hoard database* (*HDB*), whose entries are pathnames identifying objects of interest to the user at that workstation.

A simple front-end program allows a user to update the HDB using command scripts called *hoard profiles*, such as those shown in Figure 4. Since hoard profiles are just files, it is simple for an application maintainer to provide a common profile for his users, or for users collaborating on a project to maintain a common profile. A user can customize his HDB by specifying different combinations of profiles or by executing front-end commands interactively. To

facilitate construction of hoard profiles, Venus can record all file references observed between a pair of start and stop events indicated by a user.

To reduce the verbosity of hoard profiles and the effort needed to maintain them, Venus supports *meta-expansion* of HDB entries. As shown in Figure 4, if the letter 'c' (or 'd') follows a pathname, the command also applies to immediate children (or all descendants). A '+' following the 'c' or 'd' indicates that the command applies to all future as well as present children or descendants. A hoard entry may optionally indicate a *hoard priority*, with higher priorities indicating more critical objects.

The current priority of a cached object is a function of its hoard priority as well as a metric representing recent usage. The latter is updated continuously in response to new references, and serves to age the priority of objects no longer in the working set. Objects of the lowest priority are chosen as victims when cache space has to be reclaimed.

To resolve the pathname of a cached object while disconnected, it is imperative that all the ancestors of the object also be cached. Venus must therefore ensure that a cached directory is not purged before any of its descendants. This *hierarchical cache management* is not needed in traditional file caching schemes because cache misses during name translation can be serviced, albeit at a performance cost. Venus performs hierarchical cache management by assigning infinite priority to directories with cached children. This automatically forces replacement to occur bottom-up.

Hoard Walking

We say that a cache is in *equilibrium*, signifying that it meets user expectations about availability, when no uncached object has a higher priority than a cached object. Equilibrium may be disturbed as a result of normal activity. For example, suppose an object, *A*, is brought into the cache on demand, replacing an object, *B*. Further suppose that *B* is mentioned in the HDB, but *A* is not. Some time after activity on *A* ceases, its priority will decay below the hoard priority of *B*. The cache is no longer in equilibrium, since the cached object *A* has lower priority than the uncached object *B*.

Venus periodically restores equilibrium by performing an operation known as a *hoard walk*. A hoard walk occurs every 10 minutes in our current implementation, but one may be explicitly requested by a user prior to voluntary disconnection. The walk occurs in two phases. First, the *name bindings* of

HDB entries are reevaluated to reflect update activity by other Coda clients. For example, new children may have been created in a directory whose pathname is specified with the '+' option in the HDB. Second, the priorities of all entries in the cache and HDB are reevaluated, and objects fetched or evicted as needed to restore equilibrium.

Hoard walks also address a problem arising from callback breaks. In traditional callback-based caching, data is refetched only on demand after a callback break. But in Coda, such a strategy may result in a critical object being unavailable should a disconnection occur before the next reference to it. Refetching immediately upon callback break avoids this problem, but ignores a key characteristic of Unix environments: once an object is modified, it is likely to be modified many more times by the same user within a short interval [14], [6]. An immediate refetch policy would increase client-server traffic considerably, thereby reducing scalability.

Our strategy is a compromise that balances availability, consistency, and scalability. For files and symbolic links, Venus purges the object on callback break, and refetches it on demand or during the next hoard walk, whichever occurs earlier. If a disconnection were to occur before refetching, the object would be unavailable. For directories, Venus does not purge on callback break, but marks the cache entry suspicious. A stale cache entry is thus available should a disconnection occur before the next hoard walk or reference. The acceptability of stale directory data follows from its particular callback semantics. A callback break on a directory typically means that an entry has been added to or deleted from the directory. It is often the case that other directory entries and the objects they name are unchanged. Therefore, saving the stale copy and using it in the event of untimely disconnection causes consistency to suffer only a little, but increases availability considerably.

4.4 Emulation

In the emulation state, Venus performs many actions normally handled by servers. For example, Venus now assumes full responsibility for access and semantic checks. It is also responsible for generating temporary *file identifiers* (fids) for new objects, pending the assignment of permanent fids at reintegration. But although Venus is functioning as a *pseudo-server*, updates accepted by it have to be revalidated with respect to integrity and protection by real servers. This follows from the Coda policy of trusting only servers, not clients.

To minimize unpleasant delayed surprises for a disconnected user, it behooves Venus to be as faithful as possible in its emulation.

Cache management during emulation is done with the same priority algorithm used during hoarding. Mutating operations directly update the cache entries of the objects involved. Cache entries of deleted objects are freed immediately, but those of other modified objects assume infinite priority so that they are not purged before reintegration. On a cache miss, the default behavior of Venus is to return an error code. A user may optionally request Venus to block his processes until cache misses can be serviced.

Logging

During emulation, Venus records sufficient information to replay update activity when it reintegrates. It maintains this information in a per-volume log of mutating operations called a *replay log*. Each log entry contains a copy of the corresponding system call arguments as well as the version state of all objects referenced by the call.

Venus uses a number of optimizations to reduce the length of the replay log, resulting in a log size that is typically a few percent of cache size. A small log conserves disk space, a critical resource during periods of disconnection. It also improves reintegration performance by reducing latency and server load.

One important optimization to reduce log length pertains to `write` operations on files. Since Coda uses whole-file caching, the `close` after an `open` of a file for modification installs a completely new copy of the file. Rather than logging the `open`, `close`, and intervening `write` operations individually, Venus logs a single `store` record during the handling of a `close`.

Another optimization consists of Venus discarding a previous `store` record for a file when a new one is appended to the log. This follows from the fact that a `store` renders all previous versions of a file superfluous. The `store` record does not contain a copy of the file's contents, but merely points to the copy in the cache.

We are currently implementing two further optimizations to reduce the length of the replay log. The first generalizes the optimization described in the previous paragraph such that any operation which *overwrites* the effect of earlier operations may cancel the corresponding log records. An example would be the cancelling of a `store` by a subsequent `unlink` or `truncate`. The second opti-

mization exploits knowledge of *inverse* operations to cancel both the inverting and inverted log records. For example, a `rmdir` may cancel its own log record as well as that of the corresponding `mkdir`.

Persistence

A disconnected user must be able to restart his machine after a shutdown and continue where he left off. In case of a crash, the amount of data lost should be no greater than if the same failure occurred during connected operation. To provide these guarantees, Venus must keep its cache and related data structures in non-volatile storage.

Meta-data, consisting of cached directory and symbolic link contents, status blocks for cached objects of all types, replay logs, and the HDB, is mapped into Venus' address space as *recoverable virtual memory* (RVM). Transactional access to this memory is supported by the RVM library [12] linked into Venus. The actual contents of cached files are not in RVM, but are stored as local Unix files.

The use of transactions to manipulate meta-data simplifies Venus' job enormously. To maintain its invariants Venus need only ensure that each transaction takes meta-data from one consistent state to another. It need not be concerned with crash recovery, since RVM handles this transparently. If we had chosen the obvious alternative of placing meta-data in local Unix files, we would have had to follow a strict discipline of carefully timed synchronous writes and an ad-hoc recovery algorithm.

RVM supports local, non-nested transactions and allows independent control over the basic transactional properties of atomicity, permanence, and serializability. An application can reduce commit latency by labelling the commit as *no-flush*, thereby avoiding a synchronous write to disk. To ensure persistence of no-flush transactions, the application must explicitly flush RVM's write-ahead log from time to time. When used in this manner, RVM provides *bounded persistence*, where the bound is the period between log flushes.

Venus exploits the capabilities of RVM to provide good performance at a constant level of persistence. When hoarding, Venus initiates log flushes infrequently, since a copy of the data is available on servers. Since servers are not accessible when emulating, Venus is more conservative and flushes the log more frequently. This lowers performance, but keeps the amount of data lost by a client crash within acceptable limits.

Resource Exhaustion

It is possible for Venus to exhaust its non-volatile storage during emulation. The two significant instances of this are the file cache becoming filled with modified files, and the RVM space allocated to replay logs becoming full.

Our current implementation is not very graceful in handling these situations. When the file cache is full, space can be freed by truncating or deleting modified files. When log space is full, no further mutations are allowed until reintegration has been performed. Of course, non-mutating operations are always allowed.

We plan to explore at least three alternatives to free up disk space while emulating. One possibility is to compress file cache and RVM contents. Compression trades off computation time for space, and recent work [2] has shown it to be a promising tool for cache management. A second possibility is to allow users to selectively back out updates made while disconnected. A third approach is to allow portions of the file cache and RVM to be written out to removable media such as floppy disks.

4.5 Reintegration

Reintegration is a transitory state through which Venus passes in changing roles from pseudo-server to cache manager. In this state, Venus propagates changes made during emulation, and updates its cache to reflect current server state. Reintegration is performed a volume at a time, with all update activity in the volume suspended until completion.

Replay Algorithm

The propagation of changes from client to AVSG is accomplished in two steps. In the first step, Venus obtains permanent fids for new objects and uses them to replace temporary fids in the replay log. This step is avoided in many cases, since Venus obtains a small supply of permanent fids in advance of need, while in the hoarding state. In the second step, the replay log is shipped in parallel to the AVSG, and executed independently at each member. Each server performs the replay within a single transaction, which is aborted if any error is detected.

The replay algorithm consists of four phases. In phase one the log is parsed, a transaction is begun, and all objects referenced in the log are locked. In phase two, each operation in the log is validated and then executed. The validation

consists of conflict detection as well as integrity, protection, and disk space checks. Except in the case of `store` operations, execution during replay is identical to execution in connected mode. For a `store`, an empty *shadow file* is created and meta-data is updated to reference it, but the data transfer is deferred. Phase three consists exclusively of performing these data transfers, a process known as *back-fetching*. The final phase commits the transaction and releases all locks.

If reintegrations succeeds, Venus frees the replay log and resets the priority of cached objects referenced by the log. If reintegration fails, Venus writes out the replay log to a local *replay file* in a superset of the Unix `tar` format. The log and all corresponding cache entries are then purged, so that subsequent references will cause refetch of the current contents at the AVSG. A tool is provided which allows the user to inspect the contents of a replay file, compare it to the state at the AVSG, and replay it selectively or in its entirety.

Reintegration at finer granularity than a volume would reduce the latency perceived by clients, improve concurrency and load balancing at servers, and reduce user effort during manual replay. To this end, we are revising our implementation to re-integrate at the granularity of subsequences of dependent operations within a volume. Dependent subsequences can be identified using the *precedence graph* approach of Davidson [4]. In the revised implementation Venus will maintain precedence graphs during emulation, and pass them to servers along with the replay log.

Conflict Handling

Our use of optimistic replica control means that the disconnected operations of one client may conflict with activity at servers or other disconnected clients. The only class of conflicts we are concerned with are *write/write* conflicts. *Read/write* conflicts are not relevant to the Unix file system model, since it has no notion of atomicity beyond the boundary of a single system call.

The check for conflicts relies on the fact that each replica of an object is tagged with a *storeid* that uniquely identifies the last update to it. During phase two of replay, a server compares the storeid of every object mentioned in a log entry with the storeid of its own replica of the object. If the comparison indicates equality for all objects, the operation is performed and the mutated objects are tagged with a new storeid specified in the log entry.

If a storeid comparison fails, the action taken depends on the operation being validated. In the case of a *store* of a file, the entire reintegration is aborted. But for directories, a conflict is declared only if a newly created name collides with an existing name, if an object updated at the client or the server has been deleted by the other, or if directory attributes have been modified at the server and the client. This strategy of *resolving* partitioned directory updates is consistent with our strategy in server replication [11], and was originally suggested by Locus [22].

Our original design for disconnected operation called for preservation of replay files at servers rather than clients. This approach would also allow damage to be confined by marking conflicting replicas inconsistent and forcing manual repair, as is currently done in the case of server replication. We are awaiting more usage experience to determine whether this is indeed the correct approach for disconnected operation.

5 STATUS AND EVALUATION

Today, Coda runs on IBM RTs, Decstation 3100s and 5000s, and 386-based laptops such as the Toshiba 5200. A small user community has been using Coda on a daily basis as its primary data repository since April 1990. All development work on Coda is done in Coda itself. As of July 1991 there were nearly 350MB of triply-replicated data in Coda, with plans to expand to 2GB in the next few months.

A version of disconnected operation with minimal functionality was demonstrated in October 1990. A more complete version was functional in January 1991, and is now in regular use. We have successfully operated disconnected for periods lasting four to five hours. Our experience with the system has been quite positive, and we are confident that the refinements under development will result in an even more usable system.

In the following sections we provide qualitative and quantitative answers to three important questions pertaining to disconnected operation. These are:

- 1. How long does reintegration take?
- 2. How large a local disk does one need?
- 3. How likely are conflicts?

5.1 Duration of Reintegration

In our experience, typical disconnected sessions of editing and program development lasting a few hours require about a minute for reintegration. To characterize reintegration speed more precisely, we measured the reintegration times after disconnected execution of two well-defined tasks. The first task is the Andrew benchmark [9], now widely used as a basis for comparing file system performance. The second task is the compiling and linking of the current version of Venus. Table 1 presents the reintegration times for these tasks.

The time for reintegration consists of three components: the time to allocate permanent fids, the time for the replay at the servers, and the time for the second phase of the update protocol used for server replication. The first component will be zero for many disconnections, due to the preallocation of fids during hoarding. We expect the time for the second component to fall, considerably in many cases, as we incorporate the last of the replay log optimizations described in Section 4.4. The third component can be avoided only if server replication is not used.

One can make some interesting secondary observations from Table 1. First, the total time for reintegration is roughly the same for the two tasks even though the Andrew benchmark has a much smaller elapsed time. This is because the Andrew benchmark uses the file system more intensively. Second, reintegration for the Venus make takes longer, even though the number of entries in the replay log is smaller. This is because much more file data is back-fetched in the third phase of the replay. Finally, neither task involves any think time. As a result, their reintegration times are comparable to that after a much longer, but more typical, disconnected session in our environment.

5.2 Cache Size

A local disk capacity of 100MB on our clients has proved adequate for our initial sessions of disconnected operation. To obtain a better understanding of the cache size requirements for disconnected operation, we analyzed file reference traces from our environment. The traces were obtained by instrumenting workstations to record information on every file system operation, regardless of whether the file was in Coda, AFS, or the local file system.

Our analysis is based on simulations driven by these traces. Writing and validating a simulator that precisely models the complex caching behavior of Venus

	Elapsed Time (seconds)	Reintegration Time (seconds)				Size of Replay Log		Data Back-Fetched (Bytes)
		Total	AllocFid	Replay	COP2	Records	Bytes	
Andrew Benchmark	288 (3)	43 (2)	4 (2)	29 (1)	10 (1)	223	65,010	1,141,315
Venus Make	3,271 (28)	52 (4)	1 (0)	40 (1)	10 (3)	193	65,919	2,990,120

This data was obtained with a Toshiba T5200/100 client (12MB memory, 100MB disk) reintegrating over an Ethernet with an IBM RT-APC server (12MB memory, 400MB disk). The values shown above are the means of three trials. Figures in parentheses are standard deviations.

Table 1 Time for Reintegration

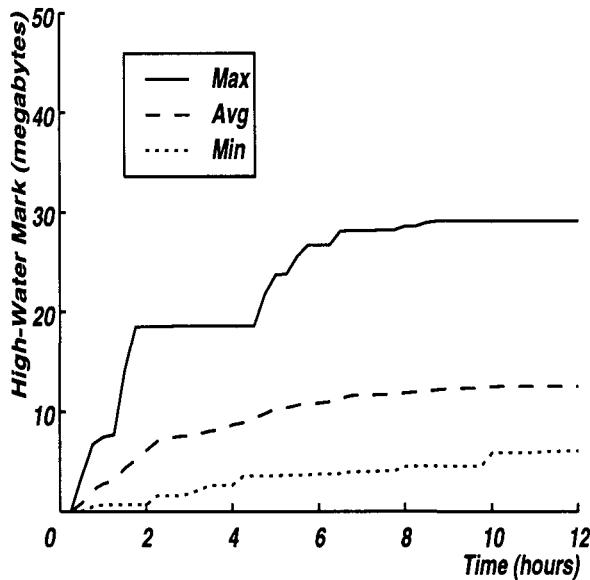
would be quite difficult. To avoid this difficulty, we have modified Venus to act as its own simulator. When running as a simulator, Venus is driven by traces rather than requests from the kernel. Code to communicate with the servers, as well code to perform physical I/O on the local file system are stubbed out during simulation.

Figure 5 shows the *high-water mark* of cache usage as a function of time. The actual disk size needed for disconnected operation has to be larger, since both the explicit and implicit sources of hoarding information are imperfect. From our data it appears that a disk of 50-60MB should be adequate for operating disconnected for a typical workday. Of course, user activity that is drastically different from what was recorded in our traces could produce significantly different results.

We plan to extend our work on trace-driven simulations in three ways. First, we will investigate cache size requirements for much longer periods of disconnection. Second, we will be sampling a broader range of user activity by obtaining traces from many more machines in our environment. Third, we will evaluate the effect of hoarding by simulating traces together with hoard profiles that have been specified *ex ante* by users.

5.3 Likelihood of Conflicts

In our use of optimistic server replication in Coda for nearly a year, we have seen virtually no conflicts due to multiple users updating an object in different



This graph is based on a total of 10 traces from 5 active Coda workstations. The curve labelled "Avg" corresponds to the values obtained by averaging the high-water marks of all workstations. The curves labelled "Max" and "Min" plot the highest and lowest values of the high-water marks across all workstations. Note that the high-water mark does not include space needed for paging, the HDB or replay logs.

Figure 5 High-Water Mark of Cache Usage

network partitions. While gratifying to us, this observation is subject to at least three criticisms. First, it is possible that our users are being cautious, knowing that they are dealing with an experimental system. Second, perhaps conflicts will become a problem only as the Coda user community grows larger. Third, perhaps extended voluntary disconnections will lead to many more conflicts.

To obtain data on the likelihood of conflicts at larger scale, we instrumented the AFS servers in our environment. These servers are used by over 400 computer science faculty, staff and graduate students for research, program development, and education. Their usage profile includes a significant amount of collaborative activity. Since Coda is descended from AFS and makes the same kind of usage assumptions, we can use this data to estimate how frequent conflicts would be if Coda were to replace AFS in our environment.

Type of Volume	Number of Volumes	Type of Object	Total Mutations	Same User	Different Users					
					Total	< 1 min	< 10 min	< 1 hr	< 1 day	< 1 wk
User	529	Files Directories	3,287,135 4,132,066	99.87% 99.80%	0.13% 0.20%	0.04% 0.04%	0.05% 0.07%	0.06% 0.10%	0.09% 0.15%	0.09% 0.16%
Project	108	Files Directories	4,437,311 5,391,224	99.66% 99.63%	0.34% 0.37%	0.17% 0.00%	0.25% 0.01%	0.26% 0.03%	0.28% 0.09%	0.30% 0.15%
System	398	Files Directories	5,526,700 4,338,507	99.17% 99.54%	0.83% 0.46%	0.06% 0.02%	0.18% 0.05%	0.42% 0.08%	0.72% 0.27%	0.78% 0.34%

This data was obtained between June 1990 and May 1991 from the AFS servers in the cs.cmu.edu cell. The servers stored a total of about 12GB of data. The column entitled "Same User" gives the percentage of mutations in which the user performing the mutation was the same as the one performing the immediately preceding mutation on the same file or directory. The remaining mutations contribute to the column entitled "Different User".

Table 2 Sequential Write-Sharing in AFS

Every time a user modifies an AFS file or directory, we compare his identity with that of the user who made the previous mutation. We also note the time interval between mutations. For a file, only the close after an open for update is counted as a mutation; individual write operations are not counted. For directories, all operations that modify a directory are counted as mutations.

Table 2 presents our observations over a period of twelve months. The data is classified by volume type: *user* volumes containing private user data, *project* volumes used for collaborative work, and *system* volumes containing program binaries, libraries, header files and other similar data. On average, a project volume has about 2600 files and 280 directories, and a system volume has about 1600 files and 130 directories. User volumes tend to be smaller, averaging about 200 files and 18 directories, because users often place much of their data in their project volumes.

Table 2 shows that over 99% of all modifications were by the previous writer, and that the chances of two different users modifying the same object less than a day apart is at most 0.75%. We had expected to see the highest degree of write-sharing on project files or directories, and were surprised to see that it actually occurs on system files. We conjecture that a significant fraction of this sharing arises from modifications to system files by operators, who change shift periodically. If system files are excluded, the absence of write-sharing is even

more striking: more than 99.5% of all mutations are by the previous writer, and the chances of two different users modifying the same object within a week are less than 0.4%! This data is highly encouraging from the point of view of optimistic replication. It suggests that conflicts would not be a serious problem if AFS were replaced by Coda in our environment.

6 RELATED WORK

Coda is unique in that it exploits caching for both performance and high availability while preserving a high degree of transparency. We are aware of no other system, published or unpublished, that duplicates this key aspect of Coda.

By providing tools to link local and remote name spaces, the Cedar file system [19] provided rudimentary support for disconnected operation. But since this was not its primary goal, Cedar did not provide support for hoarding, transparent reintegration or conflict detection. Files were versioned and immutable, and a Cedar cache manager could substitute a cached version of a file on reference to an unqualified remote file whose server was inaccessible. However, the implementors of Cedar observe that this capability was not often exploited since remote files were normally referenced by specific version number.

Birrell and Schroeder pointed out the possibility of “stashing” data for availability in an early discussion of the Echo file system [13]. However, a more recent description of Echo [8] indicates that it uses stashing only for the highest levels of the naming hierarchy.

The FACE file system [3] uses stashing but does not integrate it with caching. The lack of integration has at least three negative consequences. First, it reduces transparency because users and applications deal with two different name spaces, with different consistency properties. Second, utilization of local disk space is likely to be much worse. Third, recent usage information from cache management is not available to manage the stash. The available literature on FACE does not report on how much the lack of integration detracted from the usability of the system.

An application-specific form of disconnected operation was implemented in the PCMAIL system at MIT [Lambert88]. PCMAIL allowed clients to disconnect, manipulate existing mail messages and generate new ones, and re-synchronize with a central repository at reconnection. Besides relying heavily on the se-

mantics of mail, PCMAIL was less transparent than Coda since it required manual re-synchronization as well as pre-registration of clients with servers.

The use of optimistic replication in distributed file systems was pioneered by Locus [22]. Since Locus used a peer-to-peer model rather than a client-server model, availability was achieved solely through server replication. There was no notion of caching, and hence of disconnected operation.

Coda has benefited in a general sense from the large body of work on transparency and performance in distributed file systems. In particular, Coda owes much to AFS [18], from which it inherits its model of trust and integrity, as well as its mechanisms and design philosophy for scalability.

7 FUTURE WORK

Disconnected operation in Coda is a facility under active development. In earlier sections of this paper we described work in progress in the areas of log optimization, granularity of reintegration, and evaluation of hoarding. Much additional work is also being done at lower levels of the system. In this section we consider two ways in which the scope of our work may be broadened.

An excellent opportunity exists in Coda for adding *transactional support to Unix*. Explicit transactions become more desirable as systems scale to hundreds or thousands of nodes, and the informal concurrency control of Unix becomes less effective. Many of the mechanisms supporting disconnected operation, such as operation logging, precedence graph maintenance, and conflict checking would transfer directly to a transactional system using optimistic concurrency control. Although transactional file systems are not a new idea, no such system with the scalability, availability, and performance properties of Coda has been proposed or built.

A different opportunity exists in extending Coda to support *weakly-connected operation*, in environments where connectivity is intermittent or of low bandwidth. Such conditions are found in networks that rely on voice-grade lines, or that use wireless technologies such as packet radio. The ability to mask failures, as provided by disconnected operation, is of value even with weak connectivity. But techniques which exploit and adapt to the communication opportunities at hand are also needed. Such techniques may include more aggressive write-back

policies, compressed network transmission, partial file transfer, and caching at intermediate levels.

8 CONCLUSIONS

Disconnected operation is a tantalizingly simple idea. All one has to do is to pre-load one's cache with critical data, continue normal operation until disconnection, log all changes made while disconnected, and replay them upon reconnection.

Implementing disconnected operation is not so simple. It involves major modifications and careful attention to detail in many aspects of cache management. While hoarding, a surprisingly large volume and variety of interrelated state has to be maintained. When emulating, the persistence and integrity of client data structures become critical. During reintegration, there are dynamic choices to be made about the granularity of reintegration.

Only in hindsight do we realize the extent to which implementations of traditional caching schemes have been simplified by the guaranteed presence of a lifeline to a first-class replica. Purging and refetching on demand, a strategy often used to handle pathological situations in those implementations, is not viable when supporting disconnected operation. However, the obstacles to realizing disconnected operation are not insurmountable. Rather, the central message of this paper is that disconnected operation is indeed feasible, efficient and usable.

One way to view our work is to regard it as an extension of the idea of *write-back caching*. Whereas write-back caching has hitherto been used for performance, we have shown that it can be extended to mask temporary failures too. A broader view is that disconnected operation allows graceful transitions between states of *autonomy* and *interdependence* in a distributed system. Under favorable conditions, our approach provides all the benefits of remote data access; under unfavorable conditions, it provides continued access to critical data. We are certain that disconnected operation will become increasingly important as distributed systems grow in scale, diversity and vulnerability.

Acknowledgements

We wish to thank Lily Mummert for her invaluable assistance in collecting and postprocessing the file reference traces used in Section 5.2, and Dimitris Varotsis, who helped instrument the AFS servers which yielded the measurements of Section 5.3. We also wish to express our appreciation to past and present contributors to the Coda project, especially Puneet Kumar, Hank Mashburn, Maria Okasaki, and David Steere.

This work was supported by the Defense Advanced Research Projects Agency (Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio, 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597), National Science Foundation (PYI Award and Grant No. ECD 8907068), IBM Corporation (Faculty Development Award, Graduate Fellowship, and Research Initiation Grant), Digital Equipment Corporation (External Research Project Grant), and Bellcore (Information Networking Research Grant).

REFERENCES

- [1] Burrows, M., "Efficient Data Sharing", *Ph.D. Thesis, University of Cambridge, Computer Laboratory*, December, 1988.
- [2] Cate, V., Gross, T., "Combining the Concepts of Compression and Caching for a Two-Level File System", *Proceedings of the 4th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*, April 1991.
- [3] Cova, L.L., "Resource Management in Federated Computing Environments", *Ph.D. Thesis, Department of Computer Science, Princeton University*, October 1990.
- [4] Davidson, S., "Optimism and Consistency in Partitioned Distributed Database Systems", *ACM Transactions on Database Systems*, Vol. 3, No. 9, September 1984.
- [5] Davidson, S.B., Garcia-Molina, H., Skeen, D., "Consistency in Partitioned Networks", *ACM Computing Surveys*, Vol 17, No 3, September, 1985.
- [6] Floyd, R.A., "Transparency in Distributed File Systems", *Technical Report TR 272, Department of Computer Science, University of Rochester*, 1989.

- [7] Gray, C.G., Cheriton, D.R., "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency", *Proceedings of the 12th ACM Symposium on Operating System Principles*, December 1989.
- [8] Hisgen, A., Birrell, A., Mann, T., Schroeder, M., Swart, G., "Availability and Consistency Tradeoffs in the Echo Distributed File System", *Proceedings of the Second Workshop on Workstation Operating Systems*, September, 1989.
- [9] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., West, M.J., "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988.
- [10] Kleiman, S.R., "Vnodes: An Architecture for Multiple File System Types in Sun UNIX", *Summer Usenix Conference Proceedings*, 1986.
- [11] Kumar, P., Satyanarayanan, M., "Log-Based Directory Resolution in the Coda File System", *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, San Diego, CA, January 1993.
- [12] Mashburn, H., Satyanarayanan, M., "RVM: Recoverable Virtual Memory User Manual", *School of Computer Science, Carnegie Mellon University*, 1991.
- [13] Needham, R.M., Herbert, A.J., "Report on the Third European SIGOPS Workshop: "Autonomy or Interdependence in Distributed Systems", *SIGOPS Review*, Vol. 23, No. 2, April 1989.
- [14] Ousterhout, J., Da Costa, H., Harrison, D., Kunze, J., Kupfer, M., "A Trace-Driven Analysis of the 4.2BSD File System", *Proceedings of the 10th ACM Symposium on Operating System Principles*, December 1985.
- [15] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B., "Design and Implementation of the Sun Network Filesystem", *Summer Usenix Conference Proceedings*, 1985.
- [16] Satyanarayanan, M., "On the Influence of Scale in a Distributed System", *Proceedings of the 10th International Conference on Software Engineering*, April 1988.
- [17] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., Steere, D.C., "Coda: A Highly Available File System for a Distributed Workstation Environment", *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990.

- [18] Satyanarayanan, M., "Scalable, Secure, and Highly Available Distributed File Access", *IEEE Computer*, Vol. 23, No. 5, May 1990.
- [19] Schroeder, M.D., Gifford, D.K., Needham, R.M., "A Caching File System for a Programmer's Workstation", *Proceedings of the 10th ACM Symposium on Operating System Principles*, December 1985.
- [20] Steere, D.C., Kistler, J.J., Satyanarayanan, M., "Efficient User-Level Cache File Management on the Sun Vnode Interface", *Summer Usenix Conference Proceedings*, Anaheim, June 1990.
- [21] "Decorum File System", *Transarc Corporation*, January 1990.
- [22] Walker, B., Popek, G., English, R., Kline, C., Thiel, G., "The LOCUS Distributed Operating System", *Proceedings of the 9th ACM Symposium on Operating System Principles*, October, 1983

EXPERIENCE WITH DISCONNECTED OPERATION IN A MOBILE COMPUTING ENVIRONMENT

M. Satyanarayanan, James J. Kistler,
Lily B. Mummert, Maria R. Ebling,
Puneet Kumar, and Qi Lu

*School of Computer Science,
Carnegie Mellon University.*

ABSTRACT

In this paper we present qualitative and quantitative data on file access in a mobile computing environment. This information is based on actual usage experience with the Coda File System over a period of about two years. Our experience confirms the viability and effectiveness of *disconnected operation*. It also exposes certain deficiencies of the current implementation of Coda, and identifies new functionality that would enhance its usefulness for mobile computing. The paper concludes with a description of what we are doing to address these issues.

1 INTRODUCTION

Portable computers are commonplace today. In conjunction with high- and low-bandwidth cordless networking technology, such computers will soon provide a pervasive hardware base for *mobile computing*. A key requirement of this new world of computing will be the ability to access critical data regardless of location. Data from shared file systems must be made available to programs running on mobile computers. But mobility poses serious impediments to meeting this requirement.

Permission has been granted by the USENIX Association to reprint this paper. This paper was originally published in the USENIX Association Conference Proceedings, 1993. Copyright ©USENIX Association, 1993.

We begin this paper by describing how shared file access is complicated by the constraints of mobile computing. We then show how the design of the *Coda File System* addresses these constraints. The bulk of the paper focuses on our usage experience with Coda. We present qualitative and quantitative data that shed light on Coda's design choices. Based on our experience, we have identified a number of ways in which Coda could be improved. The paper concludes with a description of our current work along these dimensions.

2 CONSTRAINTS OF MOBILE COMPUTING

Access to shared data in a mobile environment is complicated by three fundamental constraints. These constraints are intrinsic to mobility, and are not just artifacts of current technology:

- *Mobile elements are resource-poor relative to static elements.* For a given cost and level of technology, mobile elements are slower and have less memory and disk space than static elements. Weight, power, and size constraints will always conspire to preserve this inequity.
- *Mobile elements are more prone to loss, destruction, and subversion than static elements.* A Wall Street stockbroker is more likely to be mugged on the streets of Manhattan and have his or her laptop stolen than to have the workstation in a locked office be physically subverted. Even if security isn't a problem, portable computers are more vulnerable to loss or damage.
- *Mobile elements must operate under a much broader range of networking conditions.* A desktop workstation can typically rely on LAN or WAN connectivity. A laptop in a hotel room may only have modem or ISDN connectivity. Outdoors, a laptop with a cellular modem may find itself in intermittent contact with its nearest cell.

These constraints violate many of the assumptions upon which today's distributed systems are based. Further, the ubiquity of portable computers will result in mobile computing systems that are much larger than the distributed systems of today. *Scalability* will thus be a continuing concern.

Ideally, mobility should be completely *transparent* to users. Transparency relieves users of the need to be constantly aware of the details of their computing

environment, thus allowing them to focus on the real tasks at hand. The adaptation necessary to cope with the changing environment should be initiated by the system rather than by users. Of course, perfect transparency is an unattainable ideal. But that should not deter us from exploring techniques that enable us to come as close as possible to the ideal.

3 OVERVIEW OF CODA FILE SYSTEM

Coda, a descendant of the Andrew File System [4], offers continued access to data in the face of server and network failures. Earlier papers [7], [9], [14], [15], [16], [17] have described various aspects of Coda in depth. Here we only provide enough detail to make the rest of the paper comprehensible.

Coda is designed for an environment consisting of a large collection of untrusted Unix¹ clients and a much smaller number of trusted Unix file servers. The design is optimized for the access and sharing patterns typical of academic and research environments. It is specifically not intended for applications such as online transaction processing applications that exhibit highly concurrent, fine granularity update patterns.

Each Coda client has a local disk and can communicate with the servers over a high bandwidth network. Clients view Coda as a single, location-transparent shared Unix file system. The Coda namespace is mapped to individual file servers at the granularity of subtrees called *volumes*. At each client, a *cache manager* (*Venus*) dynamically obtains and caches data as well as volume mappings.

Coda uses two distinct, but complementary, mechanisms to achieve high availability. Both mechanisms rely on an *optimistic replica control* strategy. This offers the highest degree of availability, since data can be updated in any network partition. The system ensures detection and confinement of conflicting updates after their occurrence, and provides mechanisms to help users recover from such conflicts.

¹Unix is a trademark of Unix System Laboratories.

3.1 Server Replication

The first high-availability mechanism, *server replication*, allows volumes to have read-write replicas at more than one server. The set of replication sites for a volume is its *volume storage group (VSG)*. The subset of a VSG that is currently accessible is a client's *accessible VSG (AVSG)*. The performance cost of server replication is kept low by callback-based caching [6] at clients, and through the use of parallel access protocols. Modifications at a Coda client are propagated in parallel to all AVSG sites, and eventually to missing VSG sites.

3.2 Disconnected Operation

Although server replication is an important part of Coda, it is the second high-availability mechanism, *disconnected operation*, that is a key enabling technology for mobile computing [8]. A client becomes disconnected with respect to a volume when no server in its VSG is accessible. An *involuntary disconnection* can occur in a mobile computing environment when there is a temporary impediment to communication. This can be caused by limitations such as short range, inability to operate underground and in steel-framed buildings, or line-of-sight constraints. A *voluntary disconnection* can occur when a user deliberately operates isolated from a network. This may happen because no networking capability is available at the location of a mobile computer, or to avoid use of the network for cost or power consumption reasons.

While disconnected, Venus services file system requests by relying solely on the contents of its cache. Since cache misses cannot be serviced or masked, they appear as failures to application programs and users. The persistence of changes made while disconnected is achieved via an operation log implemented on top of a transactional facility called *RVM* [16]. Venus implements a number of optimizations to reduce the size of the operation log.

To support disconnected operation, Venus operates in one of three states: *hoarding*, *emulation*, and *reintegration*. Venus is normally in the hoarding state, relying on server replication but always on the alert for possible disconnection. The hoarding state is so named because a key responsibility of Venus in this state is to ensure that critical objects are in the cache at the moment of disconnection. Upon disconnection, Venus enters the emulation state and remains there for the duration of disconnection. Upon reconnection, Venus enters the reintegration state, resynchronizes its cache with its AVSG, and then reverts to the hoarding state.

Venus combines implicit and explicit sources of information in its *priority-based cache management* algorithm. The implicit information consists of recent reference history, as in traditional caching algorithms. Explicit information takes the form of a per-client *hoard database (HDB)*, whose entries are pathnames identifying objects of interest to the user at that client. A simple front-end program called *hoard* allows a user to update the HDB directly or via command scripts called *hoard profiles*.

Venus periodically reevaluates which objects merit retention in the cache via a process known as *hoard walking*. Hoard walking is necessary to meet user expectations about the relative importance of objects. When a cache meets these expectations, it is said to be in *equilibrium*.

4 IMPLEMENTATION STATUS

Disconnected operation in Coda was implemented over a period of two to three years. A version of disconnected operation with minimal functionality was demonstrated in October 1990. A more complete version was functional in early 1991 and began to be used regularly by members of the Coda group. By the end of 1991 almost all of the functionality had been implemented, and the user community had expanded to include several users outside the Coda group. Several of these new users had no connection to systems research whatsoever. Since mid-1992 implementation work has consisted mainly of performance tuning and bug-fixing. The current user community includes about 30 users, of whom about 20 use Coda on a regular basis. During 1992 the code was also made available to several sites outside of Carnegie Mellon University (CMU), and they are now using the system on a limited basis.

There are currently about 25 laptop and about 15 desktop clients in use. The laptops are mostly 386-based IBM PS2/L40's and the desktops are a mix of DECstation 5000/200's, Sun Sparcstations, and IBM RTs. We expect to be adding about 20 newer 486-based laptops in the near future. We currently have three DECstation 5000/200's with 2GB of disk storage in use as production servers, volumes being triply replicated across them. Additional servers are used for debugging and stress-testing pre-release versions of the system.

The production servers currently hold about 150 volumes. Roughly 25% of the volumes are *user* volumes, meaning that they are assigned to specific users who have sole administrative authority over them. Users are free, of course,

to extend access rights to others by changing access-control lists on specific objects in the volume. Approximately 65% of the volumes are *project* volumes, for which administrative rights are assigned collectively to the members of a group. Most of the project volumes are used by the Coda project itself, although there are three or four other groups which have some project volumes. The other 10% of the volumes are *system* volumes, which contain program binaries, libraries, header files, and the like.

To limit our logistical and manpower commitments, we use Coda in slightly different ways on our desktop and laptop clients. On desktop clients, Coda is currently used only for user and project data. The system portions of their namespaces are in AFS, and maintenance of these namespaces is by the CMU facilities staff. Disconnected operation on these machines is therefore restricted to cases in which AFS servers are accessible but Coda servers are not. Such cases can arise when Coda servers have crashed or are down for maintenance, or when a network partitioning has separated a client from the Coda servers but not from AFS servers.

Our mobile clients do not use AFS at all and are therefore completely dependent on Coda. The system portions of the name space for this machine type are maintained by us in Coda. To minimize this maintenance effort, we initially supported only a minimal subset of the system software and have grown the size of the supported subset only in response to user requests. This strategy has worked out very well in practice, resulting in a highly usable mobile computing environment. Indeed, there are many more people wishing to use Coda laptops than we can accommodate with hardware or support services.

Porting Coda to a new machine type is relatively straightforward. Most of the code is outside the kernel. The only in-kernel code, a VFS driver[17], is small and entirely machine independent. Porting simply involves recompiling the Coda client and server code, and ensuring that the kernel works on the specific piece of hardware.

5 QUALITATIVE EVALUATION

The nature of our testbed environment has meant that we have more experience with voluntary than with involuntary disconnected operation. The most common disconnection scenario has been a user detaching his or her laptop and taking it home to work in the evening or over the weekend. We have also had

cases where users have taken their laptops out of town, on business trips and on vacations, and operated disconnected for a week or more.

Although the dependence of our desktop workstations on AFS has limited our experience with involuntary disconnections, it has by no means eliminated it. Particularly during the early stages of development, the Coda servers were quite brittle and subject to fairly frequent crashes. When the crash involved corruption of server meta-data (alas, a common occurrence) repairing the problem could take hours or even days. Hence, there were many opportunities for clients to involuntarily operate disconnected from user and project data.

We present our observations of hoarding, server emulation, and reintegration in the next three sections. This is followed by a section with observations that apply to the architecture as a whole.

5.1 Hoarding

In our experience, hoarding has substantially improved the usefulness of disconnected operation. Disconnected cache misses have occurred, of course, and at times they were quite painful, but there is no doubt that both the number and the severity of those misses were dramatically reduced by hoarding. Moreover, this was realized without undue burden on users and without degradation of connected mode performance.

Our experience has confirmed one of the main premises of hoarding: that implicit and explicit sources of reference information are both important for avoiding disconnected cache misses, and that a simple function of hoard and reference priorities can effectively extract and combine the information content of both sources. It also confirms that the cache manager must actively respond to local and remote disequilibrating events if the cache state is to meet user expectations about availability. In the rest of this section we examine specific aspects of hoarding in more detail.

Hoard Profiles

The aggregation of hints into profiles is a natural step. If profiles had not been proposed and support for them had not been built into the `hoard` tool, it's certain that users would have come up with their own ad-hoc profile formulations and support mechanisms. No one, not even the least system-savvy of our users, has had trouble understanding the concept of a profile or making

```

# Personal files          # X11 files
a /coda/usr/satya 100:d+  # (from X11 maintainer)
a /coda/usr/satya/papers/mobile93 1000:d+ a /usr/X11/bin/X
                                            a /usr/X11/bin/Xvga
# System files           a /usr/X11/bin/mwm
a /usr/bin 100:d+        a /usr/X11/bin/startx
a /usr/etc 100:d+       a /usr/X11/bin/xclock
a /usr/include 100:d+   a /usr/X11/bin/xinit
a /usr/lib 100:d+       a /usr/X11/bin/xterm
a /usr/local/gnu d+    a /usr/X11/include/X11/bitmaps c+
a /usr/local/rcs d+    a /usr/X11/lib/app-defaults d+
a /usr/ucb d+          a /usr/X11/lib/fonts/misc c+
                                            a /usr/X11/lib/system.mwmrc

```

(a)

(b)

These are typical hoard profiles in actual use by some of our users. The 'a' at the beginning of a line indicates an add-entry command. Other commands are delete an entry, clear all entries, and list entries. The numbers following some pathnames specify hoard priorities (default 10). The 'c+' and 'd+' notations indicate meta-expansion, as explained in Section 1.

Figure 1 Sample Hoard Profiles

modifications to pre-existing profiles on their own. And, although there has been occasional direct manipulation of the HDB via the **hoard** tool, the vast majority of user/HDB interactions have been via profiles.

Most users employ about 5-10 profiles at any one time. Typically, this includes one profile representing the user's "personal" data: the contents of his or her root directory, notes and mail directories, etc. Several others cover the applications most commonly run by the user: the window system, editors and text formatters, compilers and development tools, and so forth. A third class of profile typically covers data sets: source code collections, publication and correspondence directories, collections of lecture notes, and so on. A user might keep a dozen or more profiles of this type, but only activate a few at a time (i.e., submit only a subset of them to the local Venus). The number of entries in most profiles is about 5-30, with very few exceeding 50. Figure 1 gives examples of typical hoard profiles.

Contrary to our expectations, there has been little direct sharing of profiles. Most of the sharing that has occurred has been indirect; that is, a user making

his or her own copy of a profile and then changing it slightly. There appear to be several explanations for this:

- early users of the system were not conscientious about placing application profiles in public areas of the namespace.
- our users are, for the most part, quite sophisticated. They are used to customizing their environments via files such as `.login` and `.Xdefaults` (and, indeed, many cannot resist the temptation to constantly do so).
- most of our users are working independently or on well-partitioned aspects of a few projects. Hence, there is not much incentive to share hoard profiles.

We expect that the degree of direct profile sharing will increase as our user community grows, and as less sophisticated users begin to use Coda.

Multi-Level Hoard Priorities

The earliest Coda design had only a single level of hoard priority; an object was either “sticky” or it was not. Sticky objects were expected to be in the cache at all times. Although the sticky approach would have been simpler to implement and easier for users to understand, we are certain that it would have been much less pleasant to use and far less effective in avoiding misses than our multi-level priority scheme.

We believe that a sticky scheme would have induced the following, undesirable types of hoarding behavior:

- a tendency to be conservative in specifying hints, to avoid pinning vast amounts of low-utility data.
- a proliferation of hoard profiles for the same task or data set into, for example, “small,” “medium,” and “large” variants.
- micro-management of the hoard database, to account for the facts that profiles would be smaller and more numerous and that the penalty for poor specification would be higher.

The net effect of all this is that much more time and effort would have been demanded by hoarding in a sticky scheme than is the case now. This would

have reduced the ability of users to hoard effectively, resulting in more frequent disconnected misses. Overall, the utility of disconnected operation would have been sharply reduced.

An argument besides simplicity which is sometimes used in favor of the sticky approach is that “you know for sure that a sticky object will be in the cache when you disconnect, whereas with priorities you only have increased probability that a hoarded object will be there.” That statement is simply not true. Consider a trivial example in which ten objects have been designated sticky and they occupy 90% of the total cache space. Now suppose that all ten are doubled in size by a user at another workstation. How can the local cache manager ensure that all sticky objects are cached? Clearly it cannot. The best it can do is re-fetch an arbitrary subset of the ten, leaving the rest uncached.

A negative aspect of our current priority scheme is that the range of hoard priorities is too large. Users are unable to classify objects into anywhere near 1000 equivalence classes, as the current system allows. In fact, they are often confused by such a wide range of choice. Examination of many private and a few shared profiles revealed that, while most contained at least two levels of priority, few contained more than three or four. Moreover, it was also apparent that no user employs more than six or seven distinct levels across all profiles. We therefore believe that future versions of the system should offer a priority range of about 1-10 instead of the current 1-1000. Such a change would reduce the uncertainty felt by some users as well as aid in the standardization of priorities across profiles.

Meta-Expansion

To reduce the verbosity of hoard profiles and to simplify their maintenance, Coda supports *meta-expansion* of HDB entries. If the letter ‘c’ (or ‘d’) follows a pathname in a hoard profile, the command also applies to immediate children (or all descendants). A ‘+’ following the ‘c’ or ‘d’ indicates that the command applies to all future as well as present children or descendants.

Meta-expansion has proven to be an indispensable feature of hoarding. Virtually all hoard profiles use it to some degree, and some use it exclusively. There are also many cases in which a profile would not even have been created had meta-expansion not been available. The effort in identifying the relevant individual names and maintaining the profile over time would simply have been too great. Indeed, it is quite possible that hoarding would never have reached a threshold level of acceptance if meta-expansion had not been an option.

A somewhat unexpected benefit of meta-expansion is that it allows profiles to be constructed incrementally. That is, a usable profile can almost always be had right away by including a single line of the form “`add <rootname> d+`,” where `<rootname>` is the directory heading the application or data set of interest. Typically, it is also wise to specify a low priority so that things don’t get out of hand if the sub-tree turns out to be very large. Later, as experience with the application or data set increases, the profile can be refined by removing the “root expansion” entry and replacing it with entries expanding its children. Children then known to be uninteresting can be omitted, and variations in priority can be incorporated. This process can be repeated indefinitely, with more and more hoarding effort resulting in better and better approximations of the user’s preferences.

Reference Spying

In many cases a user is not aware of the specific files accessed by an application. To facilitate construction of hoard profiles in such situations, Coda provides a spy program. This program can record all file references observed by Venus between a pair of start and stop events indicated by a user. Of course, different runtime behavior of the application can result in other files being accessed.

The `spy` program has been quite useful in deriving and tuning profiles. For example, it identified the reason why the X window system would sometimes hang when started from a disconnected workstation. It turns out that X font files are often stored in compressed format, with the X server expected to uncompress them as they are used. If the `uncompress` binary is not available when this occurs then the server will hang. Before `spy` was available, mysterious events such as this would happen in disconnected mode with annoying frequency. Since `spy`’s introduction we have been able to correct such problems on their first occurrence or, in many cases, avoid them altogether.

Periodic Hoard Walking

Background equilibration of the cache is an essential feature of hoarding. Without it there would be inadequate protection against involuntary disconnection. Even when voluntary disconnections are the primary type in an environment, periodic equilibration is still vital from a usability standpoint. First, it guards against a user who inadvertently forgets to demand a hoard walk before disconnecting. Second, it prevents a huge latency hit if and when a walk is demanded. This is very important because voluntary disconnections are often initiated

when time is critical—for example, before leaving for the airport or when one is already late for dinner. Psychologically, users find it comforting that their machine is always “mostly current” with the state of the world, and that it can be made “completely current” with very little delay. Indeed, after a short break-in period with the system, users take for granted the fact that they’ll be able to operate effectively if either voluntary or involuntary disconnection should occur.

Demand Hoard Walking

Foreground cache equilibration exists solely as an insurance mechanism for voluntary disconnections. The most common scenario for demand walking concerns a user who has been computing at their desktop workstation and is about to detach their laptop and take it home to continue work in the evening. In order to make sure that the latest versions of objects are cached, the user must force a hoard walk. An easy way to do this is to put the line “`hoard walk`” in one’s `.logout` file. Most users, however, seem to like the reassurance of issuing the command manually, and internalize it as part of their standard shutdown procedure. In any case, the requirement for demand walking before voluntary disconnection cannot be eliminated since the background walk period cannot be set too close to 0. This bit of non-transparency has not been a source of complaint from our users, but it could conceivably be a problem for a less sophisticated user community.

5.2 Server Emulation

Our qualitative evaluation of server emulation centers on two issues: transparency and cache misses.

Transparency

Server emulation by Venus has been quite successful in making disconnected operation transparent to users. Many involuntary disconnections have not been noticed at all, and for those that have the usual indication has been only a pause of a few seconds in the user’s foreground task at reintegration time. Even with voluntary disconnections, which by definition involve explicit manual actions, the smoothness of the transition has generally caused the user’s awareness of disconnection to fade quickly.

The high degree of transparency is directly attributable to our use of a single client agent to support both connected and disconnected operation. If, like FACE [1], we had used a design with separate agents and local data stores for connected and disconnected operation, then every transition between the two modes would have been visible to users. Such transitions would have entailed the substitution of different versions of the same logical objects, severely hurting transparency.

Cache Misses

Many disconnected sessions experienced by our users, including many sessions of extended duration, involved no cache misses whatsoever. We attribute this to two primary factors. First, as noted in the preceding subsection, hoarding has been a generally effective technique for our user population. Second, most of our disconnections were of the voluntary variety, and users typically embarked on those sessions with well-formed notions of the tasks they wanted to work on. For example, they took their laptop home with the intent of editing a particular paper or working on a particular software module; they did not normally disconnect with the thought of choosing among dozens of distinct tasks.

When disconnected misses did occur, they often were not fatal to the session. In most such cases the user was able to switch to another task for which the required objects were cached. Indeed, it was often possible for a user to “fall-back” on different tasks two or three times before they gave up and terminated the session. Although this is a result we expected, it was still quite a relief to observe it in practice. It confirmed our belief that hoarding need not be 100% effective in order for the system to be useful.

On a cache miss, the default behavior of Venus is to return an error code. A user may optionally request Venus to block processes until cache misses can be serviced. In our experience, users have made no real use of the blocking option for handling disconnected misses. We conjecture that this is due to the fact that all of our involuntary disconnections have occurred in the context of networks with high mean-time-to-repair (MTTR). We expect blocking will be a valuable and commonly used option in networks with low MTTRs.

5.3 Reintegration

Our qualitative evaluation of reintegration centers on two issues: performance and failures.

Performance

The latency of reintegration has not been a limiting factor in our experience. Most reintegrations have taken less than a minute to complete, with the majority having been in the range of 5-20 seconds. Moreover, many reintegrations have been triggered by background Venus activity rather than new user requests, so the perceived latency has often been nil.

Something which we have not experienced but consider a potential problem is the phenomenon of a *reintegration storm*. Such a storm could arise when many clients try to reintegrate with the same server at about the same time. This could occur, for instance, following recovery of a server or repair of a major network artery. The result could be serious overloading of the server and greatly increased reintegration times. We believe that we have not observed this phenomenon yet because our client population is too small and because most of our disconnections have been voluntary rather than the result of failures. We do, however, have two ideas on how the problem should be addressed:

- Have a server return a “busy” result once it reaches a threshold level of reintegration activity. Clients could back-off different amounts of time according to whether their reintegration was triggered by foreground or background activity, then retry. The back-off amounts in the foreground case would be relatively short and those in the background relatively long.
- Break operation logs into independent parts and reintegrate the parts separately. Of course, only the parts corresponding to foreground triggering should be reintegrated immediately; reintegration of the other parts should be delayed until the storm is over.

Detected Failures

Failed reintegrations have been very rare in our experience with Coda. The majority of failures that have occurred have been due to bugs in the implementation rather than update conflicts. We believe that this mostly reflects the low degree of write-sharing intrinsic to our environment. There is no doubt,

however, that it also reflects certain behavioral adjustments on the part of our users. The most significant such adjustments were the tendencies to favor indirect over direct forms of sharing, and to avoid synchronization actions when one was disconnected. So, for example, if two users were working on the same paper or software module, they would be much more likely to each make their own copy and work on it than they would to make incremental updates to the original object. Moreover, the “installation” of a changed copy would likely be delayed until a user was certain he or she was connected. Of course, this basic pattern of sharing is the dominant one found in any Unix environment. The observation here is that it appeared to be even more common among our users than is otherwise the case.

Although detected failures have been rare, recovering from those that have occurred has been irksome. If reintegration fails, Venus writes out the operation log and related container files to a local file called a *closure*. A tool is provided for the user to inspect the contents of a closure, to compare it to the state at the AVSG, and to replay it selectively or in its entirety.

Our approach of forming closures and storing them at clients has several problems:

- there may not be enough free space at the client to store the closure. This is particularly true in the case of laptops, on which disk space is already precious.
- the recovery process is tied to a particular client. This can be annoying if a user ever uses more than one machine.
- interpreting closures and recovering data from them requires at least an intermediate level of system expertise. Moreover, even for expert users it can be difficult to determine exactly why some reintegrations failed.

The first two limitations could be addressed by migrating closures to servers rather than keeping them at clients. That strategy was, in fact, part of the original design for disconnected operation, and it continues to look like a worthwhile option.

We believe that the third problem can be addressed through a combination of techniques that reduce the number of failures that must be handled manually, and simplify the handling of those that remain. We discuss our current work in this area in Section 7.3.

5.4 Other Observations

Optimistic Replication

The decision to use optimistic rather than pessimistic replica control was undoubtedly the most fundamental one in the design of Coda. Having used the system for more than two years now, we remain convinced that the decision was the correct one for our type of environment.

Any pessimistic protocol must, in one way or another, allocate the rights to access objects when disconnected to particular clients. This allocation involves an unpleasant compromise between availability and ease of use. On the one hand, eliminating user involvement increases the system's responsibility, thereby lowering the sophistication of the allocation decisions. Bad allocation decisions translate directly into lowered availability; a disconnected client either does not have a copy of a critical object, or has a copy that it cannot use because of insufficient rights. On the other hand, the more involved users are in the allocation process, the less transparent the system becomes.

An optimistic replication approach avoids the need to make a priori allocation decisions altogether. Our users have never been faced with the situation in which they are disconnected and have an object cached, but they cannot access it because of insufficient replica control rights. Similarly, they have never had to formally "grab control" of an object in anticipation of disconnection, nor have they had to "wrest control" from another client that had held rights they didn't really need. The absence of these situations has been a powerful factor in making the system effective and pleasant to use.

Of course, there is an advantage of pessimistic over optimistic replica control, which is that reintegration failures cannot occur. Our experience indicates that, in a Unix file system environment, this advantage is not worth much because there simply are very few failed reintegrations. The amount and nature of sharing in the workload make reintegration failures unlikely, and users adopt work habits that reduce their likelihood even further. In effect, the necessary degree of cross-partition synchronization is achieved voluntarily, rather than being enforced by a pessimistic algorithm.

Herlihy [3] once gave the following motivation for optimistic concurrency control, which applies equally well to optimistic replica control:

...[optimistic replica control] is based on the premise that it is more effective to apologize than to ask permission.

In our environment, the cases in which one would wrongfully be told “no” when asking permission vastly outnumber those in which a “no” would be justified. Hence, we have found it far better to suffer the occasional indignity of making an apology than the frequent penalty of a wrongful denial.

Security

There have been no detected violations of security in our use of Coda, and we believe that there have been no undetected violations either. The friendliness of our testbed environment is undoubtedly one important explanation for this. However, we believe that the Coda implementation would do well security-wise even under more hostile conditions.

The basis for this belief is the faithful emulation of the AFS security model. Coda servers demand to see a user’s credentials on every client request, including reintegration. Credentials can be stolen, but this requires subversion of a client or a network-based attack. Network attacks can be thwarted through the use of (optional) message encryption, and the danger of stolen credentials is limited by associating fixed lifetimes with them. Access-control lists further limit the damage due to credential theft by confining it to areas of the namespace legitimately accessible to the subverted principal. Disconnected operation provides no back-doors that can be used to circumvent these controls.

AFS has provided good security at large-scale and under circumstances that are traditionally somewhat hostile. Indeed, we know of no other distributed file system in widespread use that provides better security with a comparable level of functionality. This strongly suggests that security would not be a factor limiting Coda’s deployment beyond our testbed environment.

Public Workstations

Some computing environments include a number of public workstation clusters. Although it was never a primary goal to support disconnected operation in that domain, it was something that we hoped would be possible and which influenced Coda’s early design to some degree.

Our experience with disconnected operation has convinced us that it is simply not well suited to public access conditions. One problem is that of security. Without disconnected operation, it is the case that when a user leaves a public workstation his or her data is all safely at servers and he or she is totally independent of that workstation. This allows careful users to flush their authentication tokens and their sensitive data from the cache when they depart, and to similarly “scrub” the workstation clean when they arrive. But with disconnected operation, scrubbing is not necessarily an option. The departing user cannot scrub if he or she has dirty objects in the cache, waiting to be reintegrated. The need to leave valid authentication tokens with the cache manager is particularly worrying, as that exposes the user to arbitrary damage. And even if damage does not arise due to security breach, the departing user still must worry that a future user will scrub the machine and thereby lose his or her pending updates.

The other major factor that makes disconnected operation unsuited to public workstations is the latency associated with hoarding. Loading a cache with one’s full “hoardable set” can take many minutes. Although this is done in the background, it can still slow a client machine down considerably. Moreover, if a user only intends to use a machine briefly, as is often the case with public machines, then the effort of hoarding is likely to be a waste. It is only when the cost of hoarding can be amortized over a long usage period that it becomes a worthwhile exercise.

6 QUANTITATIVE EVALUATION

An earlier paper [7] presented measurements that shed light on key aspects of disconnected operation in Coda. Perhaps the most valuable of those measurements was the compelling evidence that optimistic replication in a Coda-like environment would indeed lead to very few write-write conflicts. That evidence was based on a year-long study of a 400-user AFS cell at CMU. The data showed that cross-user write-sharing was extremely rare. Over 99% of all file and directory modifications were by the previous writer, and the chances of two different users modifying the same object less than a day apart was at most 0.72%. If certain system administration files which skewed the data were excluded, the absence of write-sharing was even more striking: more than 99.7% of all mutations were by the previous writer, and the chances of two different users modifying the same object within a week were less than 0.3%.

Trace Identifier	Machine Name	Machine Type	Simulation Start	Trace Records
Work-Day #1	brahms.coda.cs.cmu.edu	IBM RT-PC	25-Mar-91, 11:00	195289
Work-Day #2	holst.coda.cs.cmu.edu	DECstation 3100	22-Feb-91, 09:15	348589
Work-Day #3	ives.coda.cs.cmu.edu	DECstation 3100	05-Mar-91, 08:45	134497
Work-Day #4	mozart.coda.cs.cmu.edu	DECstation 3100	11-Mar-91, 11:45	238626
Work-Day #5	verdi.coda.cs.cmu.edu	DECstation 3100	21-Feb-91, 12:00	294211
Full-Week #1	concord.nectar.cs.cmu.edu	Sun 4/330	26-Jul-91, 11:41	3948544
Full-Week #2	holst.coda.cs.cmu.edu	DECstation 3100	18-Aug-91, 23:21	3492335
Full-Week #3	ives.coda.cs.cmu.edu	DECstation 3100	03-May-91, 12:15	4129775
Full-Week #4	messiaen.coda.cs.cmu.edu	DECstation 3100	27-Sep-91, 00:15	1613911
Full-Week #5	purcell.coda.cs.cmu.edu	DECstation 3100	21-Aug-91, 14:47	2173191

Table 1 Vital Statistics for the Work-Day and Full-Week Traces

In the following sections we present new measurements of Coda. These measurements either address questions not considered in our earlier paper, or provide more detailed and up-to-date data on issues previously addressed. The questions we address here are:

- How large a local disk does one need?
- How noticeable is reintegration?
- How important are optimizations to the operation log?

6.1 Methodology

To estimate disk space requirements we relied on simulations driven by an extensive set of file reference traces that we had collected [12]. Our analysis was comprehensive, taking into account the effect of all references in a trace whether they were to Coda, AFS or the local file system. The traces were carefully selected for sustained high levels of activity from over 1700 samples. We chose 10 workstation traces, 5 representing 12-hour workdays and the other 5 representing week-long activity. Table 1 identifies these traces and presents a summary of the key characteristics of each.

To address the question of reintegration latency, we performed a well-defined set of activities while disconnected and timed the duration of the reintegration phase after each. One of these activities was the running of the Andrew benchmark [4]. Another was the compilation of the then-current version of Venus. A third class of activities corresponded to the set of traces in Table 1. We

effectively “inverted” these traces and generated a command script from each. When executed, each of these scripts produced a trace isomorphic to the one it was generated from. This gave us a controlled and repeatable way of emulating real user activity.

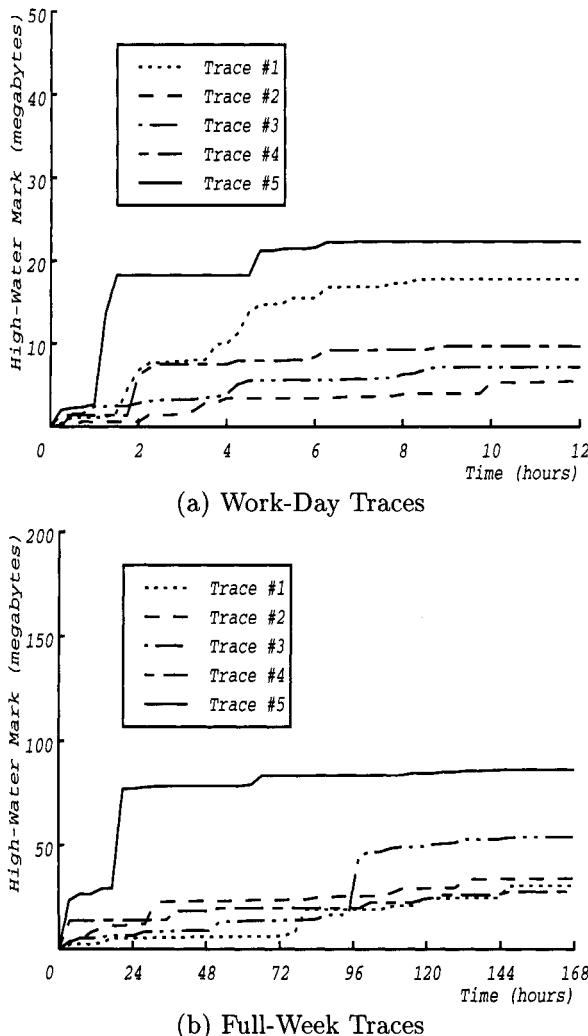
We combined these two techniques to assess the value of log optimizations. Using our trace suite, we compared disk usage with and without optimizations enabled. We also measured the impact of log optimizations on reintegration latency for the set of activities described in the previous paragraph.

6.2 Disk Space Requirements

Figure 2 shows the *high-water mark* of cache space usage for the Work-Day and Full-Week traces as a function of time. The high-water mark is simply the maximum cache space in use at the current and all previous points in the simulation. The high-water mark therefore never declines, although the current cache space in use may (due to the deletion of objects).

These curves indicate that cache space usage tends to grow rapidly at the start, but tapers off quite soon. For example, most of the Work-Day traces had reached 80% of their 12-hour high-water marks within a few hours of their start. Similarly, all but one of the Full-Week traces had reached a substantial fraction of their 7-day high-water marks by the end of the second day. Note that it was not the case that the workstations simply became idle after the first parts of the traces; the traces were carefully selected to ensure that users were active right through to the end of the simulated periods.

These results are encouraging from the point of view of disconnected operation. The most expansive of the Work-Day traces peaked out below 25 MB, with the median of the traces peaking at around 10 MB. For the Full-Week traces, the maximum level reached was under 100 MB and the median was under 50 MB. This suggests that today’s typical desktop workstation, with a disk of 100 MB to 1 GB, should be able to support many disconnections of a week or more in duration. Even the 60-200MB disk capacity of many laptops today is adequate for extended periods of disconnected operation. These observations corroborate our first-hand experience in using Coda laptops.



This graph presents the high-water marks of cache usage for each trace in Table 1. Note that the vertical axis on the graphs for Work-Day and Full-Week traces are different.

Figure 2 High-Water Marks of Cache Space Usage

6.3 Reintegration Latency

Reintegration latency is a function of the update activity at a client while disconnected. In our use of the system, most one-day disconnections have resulted

Task	Log Record Total	Back-Fetch Total	Prelude	Latency Interlude	Postlude	Latency Total	
Andrew Benchmark	203	1.2	1.8	7.5	.8	10	(1)
Venus Make	146	10.3	1.4	36.2	.4	38	(1)
Work-Day #1 Replay	1422	4.9	6.5	54.7	10.7	72	(5)
Work-Day #2 Replay	316	.9	1.9	9.8	1.7	14	(1)
Work-Day #3 Replay	212	.8	1.0	6.2	.9	8	(0)
Work-Day #4 Replay	873	1.3	2.9	23.2	5.9	32	(3)
Work-Day #5 Replay	99	4.0	.9	20.5	.5	22	(2)
Full-Week #1 Replay	1802	15.9	15.2	138.8	21.9	176	(3)
Full-Week #2 Replay	1664	17.5	16.2	129.1	15.0	160	(2)
Full-Week #3 Replay	7199	23.7	152.6	881.3	183.0	1217	(12)
Full-Week #4 Replay	1159	15.1	5.1	77.4	7.0	90	(1)
Full-Week #5 Replay	2676	35.8	28.2	212.8	31.7	273	(9)

This data was obtained with a DECstation 5000/200 client and server. The Back-Fetch figures are in megabytes. Latency figures are in seconds. Each latency number is the mean of three trials. The numbers in parentheses in the "Latency Total" column are standard deviations. Standard deviations for the individual phases are omitted for space reasons.

Table 2 Reintegration Latency

in reintegration times of a minute or less, and a few longer disconnections have taken a few minutes. Table 2 reports the latency, number of log records, and amount of data *back-fetched* for each of our reintegration experiments. Back-fetching refers to the transfer of data from client to server representing disconnected file store operations. Reintegration occurs in three subphases: a *prelude*, an *interlude*, and *postlude*. Latency is reported separately for the subphases as well as in total. On average, these subphases contributed 10%, 80% and 10% respectively to the total latency.

These results confirm our subjective experience that reintegration after a typical one-day disconnection is hardly perceptible. The Andrew benchmark, Venus make, and four of the five Work-Day trace-replay experiments all reintegrated in under 40 seconds. The other Work-Day trace-replay experiment took only slightly more than a minute to reintegrate.

The reintegration times for the week-long traces are also consistent with our qualitative observations. Four of the five week-long trace-replay experiments reintegrated in under five minutes, with three completing in three minutes or less. The other trace-replay experiment is an outlier, requiring about 20 minutes to reintegrate.

In tracking down the reason for this anomaly, we discovered a significant shortcoming of our implementation. We found, much to our surprise, that the time for reintegration bore a non-linear relationship to the size of the operation log and the number of bytes back-fetched. Specifically, the regression coefficients were .026 for the number of log records, .0000186 for its square, and 2.535 for the number of megabytes back-fetched. The quality of fit was excellent, with an R^2 value of 0.999.

The first coefficient implies a direct overhead per log record of 26 milliseconds. This seems about right, given that many records will require at least one disk access at the server during the interlude phase. The third coefficient implies a rate of about 400 KB/s for bulk data transfer. This too seems about right, given that the maximum disk-to-disk transfer rate between 2 DECstation 5000/200s on an Ethernet that we've observed is 476 kilobytes/second. The source of the quadratic term turned out to be a naive sorting algorithm that was used on the servers to avoid deadlocks during replay. For disconnected sessions of less than a week, the linear terms dominate the quadratic term. This explains why we have never observed long reintegration times in normal use of Coda. But around a week, the quadratic term begins to dominate. Clearly some implementation changes will be necessary to make reintegration linear. We do not see these as being conceptually difficult, but they will require a fair amount of code modification.

It is worth making two additional points about reintegration latency here. First, because reintegration is often triggered by a daemon rather than a user request, perceived latency is often nil. That is, reintegrations often occur entirely in the background and do not delay user computation at all. Second, the trace-replay experiments reflect activity that was originally performed in a number of volumes. For the Work-Day traces 5-10 volumes were typically involved, and for the Full-Week traces the number was typically 10-15. For logistical reasons, the replay experiments were each performed within a single Coda volume. Hence, there was only one reintegration for each experiment. Following an actual disconnected execution of the trace activity, though, there would have been a number of smaller reintegrations instead of one large one. If the reintegrated volumes were spread over different servers, a significant amount of parallelism could have been realized. The total latency might therefore have been much smaller, perhaps by a factor of three or four.

6.4 Value of Log Optimizations

Venus uses a number of optimizations to reduce the length of the operation log. A small log conserves disk space, a critical resource during periods of disconnection. It also improves reintegration performance by reducing latency and server load. Details of these optimizations can be found elsewhere [7], [8].

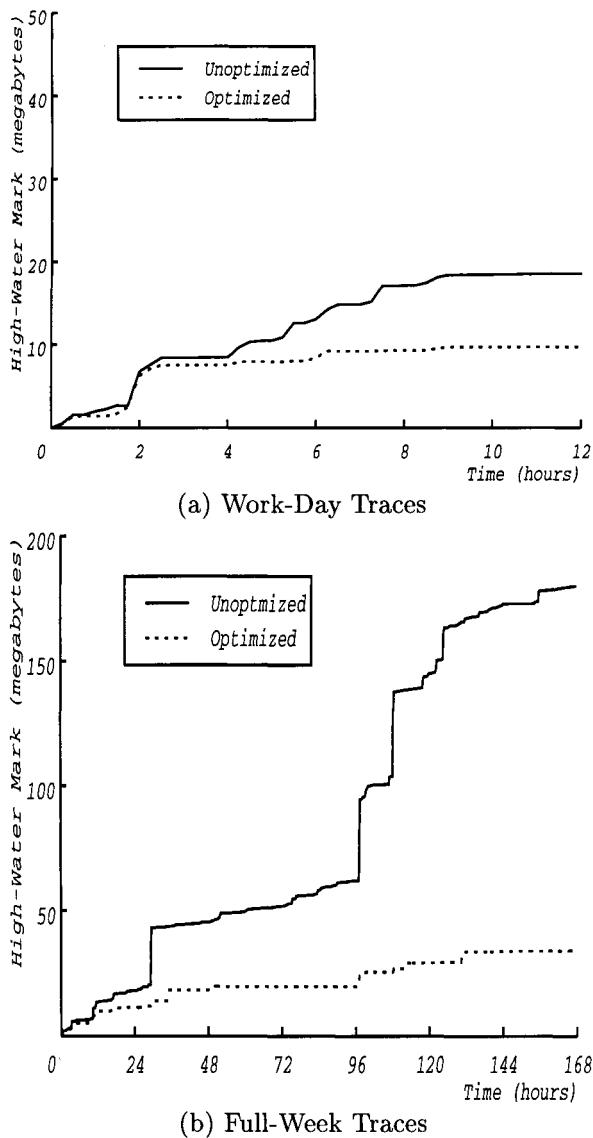
In order to understand how much space these optimizations save in practice, our Venus simulator was augmented to report cache usage statistics with the optimizations turned off as well as on. Figure 3 compares the median high-water marks of space usage for our trace suite with and without optimizations.

The differences between the curves in each case are substantial. After an initial period in which the two curves increase more or less together, the unoptimized curves continue to increase while the optimized curves taper off. For the Work-Day traces, the unoptimized total has grown to nearly twice that of the optimized case by the 12-hour mark. The trend continues unabated with the Full-Week traces, with the unoptimized total being more than 5 times that of the optimized case at the end of the week. This equates to a difference of more than 145 megabytes. The slopes of the two lines indicate that the difference would increase even further over periods of greater length.

Table 3 shows that the differences for certain individual traces are even more striking. That table lists the unoptimized and optimized totals for each trace at its termination. In addition, each total is broken down into its two constituents: cache container space and RVM space. Cache container space refers to the space used by the local files that are used to hold the images of the current versions of Coda files.

The greatest savings tend to be realized in cache container space, although the RVM space savings can also be substantial. The far right column shows the ratio of unoptimized to optimized total space usage. The maximum ratio for the Work-Day traces is 3.1, indicating that more than three times the amount of space would have been needed without the optimizations. The maximum ratio for the Full-Week traces is an astonishing 28.9, which corresponds to a difference of more than 850 megabytes.

These results confirm that log optimizations are critical for managing space at a disconnected client. But they are also important for keeping reintegration latency low. To confirm this, we used the regression results and measured values of unoptimized log records and data back-fetched from the experiments reported



Each curve above represents the median values of the high-water marks of space usage for the five corresponding traces. Note that the vertical axis on the two graphs are different.

Figure 3 Optimized versus Unoptimized Cache Space High-Water Marks

Trace	Container Space		RVM Space		Unopt	Total Spcae	
	Unopt	Opt	Unopt	Opt		Opt	Ratio
Work-Day #1	34.6	15.2	2.9	2.7	37.5	17.8	2.1
Work-Day #2	14.9	4.0	2.3	1.5	17.2	5.5	3.1
Work-Day #3	7.9	5.8	1.6	1.4	9.5	7.2	1.3
Work-Day #4	16.7	8.2	1.9	1.5	18.6	9.7	1.9
Work-Day #5	59.2	21.3	1.2	1.1	60.4	22.3	2.7
Full-Week #1	872.6	25.9	11.7	4.8	884.3	30.6	28.9
Full-Week #2	90.7	28.3	13.3	5.9	104.0	34.2	3.0
Full-Week #3	119.9	45.0	46.2	9.1	165.9	54.0	3.1
Full-Week #4	222.1	23.9	5.5	3.9	227.5	27.7	8.2
Full-Week #5	170.8	79.0	9.1	7.7	179.8	86.5	2.1

The figures in the “Unopt” and “Opt” columns are in megabytes

Table 3 Optimized versus Unoptimized Space Usage

Task	Log Record Total		Back-Fetch Total		Unopt	Latency	
	Unopt	Opt	Unopt	Opt		Opt	Ratio
Andrew Benchmark	211	203	1.5	1.2	10	10	1.0
Venus Make	156	146	19.5	10.3	54	38	1.4
Work-Day #1 Replay	2422	1422	19.1	4.9	221	72	3.1
Work-Day #2 Replay	4093	316	10.9	.9	446	14	31.9
Work-Day #3 Replay	842	212	2.1	.8	41	8	5.1
Work-Day #4 Replay	2439	873	8.5	1.3	196	32	6.1
Work-Day #5 Replay	545	99	40.9	4.0	123	22	5.6
Full-Week #1 Replay	33923	1802	846.9	15.9	24433	176	138.8
Full-Week #2 Replay	36855	1664	62.4	17.5	26381	160	164.9
Full-Week #3 Replay	175392	7199	75.0	23.7	576930	1217	474.1
Full-Week #4 Replay	8519	1159	199.1	15.1	2076	90	23.1
Full-Week #5 Replay	8873	2676	92.7	35.8	1930	273	7.1

Back-fetch figures are in megabytes, and latencies in seconds. The reported latencies are the means of three trials. Standard deviations are omitted for brevity.

Table 4 Optimized versus Unoptimized Reintegration Latency

in Section 6.3. Using this information we estimated how long reintegration would have taken, had log optimizations not been done. Table 4 presents our results.

The time savings due to optimizations are enormous. The figures indicate that without the optimizations, reintegration of the trace-replay experiments would have averaged 10 times longer than actually occurred for the Work-Day set, and 160 times longer for the Full-Week set. Reintegrating the unoptimized replay of Full-Week trace #3 would have taken more than 6 days, or nearly as long as the period of disconnection! Obviously, much of the extra time is due to the fact that the unoptimized log record totals are well into the range at which the quadratic steps of our implementation dominate. Although the savings will not be as great when our code is made more efficient, it will not be inconsequential by any means. Even if the quadratic term is ignored, the ratios of unoptimized to optimized latency are still pronounced: on average, 4.5 for Work-Day traces and 7.6 for the Full-Week traces.

7 WORK IN PROGRESS

Coda is a system under active development. In the following sections we describe work currently under way to enhance the functionality of Coda as well as to alleviate some of its current shortcomings.

7.1 Exploiting Weak Connectivity

Although disconnected operation in Coda has proven to be effective for using distributed file systems from mobile computers, it has several limitations:

- cache misses are not transparent. A user may be able to work in spite of some cache misses, but certain critical misses may frustrate these efforts.
- longer disconnections increase the likelihood of resource exhaustion on the client from the growing operation log and new data in the cache.
- longer disconnections also increase the probability of conflicts requiring manual intervention upon reconnection.

Wireless technologies such as cellular phone and even traditional dialup lines present an opportunity to alleviate some of the shortcomings of disconnected operation. These *weak connections* are slower than LANs, and some of the

wireless technologies have the additional properties of intermittence and non-trivial cost. The characteristics of these networks differ substantially from those of LANs, on which many distributed file systems are based.

We are exploring techniques to exploit weak connectivity in a number of ways:

- Coda clients will manage use of the network intelligently. By using the network in a preemptive, prioritized fashion, the system will be able to promptly service critical cache misses. It will propagate mutations back to the servers in the background to prevent conflicts that would arise at reintegration time and to reclaim local resources. It will allow users to weaken consistency on an object-specific basis to save bandwidth.
- Coda clients will minimize bandwidth requirements by using techniques such as batching and compression. Techniques in this class demand more server computation per request, so the state of the server will play a role in the use of these techniques.
- Coda clients will dynamically detect and adapt to changes in network performance. This will be especially important when connectivity is intermittent.

An important consideration in the use of weak connections is the issue of callback maintenance. Callback-based cache consistency schemes were designed to minimize client-server communication, but with an underlying assumption that the network is fast and reliable. After a network failure all callbacks are invalid. In an intermittent low-bandwidth network, the cost of revalidation may be substantial and may nullify the performance benefits of callback-based caching.

To address this issue we have introduced the concept of *large granularity callbacks*. A large granularity trades off precision of invalidation for speed of validation after connectivity changes. Venus will choose the granularity on a per-volume basis, adapting to the current networking conditions as well as the observed rate of callback breaks due to mutations elsewhere in the system. Further details on this approach can be found in a recent paper [11].

7.2 Hoarding Improvements

We are in the process of developing tools and techniques to reduce the burden of hoarding on users, and to assist them in accurately assessing which files to hoard. A key problem we are addressing in this context is the choice of proper metrics for evaluating the quality of hoarding.

Today, the only metric of caching quality is the *miss ratio*. The underlying assumption of this metric is that all cache misses are equivalent (that is, all cache misses exact roughly the same penalty from the user). This assumption is valid in the absense of disconnections and weak connections because the performance penalty resulting from a cache miss is small and independent of file length. This assumption is not valid during disconnected operation and may not be valid for weakly-connected operation, depending on the strength of the connection. The cache miss ratio further assumes that the timing of cache misses is irrelevant. But the user may react differently to a cache miss occurring within the first few minutes of disconnection than to one occurring near the end of the disconnection.

We are extending the analysis of hoarding tools and techniques using new metrics such as:

- the time until the first cache miss occurs.
- the time until a critical cache miss occurs.
- the time until the cumulative effect of multiple cache misses exceeds a threshold.
- the time until connection transparency is lost.
- the percentage of the cache actually referenced when disconnected or weakly connected, as a measure of overly-generous hoarding.
- the change in connected-mode miss ratio due to hoarding.

We plan to use these metrics to evaluate the relative value of different kinds of hoarding assistance. For example, under what circumstances does one tool prove better than another? Some of our experiments will be performed on-line as users work. Others will be performed off-line using post-mortem analysis of file reference traces. The tools we plan to build will address a variety of needs pertinent to hoarding. Examples include tools to support task-based hoarding

and a graphical interface to accept hoard information and provide feedback regarding the cache contents.

7.3 Application-Specific Conflict Resolution

Our experience with Coda has established the importance of optimistic replication for mobile computing. But optimistic replication brings with it the need to detect and resolve concurrent updates in multiple partitions. Today Coda provides for transparent resolution of directory updates. We are extending our work to support transparent resolution on arbitrary files. Since the operating system does not possess any semantic knowledge of file contents, it is necessary to obtain assistance from applications.

The key is to provide an application-independent invocation mechanism that allows pre-installed, *application-specific resolvers (ASRs)* to be transparently invoked and executed when a conflict is detected. As a practical example of this approach, consider a calendar management application. The ASR in this case might merge appointment database copies by selecting all non-conflicting appointments and, for those time slots with conflicts, choosing to retain one arbitrarily and sending mail to the rejected party(s).

We have recently described such an interface for supporting ASRs [10]. Our design addresses the following issues:

- An application-independent interface for transparently invoking ASRs.
- An inheritance mechanism to allow convenient rule-based specification of ASRs based on attributes such as file extension or position in the naming hierarchy.
- A fault tolerance mechanism that encapsulates ASR execution.

Even in situations where manual intervention is unavoidable, ASR technology may be used for partial automation. Consider, for example, the case of two users who have both edited a document or program source file. An “interactive ASR” could be employed in this case which pops up side-by-side windows containing the two versions and highlights the sections which differ. The user could then quickly perform the merge by cutting and pasting. Similarly, a more useful version of the calendar management ASR might begin with a view of the appointment schedule merged with respect to all non-conflicting time

slots, then prompt the user to choose between the alternatives for each slot that conflicts.

Another class of ASRs that may be valuable involves automatic re-execution of rejected computations by Venus. This is precisely the approach advocated by Davidson in her seminal work on optimistic replication in databases [2], and it will be feasible to use in Coda once the transactional extensions described in Section 7.4 are completed. Automatic re-execution would be appropriate in many cases involving the `make` program, for example.

7.4 Transactional Extensions for Mobile Computing

With the increasing frequency and scale of data sharing activities made possible by distributed Unix file systems such as AFS, Coda, and NFS [13], there is a growing need for effective consistency support for concurrent file accesses. The problem is especially acute in the case of mobile computing, because extended periods of disconnected or weakly-connected operation may increase the probability of read-write inconsistencies in shared data.

Consider, for example, a CEO using a disconnected laptop to work on a report for an upcoming shareholder's meeting. Before disconnection she cached a spreadsheet with the most recent budget figures available. She writes her report based on the numbers in that spreadsheet. During her absence, new budget figures become available and the server's copy of the spreadsheet is updated. When the CEO returns and reintegrates, she needs to discover that her report is based on stale budget data. Note that this is not a write-write conflict, since no one else has updated her report. Rather it is a read-write conflict, between the spreadsheet and the report. No Unix system today has the ability to detect and deal with such problems.

We are exploring techniques for extending the Unix interface with transactions to provide this functionality. A key attribute of our effort is upward compatibility with the Unix paradigm. Direct transplantation of traditional database transactions into Unix is inappropriate. The significant differences in user environment, transaction duration and object size between Unix file systems and database systems requires transactional mechanisms to be specially tailored. These considerations are central to our design of a new kind of transaction, called *isolation-only transaction*, whose use will improve the consistency properties of Unix file access in partitioned networking environments.

A distinct but related area of investigation is to explore the effects of out-of-band communication on mobile computing. For example, a disconnected user may receive information via a fax or phone call that he incorporates into the documents he is working on. What system support can we provide him to demarcate work done before that out-of-band communication? This will become important if he later needs to extricate himself from a write-write or read-write conflict.

8 CONCLUSIONS

In this paper, we have focused on disconnected operation almost to the exclusion of server replication. This is primarily because disconnected operation is the newer concept, and because it is so central to solving the problems that arise in mobile computing. However, the importance of server replication should not be underestimated. Server replication is important because it reduces the frequency and duration of disconnected operation. Thus server replication and disconnected operation are properly viewed as complementary mechanisms for high availability.

Since our original description of disconnected operation in Coda [7] there has been considerable interest in incorporating this idea into other systems. One example is the work by Huston and Honeyman [5] in implementing disconnected operation in AFS. These efforts, together with our own substantial experience with disconnected operation in Coda, are evidence of the soundness of the underlying concept and the feasibility of its effective implementation.

None of the shortcomings exposed in over two years of serious use of disconnected operation in Coda are fatal. Rather, they all point to desirable ways in which the system should evolve. We are actively refining the system along these dimensions, and have every reason to believe that these refinements will render Coda an even more usable and effective platform for mobile computing.

Acknowledgements

We wish to thank all the members of the Coda project, past and present, for their contributions to this work. David Steere, Brian Noble, Hank Mashburn, and Josh Raiff deserve special mention. We also wish to thank our brave and

tolerant user community for their willingness to use an experimental system. This work was supported by the Advanced Research Projects Agency (Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division(AFSC), U.S. Air Force, Wright-Patterson AFB under Contract F33615-90-C-1465, Arpa Order No. 7597), the National Science Foundation (Grant ECD 8907068), IBM, Digital Equipment Corporation, and Bellcore.

REFERENCES

- [1] Cova, L.L., "Resource Management in Federated Computing Environments", *Ph.D. Thesis, Department of Computer Science, Princeton University*, October 1990.
- [2] Davidson, S., "Optimism and Consistency in Partitioned Distributed Database Systems", *ACM Transactions on Database Systems*, Vol. 3, No. 9, September 1984.
- [3] Herlihy, M., "Optimistic Concurrency Control for Abstract Data Types", *Proceedings of the Fifth Annual Symposium on Principles of Distributed Computing*, August 1986.
- [4] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., West, M.J., "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988.
- [5] Huston, L., Honeyman, P., "Disconnected Operation for AFS", *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, Cambridge, MA, August 1993.
- [6] Kazar, M.L., "Synchronization and Caching Issues in the Andrew File System", *Winter Usenix Conference Proceedings*, Dallas, TX, 1988.
- [7] Kistler, J.J., Satyanarayanan, M., "Disconnected Operation in the Coda File System", *ACM Transactions on Computer Systems*, Vol. 10, No. 1, February 1992.
- [8] Kistler, J.J., "Disconnected Operation in a Distributed File System", *Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University*, May 1993.

- [9] Kumar, P., Satyanarayanan, M., "Log-Based Directory Resolution in the Coda File System", *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, San Diego, CA, January 1993.
- [10] Kumar, P., Satyanarayanan, M., "Supporting Application-Specific Resolution in an Optimistically Replicated File System", submitted to the *4th IEEE Workshop on Workstation Operating Systems*, Napa, CA, October 1993.
- [11] Mummert, L.B., Satyanarayanan, M., "File Cache Consistency in a Weakly Connected Environment", submitted to the *4th IEEE Workshop on Workstation Operating Systems*, Napa, CA, October 1993.
- [12] Mummert, L.B., "Efficient Long-Term File Reference Tracing", In preparation, 1993.
- [13] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B., "Design and Implementation of the Sun Network Filesystem", *Summer Usenix Conference Proceedings*, 1985.
- [14] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., Steere, D.C., "Coda: A Highly Available File System for a Distributed Workstation Environment", *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990.
- [15] Satyanarayanan, M., "Scalable, Secure, and Highly Available Distributed File Access", *IEEE Computer*, Vol. 23, No. 5, May 1990.
- [16] Satyanarayanan, M., Mashburn, H.H., Kumar, P., Steere, D.C., Kistler, J.J., "Lightweight Recoverable Virtual Memory", *School of Computer Science, Carnegie Mellon University*, CMU-CS-93-143, March 1993.
- [17] Steere, D.C., Kistler, J.J., Satyanarayanan, M., "Efficient User-Level Cache File Management on the Sun Vnode Interface", *Summer Usenix Conference Proceedings*, Anaheim, June 1990.

MOBILITY SUPPORT FOR SALES AND INVENTORY APPLICATIONS

Narayanan Krishnakumar and Ravi Jain

*Bell Communications Research,
445 South Street, Morristown, NJ 07960*

ABSTRACT

An important and challenging set of issues in mobile computing is the design of architectures and protocols for providing mobile users with integrated Personal Information Services and Applications (PISA), such as personalized news and financial information, and mobile sales and banking. We present a system architecture for delivery of PISA based on replicated distributed servers connected to users via a personal communications services (PCS) network. The PISA architecture takes advantage of many of the basic facilities incorporated in proposed PCS network designs.

We focus on the mobile sales and inventory application as an example of a PISA with a well-defined market segment. We deal with both application-level and infrastructure-level protocols supporting this application. At the application level, we describe a database design and protocol which supports both mobile and stationary salespersons. A key principle behind our design choices is to minimize those aspects which are needed solely to support user mobility. We propose using a site escrow method and reconfiguration protocol for supporting sales transactions, and discuss how mobile salespersons can be accommodated in this scheme. On the infrastructure-level, we discuss how the service profiles of the mobile salespersons can be maintained using a two-level hierarchy of profile databases, and show that a protocol more robust than that used for maintaining their location is needed. We finally present such a protocol for maintaining service profiles.

1 INTRODUCTION

An important and challenging area of mobile information systems is the design of architectures and protocols for providing mobile users with Personal Information Services and Applications (PISA). Examples of PISA include personalized financial and stock market information, electronic magazines, news clipping services, traveler information, as well as mobile shopping, banking, sales, inventory, and file access. As a concrete running example in this paper, we use the mobile sales and inventory application. In sec. 2 we describe this application in more detail.

We consider the situation in which PISA are primarily provided by a commercial entity called the Information Service and Applications Provider (ISAP). The ISAP maintains a set of servers which contain the appropriate information and run applications, and which are connected to the mobile user via a personal communications services (PCS) network. The ISAP need not be the same commercial and administrative entity as the PCS network provider.

The mobile user's terminal runs application software to interact with the ISAP. These interactions are divided into logical, application-dependent segments called *sessions*. For example, in the case of the mobile sales and inventory application, a session may consist of the salesperson connecting to a remote server, downloading images and text describing a product, and then logging out. Sessions may be initiated by the user or by the ISAP. The salesperson may be an employee of the ISAP, or the ISAP may be a third party providing integrated sales and inventory information (e.g. a real estate multiple listing service). It is desirable that when a session is in progress, the user is not aware of any disruption in service as the user moves.

In order to meet reliability, performance and cost objectives when the number of users is large and geographically dispersed, a distributed server architecture will be necessary. In sec. 3 we describe the system architecture for mobile sales and inventory applications where the ISAP has a distributed replicated database architecture and uses the underlying PCS network for communicating with the user. As the user moves or network load and availability changes, the server interacting with the user may need to change. Thus, real mobility on the part of the user may result in the *virtual mobility* of the server. This is accomplished by means of a *service handoff*, which is broadly analogous to a PCS call handoff or user location update (i.e., registration/deregistration) procedure [1, 18], but relatively less frequent. We have previously designed a service handoff protocol [12, 13] and described the context information which must be transferred from

the old to the new server for various classes of applications, including mobile transactions.

In this paper, we draw attention to two sets of issues that arise in PISA: application-level issues related to the design and management of application databases in a mobile environment, and infrastructure-level issues that relate to the management of service profile data that is important for the flexibility and efficiency of PISA. In sec. 4 we describe how the semantics of the sales application can be exploited to provide an appropriate application database design. In sec. 5, we extend escrow techniques from traditional database management to provide a framework for running mobile transactions. Just as for the underlying PCS network, the ISAP will need to maintain user service profiles to determine what services the user needs and is authorized to obtain. A PCS network typically uses a two-level hierarchy of databases (Home Location Register and Visitor Location Register - HLR and VLR) to maintain this information, and a user location update protocol (e.g. IS-41 [1]). For the same reason as for the PCS network, we propose in sec. 6 that the ISAP also use a two-level database hierarchy for service profiles, and discuss scenarios in which the need arises for an access protocol more robust than those (e.g. IS-41 [1]) used for managing user location information in the PCS network. We present a protocol for managing the service profile databases. We end with some concluding remarks in sec. 7.

2 APPLICATION SCENARIO: MOBILE SALES AND INVENTORY

We consider a scenario in which the user is a mobile salesperson selling financial products (like insurance, bonds, etc.), or consumable products (e.g. doctor's office supplies like gloves, syringes, etc.). The ISAP is either the company the salesperson works for, whose products are being sold, or a third-party supplier of information. The salesperson uses a personal digital assistant (PDA) as a mobile database when discussing and completing sales. To avoid confusion, we will adopt the following terminology. The salesperson is also referred to as the user of the ISAP's services. The PDA or other end equipment which the salesperson uses is called the mobile or the client of the ISAP's servers. The person to whom the sale is being made is referred to as the customer.

The user visits numerous customer offices during the course of a day and discusses their requirements, shows images of products, initiates new orders or

queries about the status of previous orders, etc., using the PDA. The PDA is capable of doing these functions using either a wireless or a wireline link to the ISAP's servers. The mobile database contains customer records as well as information regarding policies, prices and availability of the product being sold. The time available to the salesperson for meeting with the customer may be extremely limited [21]. In order to ensure that this time is utilized most effectively, the mobile database must have current information available about dynamically varying quantities such as inventory levels, delivery dates, etc. The mobile database is periodically updated from the ISAP's database. The user has a service profile stored with the ISAP which ensures that the mobile database is updated at appropriate times with the appropriate information. Orders or queries placed by the salesperson are transmitted to the ISAP database.

Observe that this is not a far-fetched scenario. Mobile sales applications are already being tested and marketed [22, 21]. Moreover, if current market projections are realized, this scenario will not be rare in the future. PDAs are forecasted to become a commonplace business accessory, with sales of PDAs exceeding 3.6 million units by 1997 [14].

3 SYSTEM ARCHITECTURE

We explore different architectural alternatives for the ISAP.

A centralized architecture: In the initial phases of the service offering, the number of users (salespersons) is likely to be relatively small. If the users are also geographically localized, and move infrequently it may be sufficient for the ISAP to maintain a *centralized* architecture, i.e., to store and process information at a central site. The ISAP and users can then communicate with each other via a PCS network, possibly owned by a separate PCS service provider, by initiating a PCS call.

Multiple independent servers architecture: Current market and technology trends project rapid increases in the number of mobile users with intelligent mobile computing and communication devices [14], and ISAPs offering nationwide, even global, services involving multimedia interactions. If these trends evolve as projected, the centralized ISAP system architecture will become infeasible, largely because of the computing and communication bottleneck at the central server. Initially, this could be addressed by installing a *centralized parallel*

lel server at the central site, i.e., a logically centralized server which physically consists of several processors working in parallel. However, as the user base becomes more geographically dispersed, the communication costs and delays involved in interacting with users from a central server site will become unacceptable.

For some applications, it will suffice to address the communication concerns by installing *multiple independent servers* at several geographically distributed sites, and connecting each server independently to the PCS network. For example, if the information being provided to users is itself geographically localized, as in details about customers in a geographical region, it is likely that most users of that information will also be localized, so communication overheads will not be serious. Note however, that inventory or sales data might not be easily partitionable geographically, so having independent servers is not a good solution.

Distributed server architecture: In general, mobile users will desire access to private and corporate databases which cannot be simply geographically partitioned into locally-accessed portions. It will then be necessary to use a *distributed server* architecture, where the information is (partially) replicated across multiple interconnected servers but the system functions as a single logical information base. For the remainder of this paper we assume a distributed server architecture.

There are several possible ways of interconnecting the servers, e.g. using a private ISAP network attached to the PCS network via a gateway, or using the PCS network itself as the inter-server communication backbone. The geographical coverage area for the information service is partitioned into *service areas*, analogous to PCS registration areas. It is likely that a service area will cover several PCS registration areas. Each service area is served by a single information server, called the *local server*, analogous to the PCS network's Mobile Switching Center (MSC) or VLR database. The connection between the ISAP and the mobile user can be set up by either side dialing the other's non-geographic telephone number.

The most basic support¹ required by the ISAP from the PCS network is that the physical connection between the user and the ISAP be maintained without interruption during a session as the user moves. Two key functions needed for this support are to *locate the mobile user* and to *perform a physical connection*

¹The PCS network can also provide additional services such as billing etc, which are outside the scope of this paper.

transfer as the user moves. Protocols for performing the physical connection transfer function in a store-and-forward-packet-switched network have been proposed by Keeton et al. [15]. However, we note that both functions above are already provided by the user location facilities and call handoff mechanisms specified in PCS standards. Thus we will assume that the following levels of protocols are already provided by the PCS network:

1. A call handoff protocol, similar to that specified in Bellcore's WACS [3] or GSM standards [18] for physical connection transfer of the wireless link when a mobile client moves from one cell to another.
2. A user location protocol, similar to that specified in the IS-41 [1] or GSM [18] standard, for registering a mobile client in a registration area and for locating and delivering calls to the client when it moves between registration areas.

We also assume that the application is running a link-level protocol which recovers from bit errors, as well as packet losses, duplication and reordering, for both wireless and wired links. (Examples of such protocols include LAPR for wireless links and LAPM for wireline voiceband modems; see [20]).

As a mobile user moves from cell to cell but within the same service area (so that the user is in contact with the same server during the move), the PCS network can perform a physical connection transfer, i.e., keep the connection continuous with the same server, using the usual PCS call handoff procedure. The call handoff may result in errors at the physical layer of the connection, e.g. bit errors or packets being dropped, which can be recovered from by using the link level protocol.

As the user moves out of one service area into another, it is desirable that the local server at the new service area take over providing the service. This *service handoff* for the *virtual mobility* of the server is broadly analogous to the PCS call handoff procedure (except that it occurs between ISAP servers rather than PCS base stations), and also has the requirement that service appear to continue transparently without interruption.² In [12, 13], we have described protocols and capabilities required in both the ISAP and the PCS network to implement service handoffs. Briefly, a service handoff consists of a transfer of context information from the old server to the new server, followed by a physical

²Note that virtual mobility differs from *service mobility* [2], which is the ability of a user to have a consistent set of services even though the user may move.

connection transfer between the old and new servers. The context information depends upon the application in progress, but essentially informs the new server of where to pick up the session after the old server left off. For example, if the mobile user was simply reading through a file, the context information would be the name of the file and a pointer (e.g. line number or byte position) from where the new server should start sending information to the mobile.

The service handoff is initiated by an ISAP process called the *matchmaker*, which is responsible for mapping users to appropriate servers, and for setting up initially and managing the connection between the user and servers of the ISAP. (The term “matchmaker”, and some of its functionality, has been borrowed from [24]. However our notion of service handoffs, the protocols we have developed and the applications we consider are quite different [12, 13].) The matchmaker can be implemented in a centralized or a distributed manner across several ISAP servers.

It is important to note that the interactions between the mobile and server(s) only imply that a connection is continuously maintained between the mobile and the server at the *session or application* level. In particular, it is not necessary that the wireless link be continuously in use, since the client and server can exchange occasional packets as necessary. Similarly, long-running interactions need not imply that the client machine is turned on at full power all the time; almost all modern mobile client devices slip into *doze* mode, yielding very substantial decreases in power consumption.

4 DATABASE SYSTEM DESIGN

In this section we describe the design of the ISAP’s database. The design choices are motivated by the need to accommodate both stationary and mobile users. A principal aim of the design is to minimize the aspects of the database which are specific to mobile users.

4.1 Background

We first review some background on how transactions may typically be handled in distributed database environments. Readers familiar with this material may skip to the following subsection.

In a transaction processing environment, transactions can concurrently access shared (possibly replicated) data. Therefore, their execution has to be carefully controlled so that correctness is preserved: for instance, the last inventory item should not be allocated to two different transactions. The traditional notion of correctness is serializability [6], *i.e.* the effect of the interleaved operations of (concurrent) transactions is the same as that produced by a sequential execution of the transactions. The algorithms used to ensure serializability are also referred to as concurrency control protocols.

One example of a concurrency control protocol is strict two-phase locking. In this protocol, a transaction acquires a *read lock* (*write lock*) on a data item before reading (writing) that item. Two locks on a data item are conflicting if either is a write lock, and a transaction may acquire a lock only if no other transaction holds a conflicting lock. This ensures that there is only one writer, but there can be multiple readers if there is no concurrent write. Furthermore, a transaction cannot acquire any more locks after it releases a lock. This defines a two-phase execution for a transaction with respect to the locks, where first there is a *growing phase* when locks are acquired, and then there is a *shrinking phase* when all the locks are released. In strict two-phase locking, all locks can be released only when the transaction commits or aborts. This mechanism ensures that the transactions are serializable in the order in which they release locks.

The corresponding notion of correctness for replicated data (where copies of the same data item are stored at several servers) is *one-copy* serializability: the effect of the execution of a set of transactions on the replicated data is equivalent to some serial execution of those transactions on a single copy. The Available Copies Algorithm [6] extends two-phase locking to the replicated environment. In this algorithm, a site reads from its own copy of the data item by using a read lock, but writes to all replicas by obtaining write locks on the data item at those replicas. The generalization of this algorithm is the Quorum Consensus (Locking) Algorithm [8, 25, 10] where a read operation on a data item locks the data item at a *read quorum* of replicas and a write operation locks the data item at a *write quorum* of replicas. One-copy serializability is guaranteed if the read and write quorums intersect.

In a replicated system, a transaction is executed at a single replica; however locks could be obtained at several sites and updates might have to be installed at several sites at the end of the transaction. Thus a co-ordination protocol is required to ensure that the transaction commits at all sites or aborts at all sites. This co-ordination protocol is the *two-phase commit* protocol: a co-ordinator sends a prepare message to the participating replicas, upon which each replica

votes whether it can commit its unit or not. If all votes are affirmative, the co-ordinator sends a commit message to the replicas, and an abort message otherwise.

Thus, in general, replicated data management involves acquiring locks at a set of sites and executing a two-phase commit at the end of the transaction to ensure the system-wide consistent commit or abort of the transaction and the release of the locks.

4.2 Escrow techniques

We assume that the (most frequently accessed portion of the) ISAP's database is replicated across several servers. Thus we would expect the ISAP to ensure the typical correctness condition of one-copy serializability. However, the requirement of one-copy serializability has been found to be quite restrictive, so recent approaches in the literature have taken into account the semantics of the application to relax this criterion and improve transaction throughput. In this paper, we take a similar approach and provide an escrow-based algorithm that accommodates mobile users easily.

We first review how the consistency of the data in the ISAP can be preserved, without discussing how mobility impacts it. Consider the situation where a salesperson is selling items from inventory, where each instance of the item is indistinguishable from the others (e.g. the item is a medical supply item, and each instance is, say, one box of the item). For ease of exposition, consider a single sale item, m . Let $Total_m$ be a (replicated) datum in the database that indicates the total number of instances of that item in stock. As salespersons make sales of that item, the problem is to ensure that the total number of items they have sold for immediate delivery, $Sales_m$, satisfies the constraint $Sales_m \leq Total_m$ at all times. As sales orders are taken or canceled, salespersons launch transactions which update the number of instances sold, $Sales_m$. These updates will typically be made as operational updates instead of value updates, i.e., instead of reading and writing the actual value of the variable $Sales_m$, transactions will issue operations to increment or decrement it. (Operational updates can be preferred over value updates for a number of reasons relating to concurrency control, and rollback and recovery of aborted transactions [6]).

As seen in sec. 4.1, if two or more salespersons launch long-running transactions which contain update operations, a traditional concurrency control algorithm would require that the variable $Sales_m$ be locked by each transaction, so that

one transaction cannot begin until the other commits and releases the lock. In a replicated system, this further entails using a distributed protocol such as quorum locking [10] and then a two-phase commit to ensure consistency. The two-phase commit protocol has some disadvantages: (a) it requires all the participants to be available at one point of time and vote yes if the transaction has to commit; any failure by any participant results in the transaction being aborted. (b) it is a blocking protocol, *i.e.* if the co-ordinator fails during some window of time, the participants have to wait for the co-ordinator to recover before a decision to commit or abort the transaction can be made. (c) it requires at least two rounds of messages between the co-ordinator and the participants before commit or abort of the transaction. However, the idea of placing instances of the item being sold in *escrow* allows data items to be locked for small intervals of time and also avoids the two-phase commit, thereby increasing throughput.

In general, an escrowable resource item refers to a resource whose instances are indistinguishable, so that the instances can be partitioned, either among transactions or among sites in a replicated database (as we see shortly). Several escrow schemes have been proposed in the literature including transaction escrow [19], site escrow [17, 23, 5], and generalized site escrow [16]. In the following we describe transaction and site escrow as a background for the combined site and transaction escrow scheme we will be using.

Transaction escrow: The transaction escrow scheme was initially proposed [19] for access to hot-spots in a single-copy (*i.e.*, non-replicated) database. In this algorithm, a transaction executes an *escrow* operation to try to place in reserve the resources that it will (potentially) use. All successful escrow operations are logged in an *escrow log*. Before executing an escrow operation, each transaction accesses the log and sees the total escrow quantities of all uncommitted transactions. The transaction then makes a worst-case decision to determine whether it can proceed. For instance, suppose the total quantity of item m in stock is $Total_m = 100$, and the quantity sold due to committed transactions is $Sales_m = 20$. Suppose there are currently two uncommitted transactions each requesting one item. Let transaction T wishing to reserve ten items now be initiated. Since the log indicates that $Sales_m \leq 22 \leq Total_m$, T can proceed irrespective of whether the other two transactions commit or abort: the constraint is maintained in any case. Note that when transaction T executes an escrow operation, T obtains a short-term lock on the escrow log to access and update the log and releases the lock after the log has been updated. (The lock release need not wait for the commit or abort of T as in a traditional transaction execution). Thus any other transactions which access $Sales_m$ are forced to wait only for the duration of the log update operation,

rather than for the entire duration of T as would occur in a traditional scheme. This feature allows long-running transactions that contain escrow operations to run concurrently so that throughput is increased.

Site escrow: In site escrow algorithms [17, 23, 4], the total number $Total_m$ of available instances of a given item m , is partitioned across the number of sites (servers) in the system. This can be thought of as each site (as against a transaction) holding a number of instances in escrow. A transaction launched by a user runs at only one site (typically, the one closest to the user). A transaction can successfully complete at a site only if the number of instances it requires does not exceed the number of instances available in escrow at that site. Each operation of the transaction acquires a lock at the site when accessing the item, just as for a traditional locking scheme. However, this lock is different in two important respects. Firstly, the lock is *local* in the sense it applies only to the site where the operation is executing, and is designed to protect the operation from other transactions executing at that site. This contrasts with traditional replica control schemes, such as quorum locking, which would require each site to lock (a subset of) the other sites before proceeding with the operation. Secondly, *the lock can be released on successful completion of the operation*, in contrast to traditional strict two-phase locking where the lock is released only at commit time of the *transaction*. By allowing each site to deplete its own escrowed instances without consulting other sites, avoiding the distributed two-phase commit, and shrinking the interval during which items are locked, the site escrow model results in higher autonomy to sites and greater throughput.

The number of instances held in escrow at each site is adjusted to reflect the consumption of instances by the transaction only if it commits; otherwise the escrow is restored to its original state. When one site requires more instances, a *redistribution* or *reconfiguration protocol* such as the point-to-point demarcation protocol [5] or a dynamic quorum-based protocol [16] is executed, so that the site can get a portion of some other sites' unassigned instances.

As an example of a reconfiguration protocol, we describe the demarcation protocol (we will use a variant of this protocol in our design.) The demarcation protocol helps preserve distributed numerical constraints such as $A + B \leq 200$, where A and B are two data items. Assume that A is stored at site 1 and B at site 2. To preserve the constraint above, each site maintains an upper limit on the data item stored at that site: site 1 has the limit A_u for A and site 2 the limit B_u for B , such that $A_u + B_u \leq 200$. Transactions are allowed to alter A or B but cannot alter A_u or B_u . The limit values at each site are essentially the site escrow quantities: a transaction at site 1 can increment A

without consulting site 2 as long as the final value of A is less than or equal to A_u (similarly for B .) However, if a transaction wishes to increase A to a value larger than A_u , a message has to be sent to site 2 to decrease B_u appropriately. Only after this co-ordination can the transaction proceed. Several policies can be used to modify the limit values dynamically [5].

Notice that this co-ordination above does not need a two-phase commit. Site 2 can take a decision to decrease B_u without needing to consult any other site, i.e., can unilaterally do so. Site 1 decreases A_u only after a confirmation message from site 2 indicating the decrease of B_u is received. Now suppose the confirmation message from site 2 to site 1 is lost due to a link failure or a network partition. Site 1 might abort the transaction subsequently. However, the effect is that site 2 has decreased B_u . Note that the integrity constraints of the system, e.g., $A + B \leq 200$, are still maintained [5]. However, a certain number of resources in the system have become unavailable, since they cannot be accessed via site 2. In practice, a reliable underlying message transport protocol is needed.

Site-transaction escrow: The two escrow schemes described above can also be combined [16]. Thus the total number $Total_m$ of available instances of item m is partitioned across sites, and in addition, each transaction at a site uses a transaction escrow scheme to allocate and deallocate resources at that site. Once again, a reconfiguration protocol is used to transfer resource instances between sites as necessary.

The site-transaction escrow scheme provides an elegant and efficient replica control mechanism for partitionable resources, and allows sites to make allocation decisions locally as far as possible. This technique is desirable in the mobile environment due to the following reasons:

1. A mobile is usually powered by a limited power source. Suppose a mobile has established a session with a server and is trying to allocate resources. If that server could possibly allocate the resources locally, this would enable quick response to the mobile and hence less power is consumed while idling.
2. When performing service handoffs (as seen in Sec. 5), the escrow model permits lesser context information to be transferred than when a traditional concurrency/replica control protocol is used. This results in quicker service handoffs and savings in cost.
3. The wireless bandwidth between the mobile and the ISAP server is limited. Thus instead of remaining in contact with a server at all times, it might

be desirable for the mobile to itself escrow some instances and allocate them locally. The site-transaction escrow scheme permits this alternative naturally.

We assume therefore that the ISAP servers use a site-transaction escrow scheme for maintaining inventory information, using a reconfiguration mechanism such as the demarcation protocol.

5 MOBILE SALES TRANSACTIONS

We now consider how mobile sales transactions can be handled given our system architecture and database design outlined above. It turns out that site-transaction escrow methods, such as used in our design, are particularly appropriate for mobility.

We first consider the situation referred to as *discrete mobility*. For instance, consider the situation in which a user makes some sales at one service area, completes all transactions, closes the session, disconnects from the PCS network and moves to another service area. The user can now start another session with the local server of the new service area and make further sales. The sales transactions at the new service area can simply operate on the escrow values stored at the new server. The decision as to whether any operation performed by the transaction is safe or unsafe can most likely be made by the local server (and if unsafe and solvable, the server can do a reconfiguration operation to permit the operation). This scheme does not represent any change to the underlying protocols to accommodate mobility.

Now consider the situation of *continuous mobility* in which the user is running a long transaction involving the allocation of multiple inventory items, and the user moves between service areas while the transaction is running. (Henceforth, unless otherwise stated, we deal only with transactions which allocate resources - the typical sales scenario. The discussion below is easily generalized to cases where transactions can have both allocation and deallocation of resources.) In our model, we assume that a service handoff occurs, so that the user starts communicating with the local server of the new service area. The service handoff protocol [12, 13] maintains continuity of the physical communication link between the user and the ISAP while the handoff occurs. However, before the physical connection transfer can actually be carried out, the context of the in-

teractions of the user with the ISAP needs to be transferred from the old to the new server.

Let us study the context information transfer of the handoff in greater detail. Essentially, a transaction involves numerous operational updates. When the user moves to the new service area, the current operation in progress at the old server is completed, and then the context of the transaction is transferred to the new server. (Observe here that the mobile can continue interacting with the old server while the context is being transferred. Only after the context is transferred does the mobile start interacting with the new server.) If a traditional locking scheme had been used for concurrency control, the context information would include the set of locks held by the transaction, along with the transaction id. Additional information relating to replica control would also need to be transferred. For instance, suppose a pessimistic quorum consensus protocol [10] with operational updates is used. In this protocol, it is not necessary that at any point of time, a server i know of the updates done at all the other servers. It is necessary though that when a transaction wishes to execute at server i , server i is brought up to date of all the updates done at the other servers. Thus server, i , before initiating an operation o of a transaction T , locks the data items accessed by o at a set of *quorum* servers. The quorum servers send the committed (timestamped) updates that they know of along with the lock grant response to i . On receiving the lock grant responses, server i merges in timestamp order the committed updates it has obtained from the quorum servers with the set of committed updates it knows about. If the quorum is a majority of servers in the system, one can ensure that at this stage [10], server i is up to date of all the committed updates in the system and also that since the lock for o was acquired, there is no other conflicting operation executing in the system. Based on the updated state, i determines the update, u_T . u_T is then appended to an *intentions list* of uncommitted updates for the transaction T . When T is ready to commit, a two-phase commit is executed across all the quorum servers of its operations, and if the decision is to commit, the intentions list is added to the list of committed updates at i and the quorum servers, otherwise the list is discarded. Thus, if this protocol is used, the context transferred from the old server to the new server would also include (a) the list of updates seen at the old server before the most recent operation in T was executed, (b) the intentions list for T , and (c) the list of quorum servers for each operation in T executed so far. The new server will have to first incorporate the list (a) into its state, set up (b) as an intentions list for T , update the lock table to remember the locks that T has already acquired, and store the list of quorum servers (so that they can participate in a two-phase commit when T is ready to commit). Using site escrow methods makes this context transfer and set-up much easier.

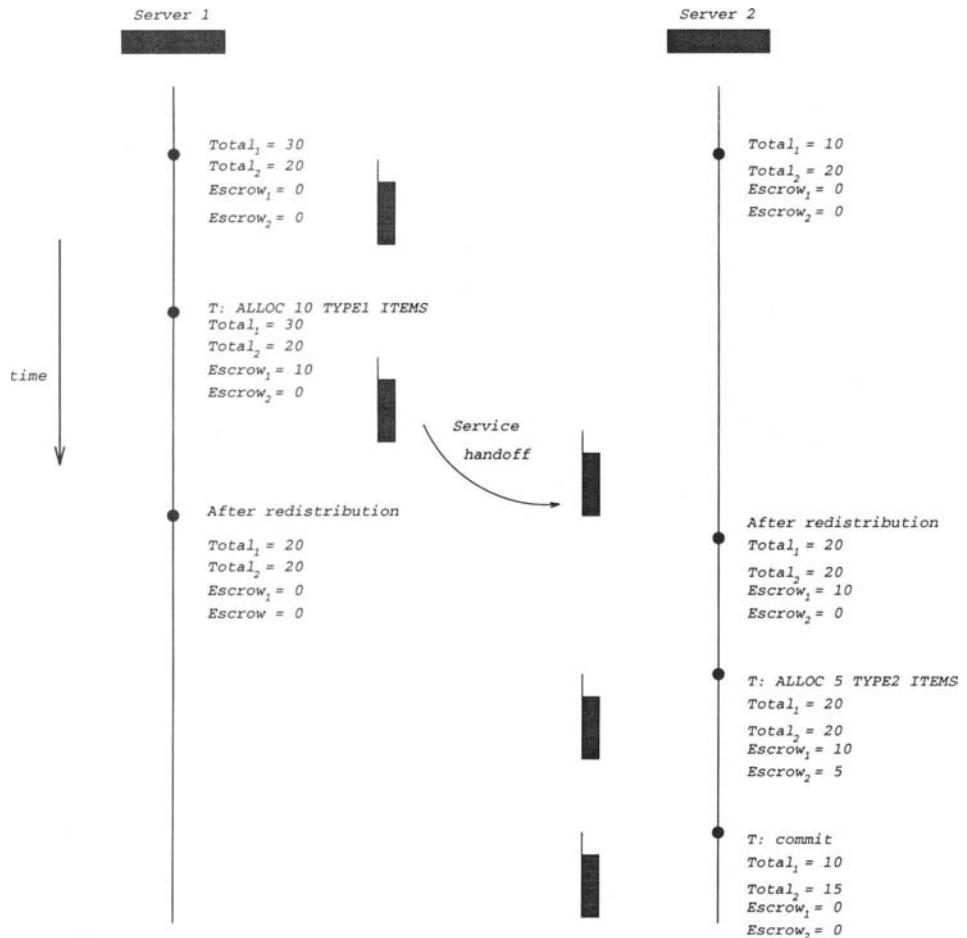


Figure 1 A mobile sales transaction example

In our design using the site escrow method, decisions in the sales transaction to allocate resources are made locally (as far as possible). So quorum information as described earlier does not have to be maintained or transferred as context. Furthermore, since sites use transaction escrow locally, locks are released after the completion of each operation. So no lock information needs to be transferred to the new server. In addition, suppose the operations in the sales transaction deal only with escrowable resource types. The new server

does not need information about the operations already performed at the old server, since the old server made its decisions based on its locally escrowed resources. Thus it is possible that the context transferred is only the information about which operation in the transaction is to be run next at the new server. However, when the transaction commits, a two-phase commit would also be required between the old server and the new server since part of the transaction has been run at the old server and the rest at the new server. (Note that the two-phase commit might be between several servers if the user moves between several service areas during the duration of the transaction, but for simplicity, we are considering only two servers here.) The benefit of the escrow idea in the case of a salesperson who does not move is that most of the time, a two-phase commit would not be required *at the end of the transaction* since requests for resources might be satisfied locally. If the salesperson moves around a lot between service areas, then a two-phase commit is almost always required at the end of the transaction, and this is undesirable.

To account for this problem, we associate a reconfiguration of item instances with a service handoff. Recall that a reconfiguration protocol is provided as part of the site escrow framework. On a service handoff, a reconfiguration of instances is performed as follows. The total number of instances at the old server is decremented by the number of instances allocated by the transaction, and the total number of instances at the new server is incremented by the same amount. Furthermore, these newly acquired instances are entered as having been escrowed by the transaction at the new server. Thus, the reconfiguration protocol is invoked to transfer the required number of instances from the old to the new server. If the transaction commits at the new server, the new server's total is decremented accordingly; if the transaction aborts, the total is not modified. This scheme thereby eliminates the need for the two-phase commit protocol at the end of the transaction, allowing the transaction to perform a simple one-phase commit at its current server. The context information which needs to be transferred during service handoff, with the site escrow method, thus includes only the transaction id and the intentions list already done at the old server. The reconfiguration protocol can be performed independently of the context information transfer - the only requirement is that the reconfiguration(s) complete before the transaction commits.

The reconfiguration itself need not involve a two-phase commit between the old and new servers similar to the demarcation protocol. However, in the absence of a two-phase commit, resources could become unavailable as seen in the demarcation protocol in sec. 4.2. Thus, there is a tradeoff between having a pair-wise two-phase commit between the old and new servers and the possibility of losing some resources in the absence of the two-phase commit.

Notice however, that if we do include a pair-wise two-phase commit between the old and new servers as part of the reconfiguration, the two-phase commit is not necessarily at the end of the transaction : we overlap it with the rest of the processing in the transaction. Furthermore, by employing pair-wise two-phase commits, we do not require all servers that participate in the transaction to be simultaneously available - this improves the reliability of the system.

An example of a mobile sales transaction is shown in Figure 1. There are two types of resource items, and $Total_1$ and $Total_2$ at each server indicates the total number of instances available there. $Escrow_1$ and $Escrow_2$ indicate the number of instances escrowed by uncommitted transactions at each site. The mobile allocates 10 items of type 1 at server 1 and then moves from server 1 to server 2. The reconfiguration protocol updates $Total_1$ and $Escrow_1$ at both the sites. The mobile submits another operation to allocate 5 items of type 2 as part of the same transaction (this could potentially have been run in parallel with the reconfiguration), and then commits.

We now discuss disconnections in more detail. Either the mobile could voluntarily disconnect from the ISAP (as in discrete mobility), or the mobile could be disconnected due to a failure in communications or of a server. In the former case, the salesperson could desire to continue selling items while being disconnected. It is then necessary that the salesperson have some idea of how many resource instances of each kind he/she expects to sell. The salesperson can allocate that many resources from the server and "cache" those resources at the mobile before disconnecting. The mobile could thereby continue to sell items by using transaction escrow on the resources that it had "cached" from the server. At some point, when the mobile reconnects, the resources that are left over could be deallocated. (Business policies would have to determine a maximum on the number of resources that can be cached, since the ISAP could quickly run out of resources even if they were not being sold: all salespersons might cache resources and not be able to sell them.) Thus the escrow model is very suitable for voluntary disconnections and local caching of resources. The second case of disconnection is when the server fails or there is a communications failure. It is then possible that a lock is held by a transaction running on a mobile, or some items have been escrowed at a server on behalf of an uncommitted transaction. Some timeout mechanism would have to be used in either case to abort the transaction and free up the resources. If the reconfiguration protocol fails during a service handoff, it is retried until the commit point of the transaction or until the timeout on escrowed resources expires at the other server, at which point the transaction is aborted.

Therefore, by using the ideas of escrow, service handoffs and reconfiguration of resources, we have provided a clean transaction framework for performing mobile sales transactions.

6 MAINTAINING SERVICE PROFILES

We now discuss the need for user service profiles and how they can be maintained.

Just as the PCS network maintains user profiles for PCS users, to determine what communication services the user needs and is authorized to obtain, it is likely that the ISAP will also need to maintain service profiles i.e., what are the information service requirements and access rights of the user. For instance, in our application, the mobile database contains client records as well as information regarding policies, prices and availability of the product being sold. It is desirable that the mobile database have reasonably current information available about quantities such as inventory levels, etc. The service profile would therefore indicate that the server should initiate a transaction which is to be run on the mobile database at appropriate instances of time, (say hourly or in response to specific market changes), to update the mobile inventory. Such a transaction could be an operational summary of the items consumed at all the sites since the last such update (such as, x instances of item 1 and y of item 2 were consumed, etc.).

In a PCS network, a two-level hierarchy of databases is used to store profiles, with the HLR at one level connected to several VLRs at the lower level. Each VLR serves one or more PCS registration areas (and MSCs). When a user moves between registration areas served by the same VLR, only the VLR needs to be notified that the user has moved. Thus the two-level database scheme reduces the signalling network traffic and database load at the HLR, so helping to prevent the HLR from becoming a performance bottleneck. For the same reasons as for the PCS network, we propose that the ISAP store the service profiles in a similar logical two-level hierarchy, using a Home Service Database (HSD) and Visitor Service Databases (VSD). With each ISAP server is associated a VSD. When the ISAP detects that the user has moved into a service area, the associated VSD is updated with information from the HSD about the user's service profile. The server in the user's current service area can use this profile to determine what interactions are required with the user, and when.

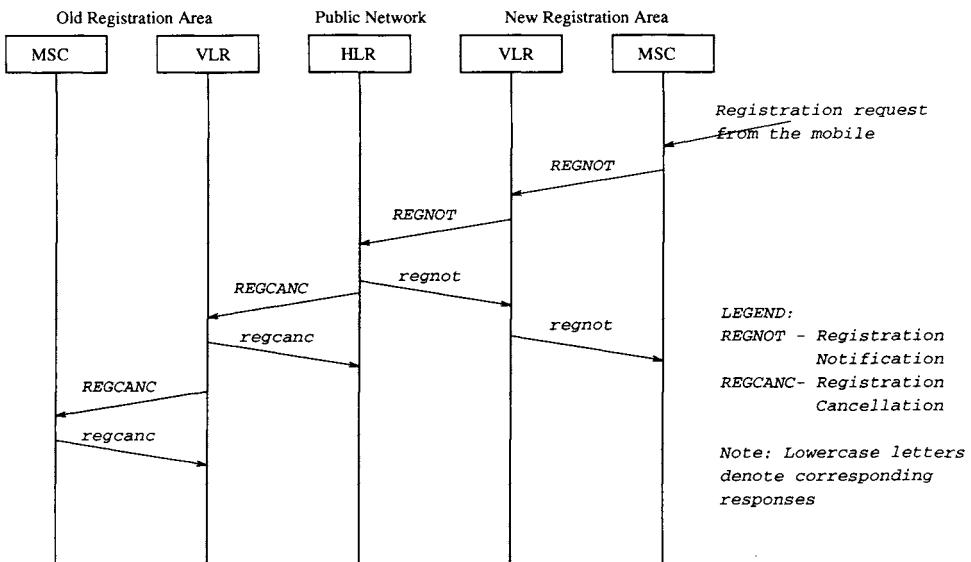


Figure 2 Signalling message flow in the IS-41 protocol

The use of a two-level hierarchy of profile databases entails a protocol for managing them. For instance, when a user moves into a new service area, the new VSD needs to obtain the user's service profile, and it may be necessary to delete the profile stored in the user's old VSD. This is analogous to a PCS location update (registration/deregistration) procedure. Therefore, it would seem that one could use a PCS user location strategy, such as that specified in the IS-41 or GSM standards, for service area registration and deregistration. However, the semantics of most PCS user location protocols (see [11] for a survey) do not ensure that a user is registered in the visited database of *exactly one* registration area at any given time, which can lead to race conditions.

As an example, consider the location update procedure in the IS-41 protocol [1], restricting attention to the databases involved. Suppose the user moves between regions served by different VLRs (see Fig. 2). The new VLR is notified (by the new MSC) that the user has moved into a registration area served by that VLR. The new VLR sends a registration notification message (abbreviated "REGNOT") to the user's HLR. The HLR is updated to reflect the new VLR as the user's serving VLR. The HLR sends a confirmation message (called "regnot") to the new VLR, and then sends a registration cancellation message

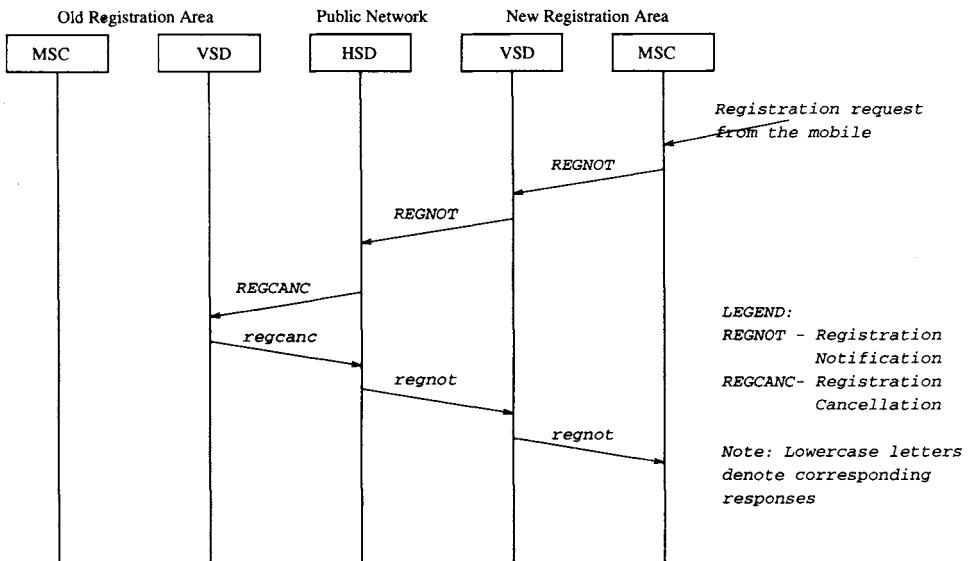


Figure 3 Message flow for a service profile management protocol, based upon a modified version of the IS-41 protocol

("REGCANC") to the old VLR. The new VLR completes the registration of the user after it receives the "regnot" message. The old VLR deletes the user's registration after it receives the "REGCANC" message, and sends a confirmation message ("regcanc") to the HLR. With this protocol, there exists a race condition between the arrival of the "regnot" message at the new VLR and the arrival of the "REGCANC" message at the old VLR, so it is possible that for some finite time the user is registered in both the VLRs.

The negative effects of race conditions during user location updates are not likely to be serious; for instance, a call to the user might not be completed during the race interval, and the caller might get a busy signal. In contrast, depending upon the application in question, race conditions during service handoffs could be serious. For instance, if a protocol similar to IS-41 is used to update VSDs, it is possible that when a user moves two VSDs contain the user's profile. In our mobile sales application example, the VSD alerts the server, at the time specified in the user's profile, to initiate the appropriate transaction on the client database. If a protocol similar to IS-41 is used to manage the VSDs when a user moves from one service area to another (whether a call is in progress or

not), the race condition described above can occur, resulting in both servers running the same update transactions twice and the mobile database having inconsistent information.

We propose one possible solution for preventing race conditions. First, note that the race condition we have described in IS-41 can result in the user being registered in two VLRs, but it can also result in the user being “active” in neither VLR. The latter situation arises because although the new VLR records the user’s location before it sends the “REGNOT” registration message to the HLR, it does not consider the user “active” until it has received the “regnot” confirmation message from the HLR. Thus if the “REGCANC” cancellation message arrives at the old VLR before the “regnot” message arrives at the new VLR, the user will be active in neither. In other words, the protocol as it stands allows the user, at different times, to be active in zero, one or two databases.

We propose using a simple modification of the IS-41 protocol for managing service profile databases, as follows (see Fig. 3). Upon receiving a “REGNOT” message from a the new VSD, the HSD first sends a “REGCANC” cancellation message to the old VSD to deactivate the user’s profile. Upon receipt of the “REGCANC” message the old VSD stops all further operations related to the user’s profile. The HSD then waits until it receives a “regcanc” confirmation message from the old VSD before sending a “regnot” registration confirmation message to the new VSD. This sequence of messages removes one ambiguity in the protocol, by ensuring that the user can never be active in two VSDs. Note that in principle the user’s service profile is transferred to the new VSD (either from the old VSD or the HSD, as appropriate) after this activation procedure is complete; in practice, the service profile may arrive at the new VSD along with the “regnot” message.

The modified protocol also has the effect of ensuring that there is a small interval during every service handoff when the user is active in neither VSD. Consider again our example scenario, where some transaction T is initiated based upon the user’s profile at a fixed time, say t . Suppose the old VSD receives the “REGCANC” message and deactivates the user at some time $t_o < t$, where t_o is the time according to the local clock at the old VSD. Suppose the user becomes active at the new VSD at time t_n , where t_n is the local time at the new VSD. As it stands, if $t_n > t$, the new VSD will not alert the new server to perform transaction T . To prevent this, the old VSD timestamps the user’s profile with time t_o as its last operation upon the profile. (Alternatively, the old VSD can send this information to the new VSD via a separate signalling message.) When the new VSD receives the profile, if $t_n \geq t_o$ the new VSD initiates all transactions which fall in the interval $[t_o, t_n]$, including transaction

T . Otherwise $t_n < t_o$ and the new VSD does not initiate any transactions based upon the user's profile until the new VSD's local clock exceeds t_o . This convention ensures that the user being active in neither VSD does not cause some transactions to be dropped, and also does not require that the clocks of the two VSDs be synchronized.

7 CONCLUSIONS

We have presented a distributed replicated server architecture for delivery of PISA, and identified the notion of service handoffs in this architecture. We have presented a database system design, based on a site escrow method, suitable for sales and inventory applications supporting both stationary and mobile users. We have discussed how the site escrow idea provides a simple method for performing service handoffs when a mobile user moves from one service area to another. Finally, we discussed a two-level hierarchy of databases for maintaining service profiles for salespersons, and designed an appropriate update protocol for it. We are currently investigating some of the issues discussed in this paper further. We have previously discussed [12, 13] how mobile transactions which access distinguishable instances, for which site escrow methods are typically not appropriate, can be supported. We are currently exploring the tradeoffs of combining such transactions with those accessing indistinguishable items.

Acknowledgements

We thank A. Grinberg, D. Hakim, M. Kramer, R. Wolff, and T. Whitaker for their helpful comments on an earlier version of parts of this paper.

REFERENCES

- [1] "Cellular radiotelecommunications intersystem operations, Rev. B", EIA/TIA, July, 1991.
- [2] "Feature description and functional analysis of Personal Communications Services (PCS) Capabilities", Bellcore Special Report, SR-TSV-00230, Apr. 1992.

- [3] "Generic criteria for Version 0.1 Wireless Access Communications Systems (WACS)", Bellcore Technical Advisory, TA-NWT-001313, Issue 1, July 1992.
- [4] G. Alonso and A. El Abbadi, "Partitioned data objects in distributed databases", University of California, Santa Barbara, Technical Report TRCS-93-06, 1993.
- [5] D. Barbara and H. Garcia-Molina, "The Demarcation Protocol : A technique for maintaining arithmetic constraints in distributed database systems", Proceedings of International Conference on Extending Data Base Technology, 1992.
- [6] P.A. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems", Addison Wesley Publishing Company, 1987.
- [7] D. Ferrari, A. Banerjea and H. Zhang, "Network support for multimedia - a discussion of the Tenet approach", Technical Report TR-92-072, Intl. Comp. Sci. Inst., Berkeley, CA, Nov. 92.
- [8] D.K. Gifford, "Weighted voting for replicated data", Proceedings of the Seventh ACM Symposium on Operating Systems Principles, pages 150–159, 1979.
- [9] J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques", Morgan Kaufmann, 1993.
- [10] M. P. Herlihy, "Concurrency vs. availability: Atomicity mechanisms for replicated data", ACM TOCS, 5 (3), 249-274, Aug. 1987
- [11] R. Jain, "A survey of user location strategies in personal communications services systems", Submitted for publication, 1993.
- [12] R. Jain and N. Krishnakumar, "Network Support for Personal Information Services to PCS Users", *IEEE Conf. Networks for Pers. Comm. (NPC)*, Long Branch, NJ, Mar. 1994.
- [13] R. Jain and N. Krishnakumar, "Service handoffs and virtual mobility for delivery of personal information services to mobile users", Submitted for publication, 1994.
- [14] J. Jerney, "A conversation with Dataquest's Jerry Purdy", Pen-based computing, pp. 7-8, Aug./Sep., 1993.

- [15] K. Keeton, B. A. Mah, S. Seshan, R. H. Katz, D. Ferrari, "Providing connection-oriented network services to mobile hosts", Proc. USENIX Symp. Mobile and Location-Independent Computing, pp. 83-102, Aug. 93.
- [16] N. Krishnakumar and A. Bernstein, "High throughput escrow algorithms for replicated databases", Proceedings of the 18th Intl. Conf. on Very Large Data Bases, 175-186, Aug. 1992
- [17] A. Kumar and M. Stonebraker, Semantics based transaction management techniques for replicated data, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 379-388, 1988.
- [18] M. Mouly and M. - B. Pautet, "The GSM System for Mobile Communications", 49 rue Louise Bruneau, Palaiseau, France, 701 pp., 1992.
- [19] P. E. O'Neil, "The escrow transactional model", ACM TODS, 11 (4), 405-430, Dec. 1986.
- [20] A. R. Noerpel, L. F. Chang and D. J. Harasty, "Radio link access procedure for a wireless access communications system", *Proc. Intl. Conf. Comm.*, May, 1994.
- [21] V. Schnee, "An excellent adventure", *Wireless*, pp. 40-43, Mar./Apr., 1994.
- [22] J. Schwartz, "Upgrade lets salespeople share data", *Comm. Week*, pp. 47-48, May 24, 1994.
- [23] N. Soparkar and A. Silberschatz, "Data-value partitioning and virtual messages", *Proceedings of the 9th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pp. 357-367, 1990.
- [24] C. Tait and D. Duchamp, "An efficient variable-consistency replicated file service", *Proc. USENIX File System Workshop*, May 92.
- [25] R.H. Thomas, "A majority consensus approach to concurrency control for multiple copy databases", *ACM Transactions on Database Systems*, 4(2):180-209, Jun. 1979.

STRATEGIES FOR QUERY PROCESSING IN MOBILE COMPUTING

Masahiko Tsukamoto,
Rieko Kadobayashi* and Shojiro Nishio

*Dept. of Information Systems Engineering,
Faculty of Engineering, Osaka University
2-1 Yamadaoka, Suita, Osaka 565, Japan*

* *ATR Media Integration & Communications Research Laboratories,
Seika-cho, Soraku-gun, Kyoto 619-02, Japan*

ABSTRACT

In this paper, we discuss several strategies for efficient processing of three types of queries concerning mobile hosts; (1) queries to obtain the location of a mobile host (*location queries*), (2) queries to determine whether the mobile host is currently active (*existence queries*), and (3) queries to obtain a piece of information from a mobile host (*data queries*). Extracting five fundamental strategies from those usually employed in mobile communication protocols, we shall discuss their features and then compare their performance. These five strategies are: *single broadcast notification* (SBN), *double broadcast notification* (WBN), *broadcast query forwarding* (BQF), *single default notification* (SDN), and *double default notification* (WDN). We show that the optimal strategy varies according to network conditions such as network topology, network scale, migration rate, and query occurrence rate.

1 INTRODUCTION

Currently, one of the most important and attractive research issues in computer networks is the development of *ubiquitously-accessed networks*, where users can access computer resources at anytime from anywhere through movable computers. A substantial amount of research has already been conducted on how to support *mobile communications* in existing network environments (see, e.g., [4], [8], [17], [18], [19], [20], and [22]). These approaches were directed at the

realization of low cost location management of *mobile hosts*. In addition to the *packet transmission* required in conventional mobile communication protocols, the mobile computing environments currently under development must resolve several data management issues such as query processing (see, e.g., [1], [6], [9], [10], [11], and [14]).

Among queries in a mobile computing environment, we consider queries called *location sensitive queries* which are used (1) to obtain the location of a mobile host, (2) to determine the existence of a mobile host, and (3) to obtain data stored in a mobile host (e.g., location dependent information such as the sensor value set on a mobile host). The strategies for processing such queries are considered to be rather different from those used for mobile communication protocols, although the key tasks for both are the management of location information of each mobile host.

In this paper, we first review the strategies currently used in mobile communication protocols, and then, extracting the fundamental ideas of these strategies, the following five strategies will be proposed for processing location sensitive queries: *single broadcast notification* (SBN), *double broadcast notification* (WBN), *broadcast query forwarding* (BQF), *single default notification* (SDN), and *double default notification* (WDN). For each of these five strategies, we shall discuss their features and then compare their performance. As a result, we show that the optimal strategy among them varies according to such conditions as network topology, network scale, and migration rate.

The paper is organized as follows. First in Section 2, we extract five basic strategies currently employed in mobile communication protocols. Then we apply these strategies to process location sensitive queries in Section 3. In Section 4, we evaluate these strategies and demonstrate that the optimal strategy changes according to several network conditions. Finally, we summarize the paper and discuss possible future work in Section 5.

2 TECHNIQUES USED IN MOBILE COMMUNICATION PROTOCOLS

In this section, we study mobile communication protocols in regard to their ability to process location sensitive queries in Section 3. Here we use the model illustrated in Figure 1 as an example. A *mobile host* is a system which is capable of moving across wireless *cells*. A *router* is a system which forwards packets

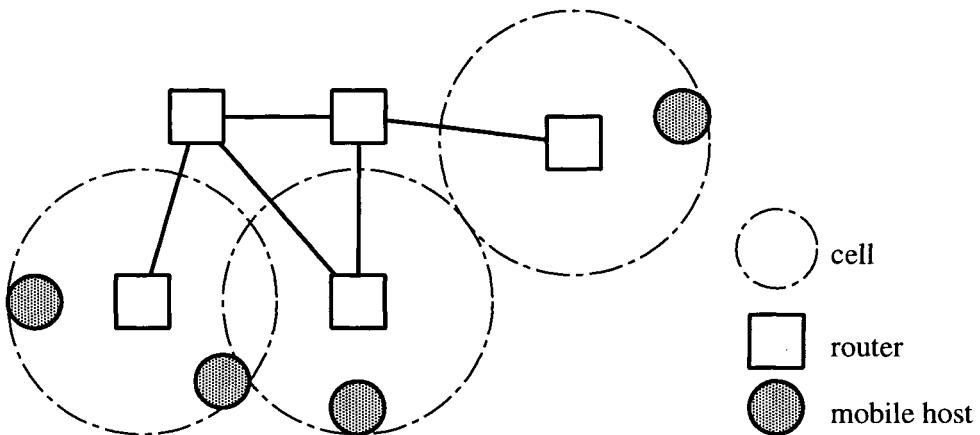


Figure 1 An example of a communication network.

from/to mobile hosts. A wireless cell is managed by a router, and the router can exchange packets with any mobile host located in its cell. Mobile communication protocols are concerned with correctly and efficiently forwarding packets to the destination mobile host.

Many researchers have proposed various protocols to support mobile communication. Teraoka et al. [19], Perkins and Bhagwat [17], and Wada et al. [22] have proposed protocols for mobile hosts in *IP networks*. Ioannidis et al. [8] have proposed protocols for small IP networks such as campus networks. In *OSI network* environments, the existing routing protocol, i.e., the *IS-IS protocol* [13], can deal with *host migration* within an area. Furthermore, several protocols for *inter-area migration* have been proposed by Carlberg [4] and Tsukamoto and Tanaka [20], and an effective protocol for *intra-area migration* has been also proposed by Tanaka and Tsukamoto [18].

Careful observation of these protocols allows us to determine that they employ the following three fundamental strategies in an integrated manner:

Broadcast Notification (BN): A mobile host's location information is broadcast throughout the network on each migration. According to this information, data packets for the mobile host are forwarded to the current location. the IS-IS protocol and OSPF [16] are examples of BN.

Default Forwarding (DF): A *default router* exists for each mobile host. The *default router* maintains that host's location information. A *local router* is located in the same network as the mobile host and notifies the default router of the mobile host's current location. Data packets to the mobile host are first forwarded to the default router and then forwarded to the local router. DF is currently the most common strategy used to support mobile hosts and is employed in [18], [19], [20], and [22].

Broadcast Query (BQ): On receiving a data packet addressed to a mobile host, the router broadcasts a query packet to all the other routers in order to forward the data packet according to the reply. BQ is used in [8].

In addition to the above three strategies, the following two strategies can be used to support mobile hosts:

Default Query (DQ): Like DF above, there exists a default router for each mobile host. The default router is informed of the mobile host's current location. On receiving a data packet addressed to a mobile host, the router asks the default router for the location of the mobile host, and then forwards the data packet to the location identified in the reply.

Broadcast Forwarding (BF): This strategy is based on a broadcast mechanism. The router receives data packets and then broadcasts them to the entire network. This strategy has been widely used in packet radio networks [7].

These five strategies can be categorized from several viewpoints. In BQ and DQ, as soon as a router receives a packet, a query packet is sent for obtaining the destination mobile host's location. We call these strategies *query-based strategies*. On the other hand, in BF and DF, as soon as a router receives a packet, the packet is forwarded to other systems without first using a query packet to obtain the destination mobile host's location. We call these strategies *forwarding-based strategies*. In DF and DQ, there is a default router for each mobile host to manage that host's location information. We call these strategies *default-based strategies*. In BN, BF, and BQ, a special system for mobile host location information management does not exist. In this case, broadcast is used for notification, forwarding, or querying. We call these strategies *broadcast-based strategies*.

Among many possible parameters which affect the network environment, the three parameters, *amount of traffic*, *migration rate*, and *packet size*, are considered to have a strong effect on the performance of the above protocols. If

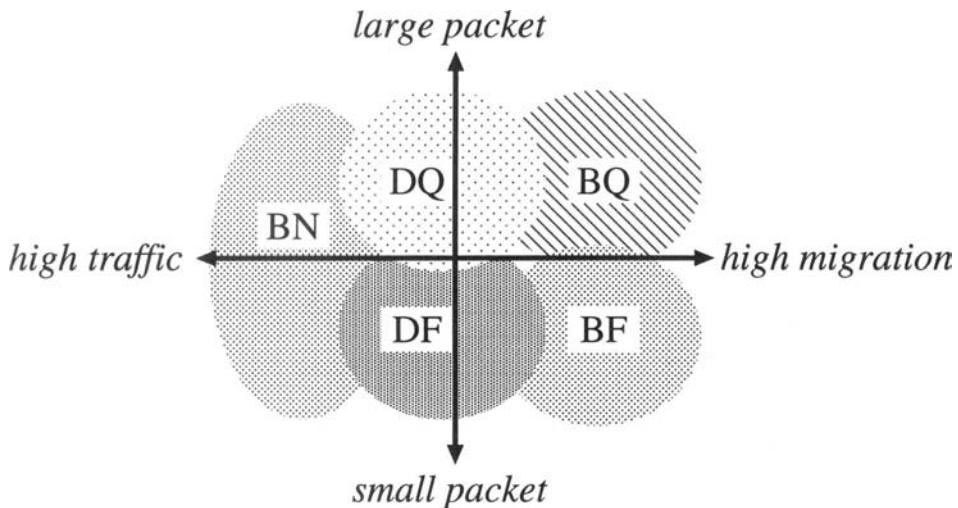


Figure 2 Best strategies for mobile communication.

high traffic and infrequent migration are assumed, BN is considered to be the best strategy for bandwidth utilization regardless of the packet size because BN provides faster convergence. If the opposite case is assumed, BF or BQ is better than the others because no location information is sent on migration. Otherwise, if traffic as well as migration rate are moderate in comparison to the abovementioned two extreme cases, then DF and DQ are effective at reducing the total traffic because packets are not broadcast on both migration of mobile hosts and forwarding of data packets.

If data packets are large, query-based strategies DQ and BQ are better than the forwarding-based strategies DF and BF, respectively. Caching of mobile information in each router can reduce the time it takes a query to complete and hence results in effective use of bandwidth. If the data packets are small, it is better to choose the forwarding-based strategies.

Generally, default-based strategies DF and DQ are more scalable than broadcast-based strategies BF and BQ, respectively. If we consider an n -grid topology and evaluate the cost by total traffic caused by control packets and data packets, the cost of default-based strategies grows as an order of $O(n)$, while the cost of broadcast-based strategies grows as an order of $O(n^2)$. However, it is

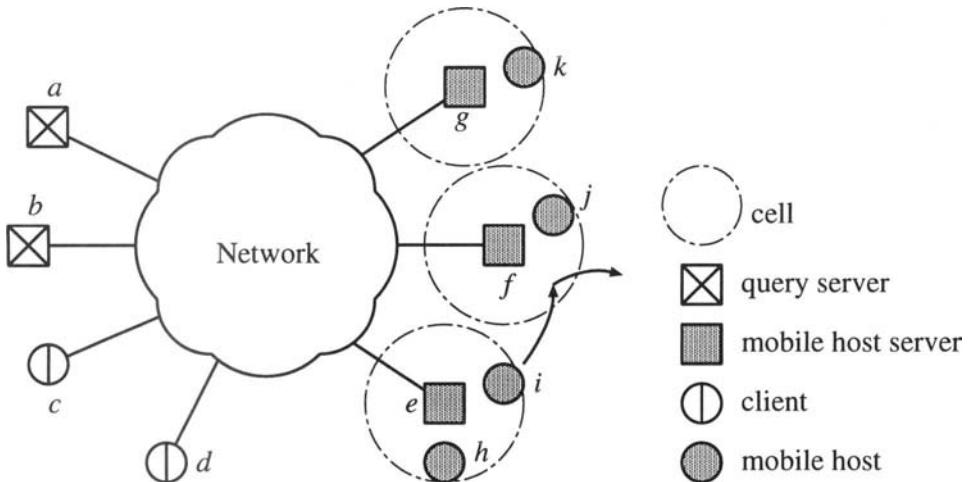


Figure 3 An example of a mobile database network.

possible that broadcast-based strategies reduce the total traffic when used in small or low cost networks.

Therefore, the best strategies to support mobile communications depend on the network conditions. Figure 2 graphically shows which strategy is superior. Most of these strategies are qualitatively analyzed in [21].

3 QUERY PROCESSING FOR LOCATION SENSITIVE QUERIES

In this section, the mobile database network model assumed in this paper is described. Then, we discuss query processing functions essential to mobile communication environments using references to the features of mobile communication protocols described in Section 2. Finally, we propose several fundamental strategies for the query processing.

3.1 Model

A *mobile host*, which can move across *cells*, can keep pieces of information that may be accessed by other systems. A *client* is a system which invokes queries about mobile hosts. A *query server* is a system which can handle mobile query processing according to the requests from clients. Each client is directly or indirectly connected to at least one query server through a global network. A *mobile host server* is a system which can directly communicate with all mobile hosts in the same cell through its wireless interface, and furthermore can communicate with all query servers through global networks. In Figure 3, we show an example of a network which includes clients, query servers, mobile host servers, and mobile hosts.

As soon as a query server receives a query from a client, it starts to process the query. In its processing, the query server exchanges packets with any mobile host servers, mobile hosts and other systems as necessary. In this model, we have the following assumptions:

- The existence of each mobile host is known to all mobile host servers located in its wireless cell. This assumption can be realized by mechanisms such as *beaconing*, which is the periodic exchange of *beacon* packets. Such mechanisms are provided by several network protocols such as the *ES-IS protocol* [12], and by several datalink protocols such as IEEE 802.11 [5].
- Each mobile host server can multicast a packet to all query servers, and each query server can multicast a packet to all mobile host servers. These assumptions can be realized by a static configuration of all associated addresses at each server, or by a multicast protocol.
- Cells are disjoint in a logical level, that is, a mobile host belongs to at most one cell. This can be achieved by using an appropriate *hand-off* mechanism.

A mobile host server can send a query packet to a mobile host in its cell to obtain a piece of information stored in the mobile host, if necessary. The following types of location sensitive queries are considered for mobile hosts.

1. *Location (L) query*: A query to obtain the location of a mobile host.

Note that an L query is not concerned with whether the mobile host is active or not.

2. *Existence* (E) query: A query to determine if the mobile host is active.
3. *Data* (D) query: A query to obtain a piece of information from a mobile host.

3.2 Comparison of query processing and communication protocols for mobile hosts

Prior to proposing query processing strategies for mobile hosts, we discuss similarities and differences between mobile communication protocols and our concerns regarding query processing for mobile hosts (hereafter we refer to this query processing as *mobile query processing*).

In both mobile communication protocols and mobile query processing, location management of mobile hosts is very important. It is therefore natural to apply the location management schema of mobile communication protocols to those of mobile query processing. There are two basic types of actions taken by routers in mobile communication protocols:

- (1) Actions taken when routers detect migration of mobile hosts.
- (2) Actions taken when routers receive packets destined for mobile hosts.

These two actions may be compared to their respective two types of actions in mobile query processing:

- (1) Actions taken by mobile host servers when they detect migration of mobile hosts.
- (2) Actions taken by query servers when they receive queries concerning mobile hosts.

Based on this correspondence, basic strategies of mobile communication protocols such as

- broadcast of location information when a mobile host moves (BN),
- notification of location information to a certain system when a mobile host moves (DF/DQ), and

- no location management of a mobile host until a router receives a packet destined for the mobile host (BF/BQ),

are applied in a straightforward manner to mobile query processing.

However, we can find some substantial differences between mobile communication protocols and mobile query processing. First, the former concerns itself with how to forward a packet from a certain point to a mobile host, while the latter concerns itself with how to obtain a piece of information about a mobile host at a certain point. This causes a topological difference of information flow; the flow for the former is from a certain point to a mobile host, and the flow for the latter is from a certain point returned to the point. Someone may think that mobile query processing can be achieved by combining packet forwarding from the query server to the mobile host and packet forwarding from the mobile host to the query server. Such a viewpoint is correct when the requested information is only stored at the mobile host. However, we can consider the case of systems other than the mobile host having the requested information. For instance, in mobile communication protocols, the default router has the location information in DQ and DF, and all routers have the location information in BN. If a similar distribution of information is applied to mobile query processing, it is possible for a query server to realize query processing without sending a packet to the mobile host. Consequently, the optimal strategies in mobile query processing cannot be obtained by simply combining the strategies for mobile communication protocols.

Secondly, the size of a packet transmitted by a query server in mobile query processing is generally considered to be much smaller than that of a packet forwarded by a router in mobile communication protocols. The former is at most several hundreds of bytes since the packet contains only the query information sent to a mobile host. On the other hand, the latter is occasionally as much as several kilo-bytes since the packet has data requested by users and its size may become very large. According to the discussion in Section 2, query-based strategies perform better than forwarding-based strategies when the size of a packet is large. Therefore, query-based strategies are not effective in mobile query processing. From this point onwards we will not concern ourselves with query-based strategies in our search for optimal strategies in mobile query processing.

The third important difference lies in the assumptions made about their operations with special emphasis on their reliability and geographic scopes. In mobile communication protocols, it is generally desirable for a single protocol

to be employed uniformly used throughout the whole global network since it is very important for many users to access the network from its arbitrary location. To meet such a requirement in a very large scale network, very high reliability of location information is necessary. For mobile query processing, a locally restricted area, such as a campus, an office, or a manufacturing plant, should employ its own query processing strategy. This is due to the fact that users generally want to protect the databases stored in their mobile host from being accessed through the global network. Therefore, in mobile query processing, each strategy is usually used in a small restricted area where it is easy for users to find the location information of mobile hosts within the same area. Thus, high reliability of the location information is not always necessary for each mobile query processing strategy. In all the mobile communication protocols that we surveyed, when we examine the high-reliability of location information, we see that notification is made by two systems for each migration of a mobile host. These two systems are the router that detects the existence of the mobile host and the router that detects that the mobile host is no longer in its cell. However it is possible to consider a protocol where notification is made by only one system for each migration of a mobile host. In this case, it is the router that detects the existence of the mobile host. This variation is considered to be valid in the case of mobile query processing, since it is efficient in decreasing the traffic of packets and such query processing does not require high-reliability of location information.

3.3 Query processing strategies for mobile hosts

Based on the discussion of the previous subsection, the five strategies discussed in Section 2 are now applied to query processing for location sensitive queries according to the following policies:

- The actions of routers on migration of mobile hosts in mobile communication protocols are taken by mobile host servers in mobile query processing.
- The actions of routers for forwarding packets to mobile hosts in mobile communication protocols are taken by query servers for processing (L, E or D) queries in mobile query processing.
- Query-based strategies, such as DQ and BQ, will not be considered in mobile query processing.
- We consider two notification methods for handling mobility.

- (1) For each migration of a mobile host, notification is made by two mobile host servers, i.e., the previous neighbor mobile host server and new neighbor mobile host server.
- (2) For each migration of a mobile host, notification is made by a mobile host server, i.e., the new neighbor mobile host server.

Now, we propose the following strategies for handling the three types of queries.

Single Broadcast Notification (SBN): This strategy is based on the BN strategy for mobile communication protocols. As soon as a mobile host server newly detects the existence of a mobile host, the location information of the mobile host is broadcast to all query servers. In this case, even if the mobile host becomes inactive or moves into another network, no query server notices that the information it holds has become obsolete until it sends a packet to the mobile host.

For an L query from a client, each query server can reply directly. For an E and/or a D query, each query server cannot directly reply to the query. Therefore it must send a packet to the mobile host server according to the information it holds.

For example, if the mobile host i moves from e 's cell to f 's in Figure 3, f notifies all query servers a and b that it is newly adjacent to i . If the client c sends an L query of i to a , a immediately replies. For an E query or a D query of i , a should forward the query to f . Moreover, even if i moves out of f 's cell, or it becomes inactive, f does not take any more action. In this way, each query server does not know whether the mobile host is really active in the cell of the mobile host server which had previously notified it of the existence of the mobile host.

double Broadcast Notification (WBN): This strategy is based on the BN strategy. As soon as a mobile host server newly detects the existence of a mobile host, the location information of the mobile host is broadcast to all query servers. Furthermore, when a mobile host server newly detects that a certain mobile host which had resided in its cell no longer exists in the cell, such non-existence information is broadcast to all query servers. In this case, if a mobile host has become inactive or moves into another network, each query server will notice that the information it holds on this mobile host has become obsolete, which makes it eliminate the entry. The IS-IS protocol uses this notification method, and this is a popular method for routing in networks in which migration is not so frequent.

Each query server can directly reply to every L query and E query. Only in the case of a D query, the query server cannot directly answer. Therefore it must send a packet to the mobile host server according to the information it holds. For example, if the mobile host i moves from e 's cell to f 's in Figure 3, besides f 's action of notifying all query servers of the existence of i in its cell, e notifies all query servers that i is no longer in its cell. Each query server, e.g., a , immediately replies to L queries as well as E queries. Only for a D query of i , a should forward it to f . Moreover, if i moves out of f 's cell, or it becomes inactive, f notifies all query servers that i is no longer in its cell. In this way, each query server knows whether the mobile host is really active in the cell of the mobile host server which has previously notified it of the existence of the mobile host.

Broadcast Query Forwarding (BQF): This strategy is based on the BF strategy used in mobile communication protocols. No action is taken if a mobile host server newly detects the existence of a mobile host. No action is taken either even if a mobile host server detects that a certain mobile host which had resided in its cell no longer exists in the cell. Mobile information can be obtained by broadcasting a packet to all mobile host servers. Therefore, for any of L, E, and D queries, the query server broadcasts a packet to all mobile host servers. The mobile host server in whose cell the objective mobile host resides gives the answers to the query server.

For example, even if the mobile host i moves from e 's cell to f 's in Figure 3, no action will be taken by either e or f . When a client c sends a query about i to a query server a , the server sends the contents of the query to all mobile host servers, and then the mobile host server f , in whose cell the mobile host i resides, replies to the query.

Single Default Notification (SDN): This strategy is based on the DF strategy for mobile communication protocols. There exists a *default server* for each mobile host, and each mobile host server and each query server know which system is the default server of each mobile host beforehand. As soon as a mobile host server newly detects the existence of a mobile host, the default server is notified of the mobile host's location. In this case, if the mobile host becomes inactive or moves into another network, the default server cannot notice that the information being held has become obsolete unless it sends a packet to the mobile host.

For an L query, a query server can reply by sending a query to its default server. For an E and a D query, the default server cannot reply. Therefore the

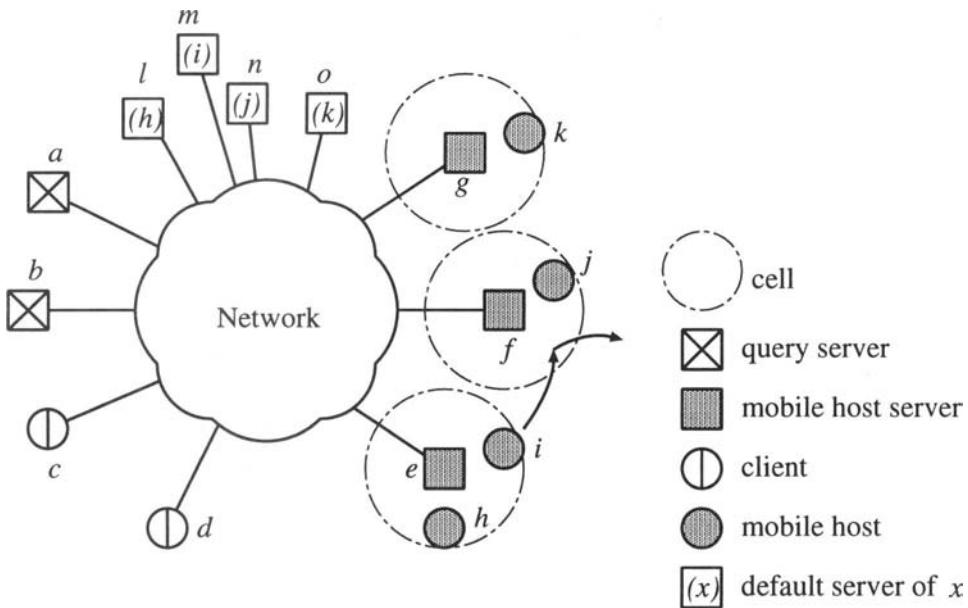


Figure 4 A network example for SDN and WDN.

default server must forward a packet to the mobile host server according to the information it holds.

In Figure 4, let the systems l , m , n , and o be the default servers of the mobile hosts h , i , j , and k , respectively.

If the mobile host i moves from e 's cell to f 's cell, f notifies m , the default server of i , of i 's existence. If a query server a receives an L query of i from a client c , it asks m for the location of i , and then m replies to this query.

For an E query and a D query of i , since m cannot directly answer, it forwards the query to f . Moreover, even if i moves out of the f 's cell, or it becomes inactive, f does not take further action. With this strategy, the default server does not know whether the mobile host is really active in the cell of the mobile host server which had provided previous notification of the mobile host's existence.

double Default Notification (WDN): This strategy is based on the DF strategy for mobile communication protocols. There exists a default server for each mobile host, and each mobile host server and each query server know beforehand which system is the default server of each mobile host. As soon as a mobile host server newly detects the existence of a mobile host, the default server is notified of the mobile host's location. Furthermore, as soon as a mobile host server newly detects that the mobile host which had resided in one of its cell no longer exists in the cell, the default server is notified by the mobile host server that the mobile host no longer resides in its cell. In this case, if the mobile host becomes inactive or moves to another network, the default server will notice that the information being held has become obsolete. The default server will then eliminate the entry.

For an L query and an E query, each query server can answer by sending a query to the default server. Only for a D query is the default server unable to reply. Therefore the default server must forward a packet to the mobile host server according to the information it holds.

For example, in Figure 4, if the mobile host i moves from e 's cell to f 's cell, f informs m , the default server of i of i 's existence. The mobile host server e also notifies m when i is no longer in its cell. If a query server a receives an L query or an E query of i from a client c , it sends the query to m , and m replies to this query. Since m itself cannot directly answer a D query of i , it forwards the query to f . Moreover, if i moves out of f 's cell, or becomes inactive, f will notify m that i is no longer in its cell.

With this strategy, the default server does know whether the mobile host is really active in the cell of the mobile host server which had provided previous notification it of the mobile host's existence.

4 EVALUATION

In this section, we compare the costs of SBN, WBN, BQF, SDN, and WDN. Here we use the following parameters in our cost estimation:

Table 1 Cost estimates of each strategy

strategy	migration	L query	E query	D query
SBN	$B_f^{(M)}$	0	$2H_{a,f}$	$2H_{a,f}$
WBN	$B_e^{(M)} + B_f^{(M)}$	0	0	$2H_{a,f}$
BQF	0	$B_a^{(Q)} + H_{a,f}$	$B_a^{(Q)} + H_{a,f}$	$B_a^{(Q)} + H_{a,f}$
SDN	$D_{f,i}^{(M)}$	$2D_{a,i}^{(Q)}$	$D_{a,i}^{(Q)} + D_{f,i}^{(M)}$ + $H_{a,f}$	$D_{a,i}^{(Q)} + D_{f,i}^{(M)}$ + $H_{a,f}$
WDN	$D_{e,i}^{(M)} + D_{f,i}^{(M)}$	$2D_{a,i}^{(Q)}$	$2D_{a,i}^{(Q)}$	$D_{a,i}^{(Q)} + D_{f,i}^{(M)}$ + $H_{a,f}$

- $B_x^{(Q)}$: the cost of broadcasting a packet from a query server x to all the mobile host servers.
 $B_x^{(M)}$: the cost of broadcasting a packet from a mobile host server x to all the query servers.
 $H_{x,y}$: the cost of sending a packet between a query server x and a mobile host server y .
 $D_{x,y}^{(Q)}$: the cost of sending a packet between a query server x and the default server of a mobile host y .
 $D_{x,y}^{(M)}$: the cost of sending a packet between a mobile host server x and the default server of a mobile host y .

Table 1 shows the costs incurred by each strategy for the following cases: migration of the mobile host i as it moves from the cell of mobile host server e to the cell of the mobile host server f , query server a processing a L query and an E query, and a D query concerning i when i is in f 's cell. In this table, we do not include the cost of transmitting a packet between a mobile host server and a mobile host.

Now, similar to other approaches such as [3], [9], [11], [15], [18], and [23], we assume that migration intervals of each mobile host and query occurrence intervals are exponentially distributed. We also assume that the location to which each mobile host moves is randomly selected from all the possible mobile host servers. Let μ be the migration rate, i.e., the mean number of migrating mobile hosts in a unit time, and λ be the query occurrence rate, i.e., the mean number of queries occurring in a unit time. Queries are composed of α of L queries, β of E queries, and γ of D queries such that $\alpha + \beta + \gamma = 1$. Let

$\overline{B^{(M)}}$ be the average of $B_x^{(M)}$ for an arbitrary mobile host server x , $\overline{B^{(Q)}}$ be the average of $B_x^{(Q)}$ for an arbitrary query server x , \overline{H} be the average of $H_{x,y}$ for an arbitrary combination of query server x and mobile host server y , $\overline{D^{(M)}}$ be the average of $D_{x,y}^{(M)}$ for an arbitrary combination of mobile host server x and mobile host y , and $\overline{D^{(Q)}}$ be the average of $D_{x,y}^{(Q)}$ for an arbitrary combination of query server x and mobile host y . The mean total cost in a unit time in each strategy is as follows:

$$\begin{aligned}\text{SBN: } & \overline{B^{(M)}}\mu + 2\overline{H}(\beta + \gamma)\lambda \\ \text{WBN: } & 2\overline{B^{(M)}}\mu + 2\overline{H}\gamma\lambda \\ \text{BQF: } & \frac{(\overline{B^{(Q)}} + \overline{H})(\alpha + \beta + \gamma)\lambda}{\overline{D^{(Q)}}} \\ \text{SDN: } & \frac{\overline{D^{(M)}}\mu + 2\overline{D^{(Q)}}\alpha\lambda + (\overline{D^{(Q)}} + \overline{D^{(M)}} + \overline{H})(\beta + \gamma)\lambda}{\overline{D^{(Q)}}} \\ \text{WDN: } & \frac{2\overline{D^{(M)}}\mu + 2\overline{D^{(Q)}}(\alpha + \beta)\lambda + (\overline{D^{(Q)}} + \overline{D^{(M)}} + \overline{H})\gamma\lambda}{\overline{D^{(Q)}} + \overline{D^{(M)}} + \overline{H}}\end{aligned}$$

Now let us consider how the predominant strategy changes according to network parameters in two typical network topologies: *n-grid* topologies and *binary tree* topologies.

Example 1 : n-grid topologies. Consider the network topology shown by Figure 5, which is scaled by integer n . Here the default server for each mobile host is the most central server if n is odd, and one of the central four servers if n is even. All servers are mobile host servers. The cost is estimated by counting the hops between servers. Packet broadcast is efficient if a certain *dynamic spanning tree algorithm* is used. B , H , and D are expressed as follows:

$$\begin{aligned}\overline{B^{(Q)}} &= \overline{B^{(M)}} = n^2 - 1 \\ \overline{H} &= \frac{2(n-1)(n+1)}{3n} \\ \overline{D^{(Q)}} &= \overline{D^{(M)}} = \begin{cases} \frac{(n+1)(n-1)}{2n} & (n : \text{odd}) \\ \frac{n}{2} & (n : \text{even}) \end{cases}\end{aligned}$$

A campus network is a typical example of *n-grid* topology since it usually ranges geographically over the campus. A grid topology is often used as a typical topology for OSI networks because a dynamic routing protocol, the IS-IS protocol, can be efficiently used in networks that include many highly-multiplexed paths, which are typical of a grid topology.

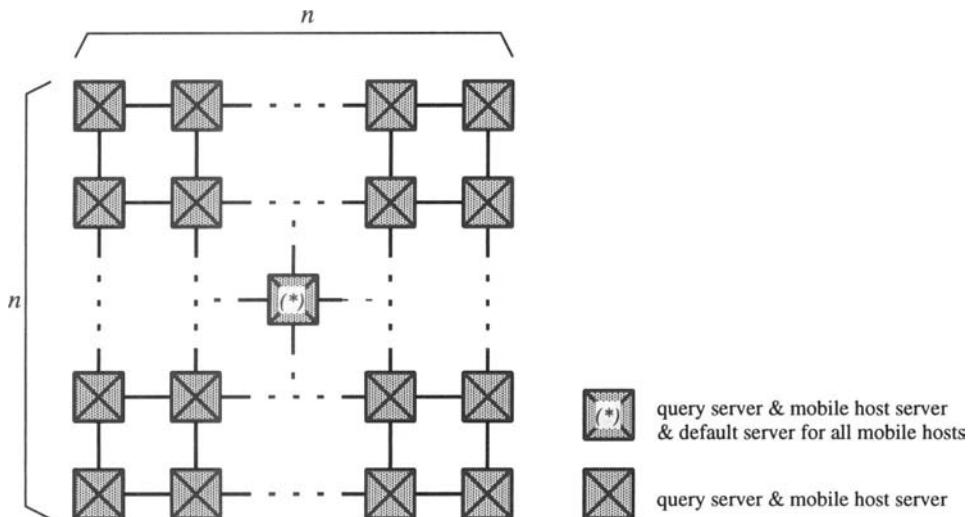


Figure 5 *n*-grid topologies.

First, we show how these three types of queries affect the total cost. Figure 6, Figure 7, and Figure 8 respectively illustrate the total costs per move for given query/migration ratio in three extreme cases, (1) $\alpha = 1, \beta = 0, \gamma = 0$, and $n = 3, 10$, (2) $\alpha = 0, \beta = 1, \gamma = 0$, and $n = 3, 10$, and (3) $\alpha = 0, \beta = 0, \gamma = 1$, and $n = 3, 10$. In these figures, the horizontal axes are the query/migration ratio λ/μ , and the vertical axes are the total cost per move. Note that query/migration ratio is similar to the call/mobility ratio in [9]. According to the query/migration ratio, the optimal strategy becomes (1) BQF, SDN, or SBN for L queries, (2) BQF, WDN, or WBN for E queries, and (3) BQF, SDN, or SBN for D queries.

Next we see complex cases of three types of queries. The total costs for the cases $\alpha = 0.2, \beta = 0.2, \gamma = 0.6$, and $n = 3, 10$ are illustrated in Figure 9. By this figure, we can see that, in both cases $n = 3$ and $n = 10$, an arbitrary strategy can be optimal according to the query/migration ratio.

For a given query/migration ratio and a given network scale n , the domain in which each strategy is optimal is illustrated by Figure 10. We can see by this figure that all strategies can be optimal according to the query/mobility ratio and the network scale n . BQF is better than the others if migration is assumed to be frequent, while SBN or WBN are best used if the occurrence of queries

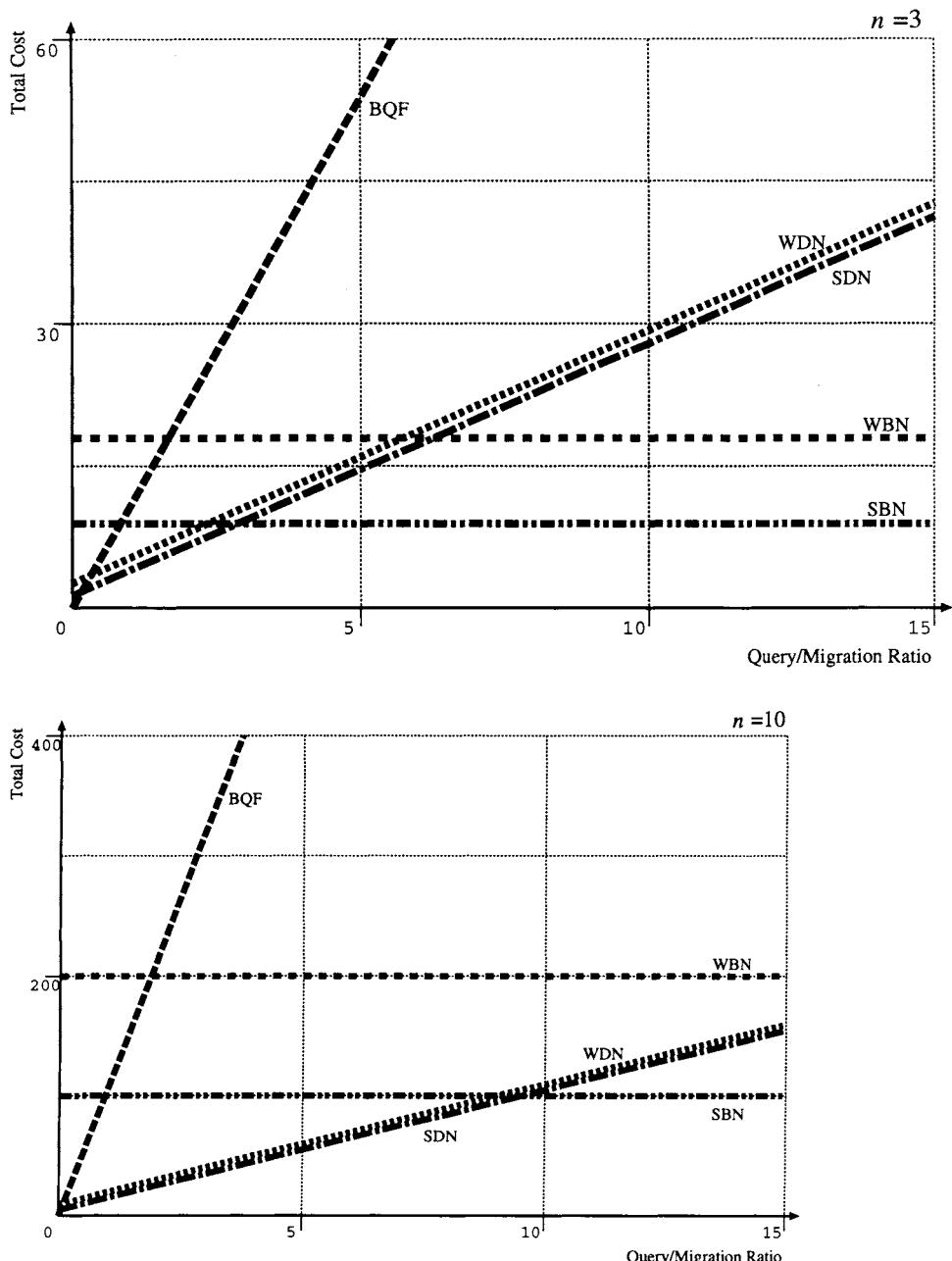


Figure 6 Total cost comparison of an n -grid topology for L queries.

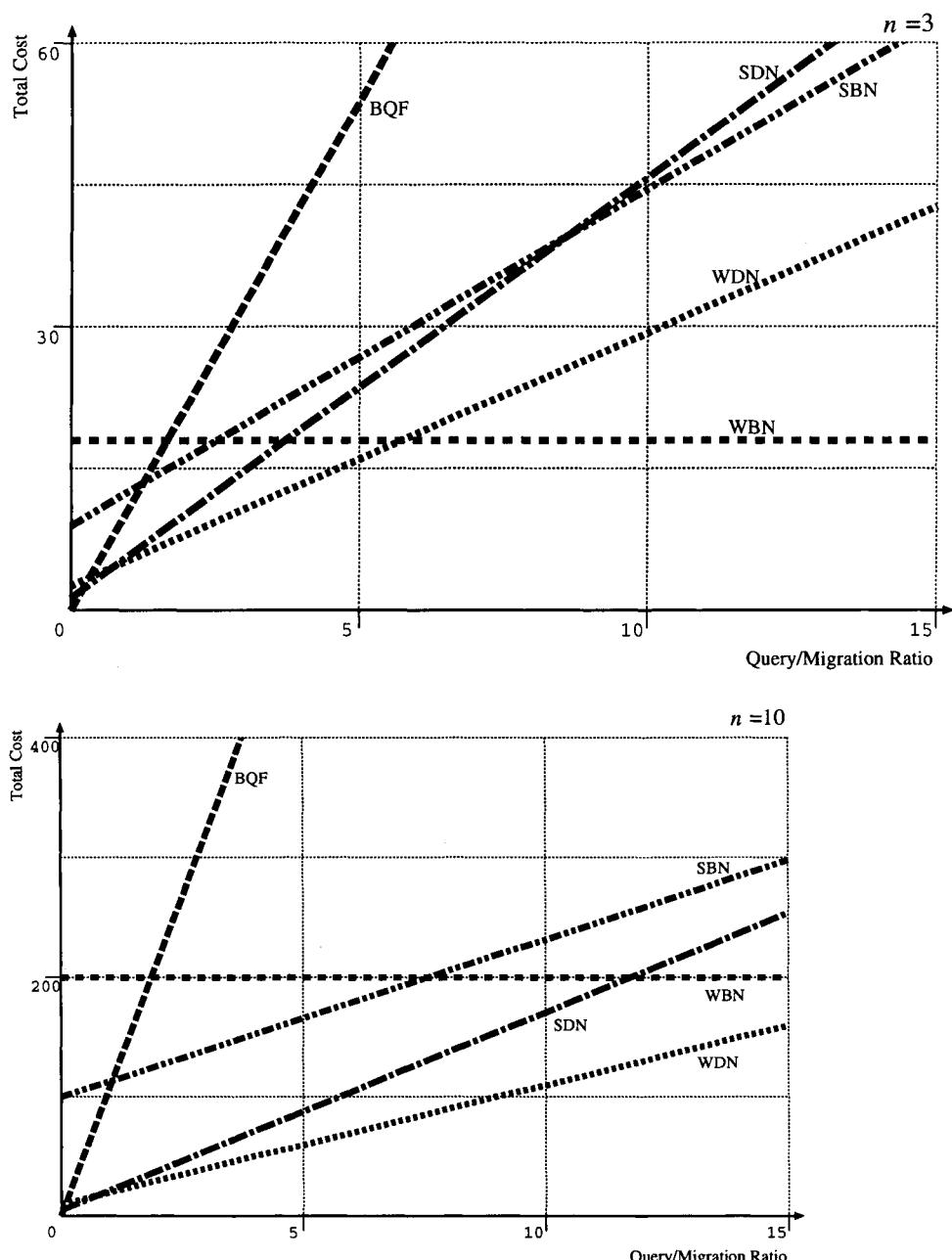


Figure 7 Total cost comparison of an n -grid topology for E queries.

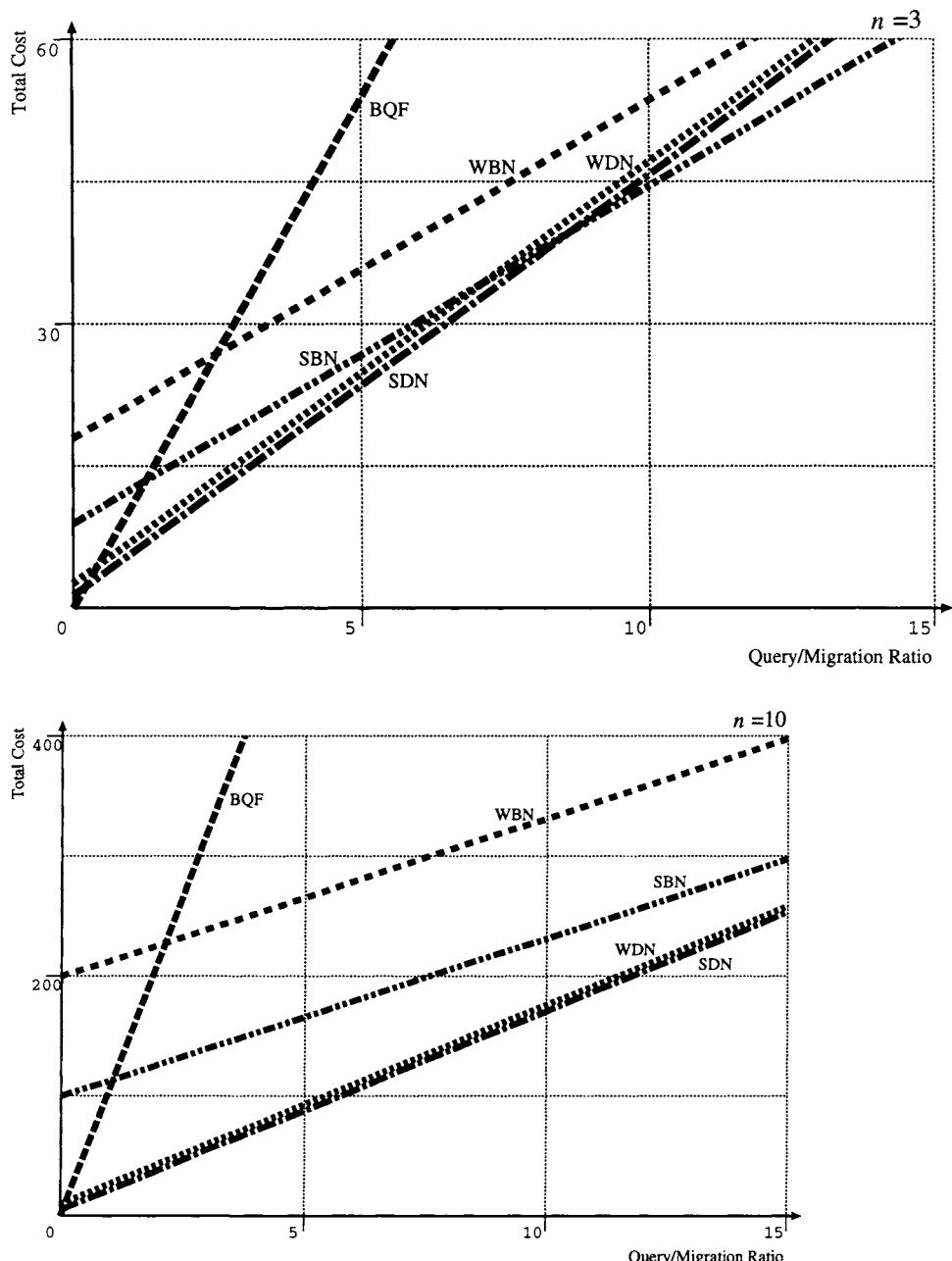


Figure 8 Total cost comparison of an n -grid topology for D queries.

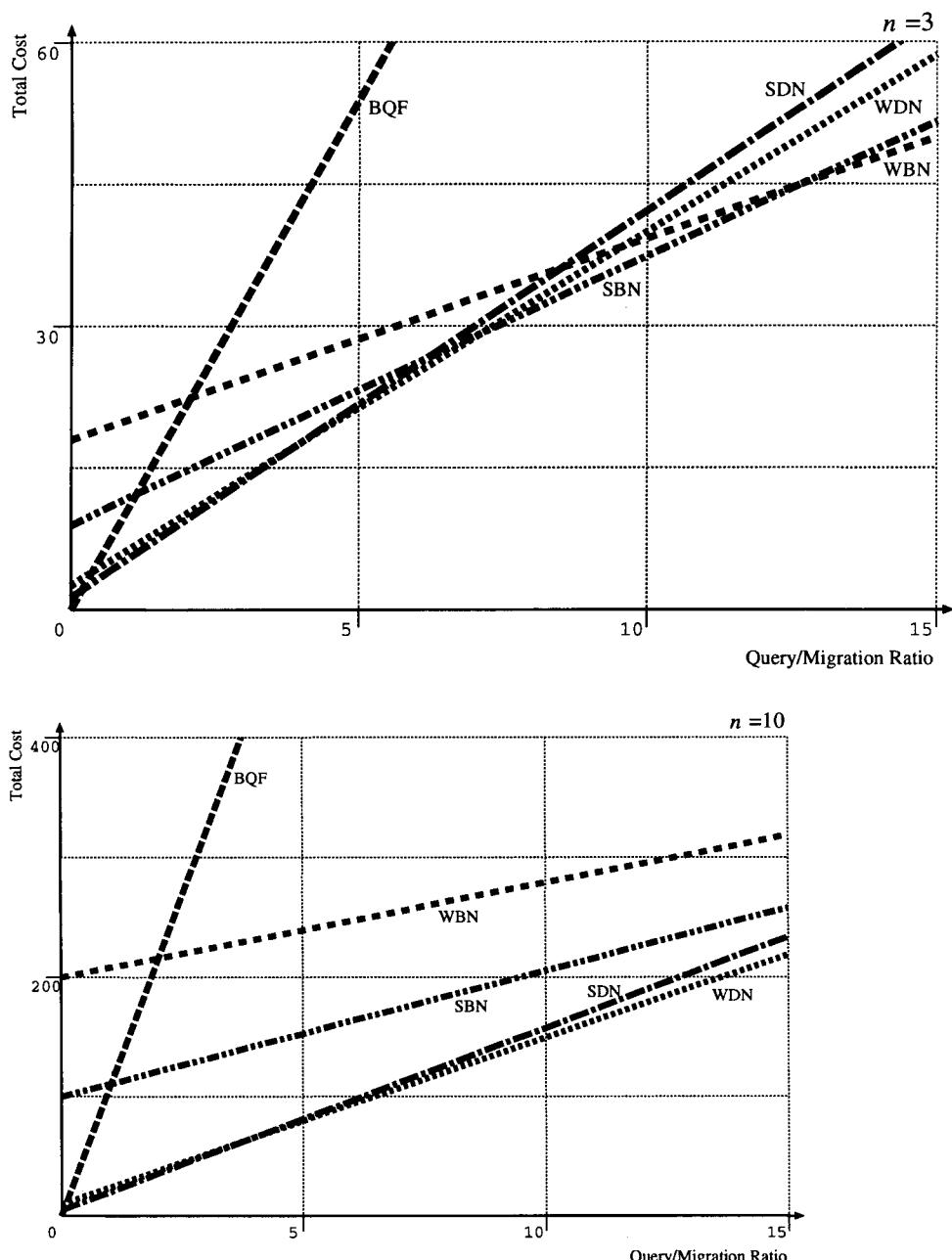


Figure 9 Total cost comparison of an n -grid topology for case $\alpha = 0.2, \beta = 0.2, \gamma = 0.6$.

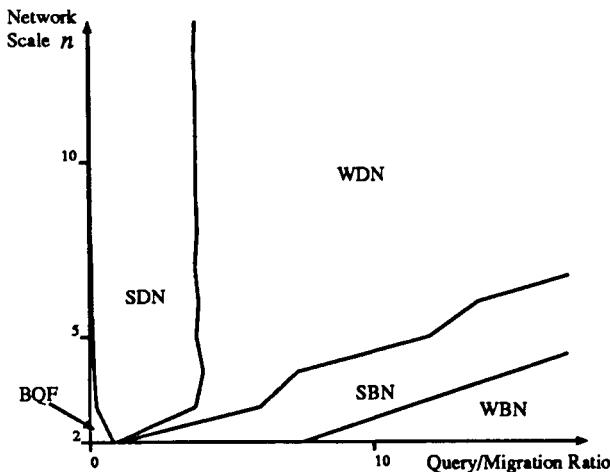


Figure 10 Optimal strategy for n -grid topologies.

is assumed to be frequent. Default-based strategies SDN and WDN are more scalable than broadcast-based strategies. The costs for SDN, WDN are $O(n)$, while those for SBN, WBN, and BQF are $O(n^2)$. From this comparison we can say that broadcast-based strategies are not appropriate for a large scale network.

Example 2 : binary tree topologies (depth n). Consider the network topology shown in Figure 11, which is scaled by integer n . Here the default server for each mobile host is the root of the tree. Leaf systems are mobile host servers as well as query servers.

A wide area network is a typical example of a binary tree topology. The *internet*, with its hierarchical construction rooted from a single backbone, is based on a binary tree topology. This topology is appropriate for the manual configuration of routing tables since there is no possibility of packet looping and no alternative choices of the forwarding direction for a given destination.

The cost is estimated from the hop count between servers. Here we assume that queries are invoked randomly by each mobile host. B , H , and D are expressed as follows:

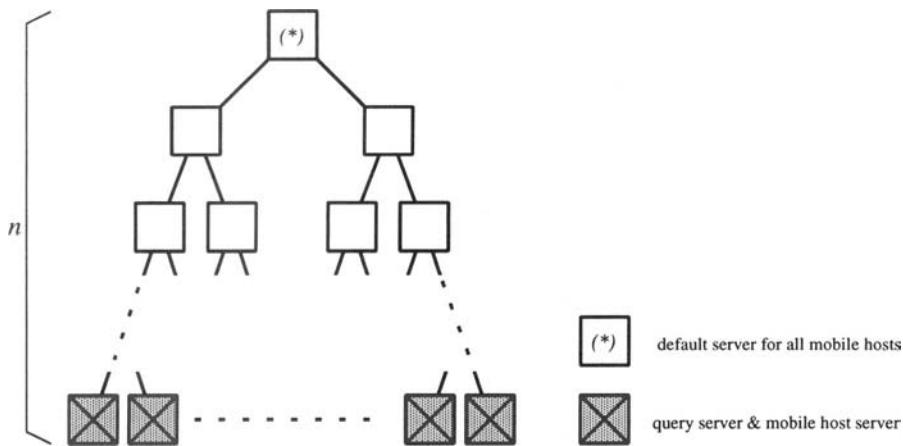


Figure 11 Binary tree topologies.

$$\begin{aligned}\overline{B^{(Q)}} &= \overline{B^{(M)}} = 2^n - 2 \\ \overline{H} &= 2(n-2) + 2^{2-n} \\ \overline{D^{(Q)}} &= \overline{D^{(M)}} = n-1\end{aligned}$$

When we assume $\alpha = 0.3$, $\beta = 0.2$, and $\gamma = 0.5$, the optimal strategy for a given n and the query/migration ratio is shown in Figure 12. We can see from this figure that all strategies can be optimal according to n and query/migration ratio. We can also see a similarity to grid topology in this case. Since broadcasting costs $O(2^n)$ in this case, the tendency in scalability is more conspicuous.

5 CONCLUSIONS

In this paper, we have discussed three types of queries L, E, and D for dealing with mobile hosts and have shown five strategies SBN, WBN, BQF, SDN, and WDN for query processing. We have analytically compared their performance from a network cost point of view and shown how the optimal strategies vary according to the conditions such as the migration rate of the mobile hosts, the query occurrence rate, and the scale of the network. The comparison was carried out for two typical network topologies: grid topology and tree topology.

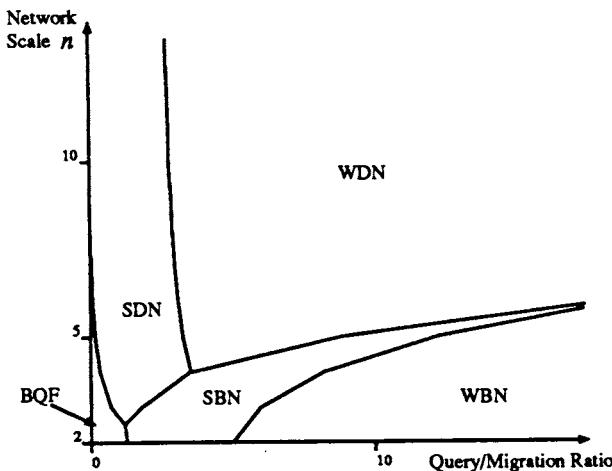


Figure 12 Optimal strategy for binary tree topologies.

As a result of this paper, we could say that implementing all these strategies enables mobile users to select the optimal strategy when the network conditions for each user differ.

The scope of this paper has been fundamental strategies. Several advanced techniques, such as *cache*, *interception* [18], *hierarchical server* [2] or *pointer* [9], were not taken into account in the cost analysis. The evaluation of the strategies using the abovementioned techniques is one of our important future research issues.

REFERENCES

- [1] Alonso, R. and Korth, H.F., "Database System Issues in Nomadic Computing," Proc. of ACM-SIGMOD'93, pp.388-392, 1993.
- [2] Awerbuch, B. and Peleg, D., "Concurrent Online Tracking of Mobile User," Proc. of ACM-SIGCOMM'91, pp.221-233, 1991.
- [3] Bar-Noy, A., Kessler, I., and Sidi, M., "Mobile Users: To Update or not to Update," Proc. of IEEE INFOCOM'94, pp.570-576, 1994.

- [4] Carlberg, K.G., "A Routing Architecture That Supports Mobile End Systems," Proc. of IEEE MILCOM'92, pp.159-164, 1992.
- [5] Dieplstraten, W., Ennis, G., and Belanger, P., "DFWMAC: Distributed Foundation Wireless Medium Access Control," IEEE P802.11-93/190, 1993.
- [6] Forman, G.H. and Zahorjan, J., "The Challenges of Mobile Computing, IEEE COMPUTER," Vol.27, No.4, pp.38-47, 1994.
- [7] Hahn, J.J. and Stolle, D.D., "Packet Radio Network Algorithms: A Survey," IEEE Communications Magazine, Vol.22, No.11, pp.41-47, 1984.
- [8] Ioannidis, J., Duchamp, D. and Maguire Jr., G.Q., "IP-based Protocols for Mobile Internetworking," Proc. ACM-SIGCOMM'91, pp.235-245, 1991.
- [9] Imielinski, T. and Badrinath, B.R., "Querying in Highly Mobile Distributed Environments," Proc. of VLDB'92, pp.41-52, 1992.
- [10] Imielinski, T. and Badrinath, B.R., "Data Management for Mobile Computing," ACM SIGMOD RECORD, Vol.22, No.1, pp.34-39, 1993.
- [11] Imielinski, T. and Badrinath, B.R., "Mobile Wireless Computing: Solutions and Challenges in Data Management," Technical Report DCS-TR-296, Department of Computer Science, Rutgers University, 1993.
- [12] ISO 9542, "Information Processing Systems - Telecommunications and Information Exchange between Systems - End System to Intermediate System Routeing Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-mode Network Service (ISO 8473)," 1988.
- [13] ISO/IEC 10589, "Information Technology - Telecommunications and Information Exchange between Systems - Intermediate System to Intermediate System Intra-Domain Routeing Information Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-mode Network Service (ISO 8473)," 1992.
- [14] Korth, H.F. and Imielinski, T., "Mobile Computing: Fertile Research Area or Black Hole?," Proc. of VLDB'93, pp.699-700, 1993.
- [15] Madhow, U., Honig, M.L. and Steiglitz, K., "Optimization of Wireless Resources for Personal Communications Mobility Tracking," Proc. of IEEE INFOCOM'94, pp.577-584, 1994.
- [16] Moy, J., "OSPF Version 2," RFC 1583, 1994.

- [17] Perkins, C.E. and Bhagwat, R., "A Mobile Networking System based on Internet Protocol," IEEE Personal Communication, Vol.1, No.1, pp.32-41, 1994.
- [18] Tanaka, R. and Tsukamoto, M., "A CLNP-based Protocol for Mobile End Systems within an Area," Proc. of IEEE International Conference on Network Protocols, pp.64-71, 1993.
- [19] Teraoka, F., Yokote, Y., and Tokoro, M., "A Network Architecture Providing Host Migration Transparency," Proc. of ACM-SIGCOMM'91, pp.45-65, 1991.
- [20] Tsukamoto, M. and Tanaka, R., "A Routing Protocol for Wide Area Migration using Default Address and Remaining Lifetime Parameter," IPSJ-DPS-Report 58-3, pp.17-24, 1992 (in Japanese).
- [21] Tsukamoto, M., Tanaka, R., and Tsumori, O., "Supporting ES-Migration within Areas in CLNP Networks," IPSJ-DPS-Report 61-30, pp.227-234, 1993 (in Japanese).
- [22] Wada, H., Yozawa, T., Ohnishi, T., and Tanaka, Y., "Mobile Computing Environment Based on Internet Packet Forwarding," Proc. of 1993 Winter USENIX, pp.503-517, 1993.
- [23] Yuan, R., "Traffic Pattern Based Mobile Routing Scheme," Computer Communications, Vol.18, No.1, pp.32-36, 1995.

THE CASE FOR WIRELESS OVERLAY NETWORKS

Randy H. Katz and Eric A. Brewer

*Computer Science Division, Department of Electrical
Engineering and Computer Science, University of California, Berkeley,
CA 94720-1776.*

ABSTRACT

Wireless data services, other than those for electronic mail or paging, have thus far been more of a promise than a success. We believe that future mobile information systems must be built upon heterogeneous *wireless overlay networks*, extending traditional wired and internetworked processing “islands” to hosts on the move over coverage areas ranging from in-room, in-building, campus, metropolitan, and wide-areas. Unfortunately, network planners continue to think in terms of homogeneous wireless communications systems and technologies. In this paper, we describe our approach towards a new wireless data networking architecture that integrates diverse wireless technologies into a seamless wireless (and wireline) internetwork. In addition, we describe the applications support services needed to make it possible for applications to continue to operate as mobile hosts roam across such networks.

1 INTRODUCTION

With the recent award of the PCS licenses (data and otherwise), combined with the successful rollout of digital direct broadcast satellite services, the digitization of cellular services, and the emerging deployments of cellular packet services, alternative wireless voice and data services are poised to proliferate. Unfortunately, network planners continue to think in terms of homogeneous wireless communications systems and technologies, providing either low bandwidth connectivity over the wide-area (for short messaging or email) or high bandwidth connectivity over a small area like a building (for conventional Internetwork access). Nobody seems to know what will be the “killer” wireless

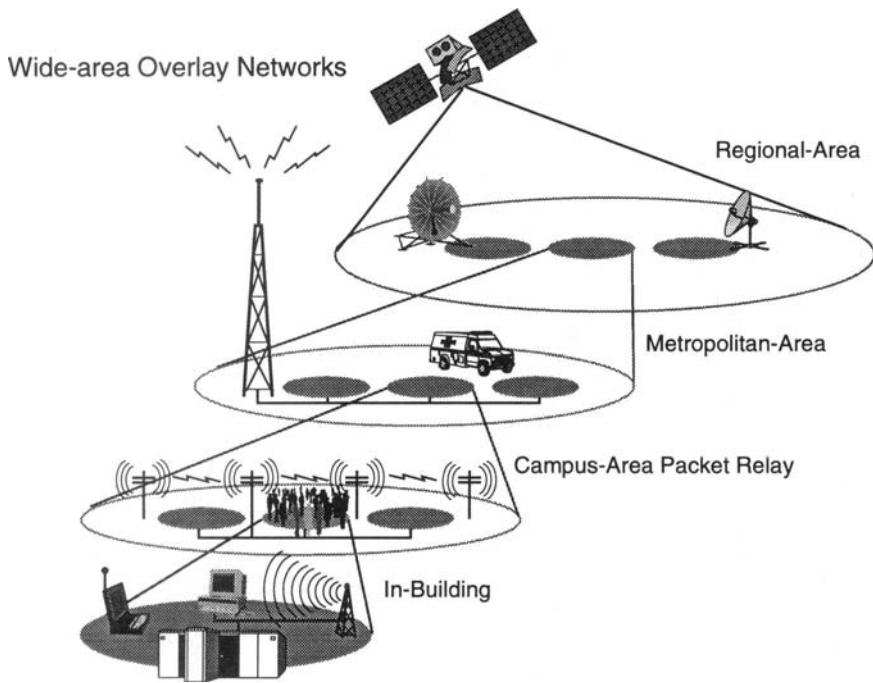


Figure 1 Wireless Overlay Networks

data application. While the marketing focus is on bandwidth, from an applications perspective, packet latency is as important an applications enabler. Yet latency in wireless data networks has not been well characterized.

Future mobile information systems are likely to be built upon heterogeneous *wireless overlay networks*, extending traditional wired and internetworked processing “islands” to hosts on the move over a wide area. For example, a high bandwidth in-room infrared (IR) network can be “overlaid” with a more moderate bandwidth radio frequency (RF) network to provide connectivity between areas of IR coverage. Or a low-tier (low mobility) in-building PCS system could be overlaid with a high-tier (high mobility) PCS system to provide cost effective connectivity bridging local and wide areas. Figure 1 illustrates the general concept of wireless overlay networks.

Because they are constructed from such diverse wireless technologies with different engineering constraints, overlay networks vary widely in bandwidth, latency, and coverage. One cannot expect a wide-area system to compete in

terms of bandwidth with an in-building system. The transmission power and methods for handling multipath fading are very different. From our viewpoint, the key technical challenge is to develop *overlay internetwork management* and *applications support services* that allow mobile applications to operate across a wide range of network performance, roaming among potentially competing service providers, and choosing among alternative overlays for best performance given the current network state and application needs.

To verify the utility of the overlay concept, we are creating a wireless overlay network testbed in the San Francisco Bay Area as a collaboration with several network service providers and technology developers, spanning from in-building wireless LANs to direct broadcast satellite systems over a regional area. We are developing pilot applications to drive the design and validation of the interfaces between applications and the network. Our overarching goal is to demonstrate a scalable architecture that can support wireless access across multiple overlay networks while delivering high levels of end-to-end performance to applications.

The rest of this paper is organized as follows. In the next section, we illustrate the value of the overlay concept by giving an applications scenario that could only be possible were such technologies available. In Section 3, we describe existing and emerging wireless data technologies, and their strengths and weaknesses for supporting bandwidth and latency-sensitive applications. Our “gateway-centered” overlay architecture is presented in Section 4. Mechanisms for network management and handoff are described in Section 5, and our approach for application support services is given in Section 6. Section 7 describes related work. Section 8 presents our summary and conclusions.

2 APPLICATIONS ENABLED BY WIRELESS OVERLAYS

Consider the following scenario of future applications made possible by wireless overlay networks. A traffic helicopter observes a serious auto accident on the Golden Gate Bridge. It communicates for an emergency medical team to be dispatched to the scene. The helicopter, using a wide-area overlay network, transmits low frame rate digitized video of the crash site to the medical team, to assist them in assessing the severity of the crash and the need for supporting units. They decide to request backup. The helicopter crew also establishes a two-way audio conversation with the ambulance driver to direct him to the

scene via the least congested traffic route, perhaps forwarding the medical team a street map with a hand annotated route.

At the scene, the medical team and police units free the victims from the wreckage, stabilize their condition, and head them to Mount Zion Hospital in San Francisco. The victims are identified, and one of the injured passengers has a serious head injury, requiring immediate emergency surgery. The team requests the preparation of a suitable medical facility at the hospital. Critical medical information — such as patient reactions to various medications — is downloaded from a regional computer-based patient care record archive at the UCSF Medical Center to the medical teams in the ambulances, again via a wide-area wireless network. Simultaneously, detailed patient records and archived neuro-images are forwarded to an image file server at Mount Zion via a high performance wireline network. The in-hospital wireless network allows the emergency room physicians to retrieve and view the patient's images and records on their portable multimedia devices while on the move. They decide that a CAT scan should be performed on the patient upon arrival. The new scan, combined with existing patient information, will support their planning of the proper surgical suite and determination of a course of operating procedure.

After the ambulance arrives at Mount Zion, the patient is taken to the CAT scanner. The archive system automatically appends the new neuro-images with the victim's existing patient information. The surgical team assesses the injury's seriousness based on this new information, and decides that a 3-D reconstruction of the CAT images is needed. Mount Zion Hospital requests UCSF's high performance computers construct a 3-D visualization of the head trauma and provide a computer-assisted guide for the surgical procedure. These are instantaneously forwarded to Mount Zion via the wireline network linking the hospitals, where they are rerouted to the surgeon's portable display via the wireless in-hospital network while he is enroute to the operating room. Its unusual nature prompts him to hold an impromptu video consultation with a distinguished neurosurgeon on his way to a lecture at the UCSF Medical School across town. Just as the two doctors agree on a course of surgery, the stretcher is wheeled into the operating room, and the neurosurgeon steps into his lecture hall.

This scenario illustrates several requirements that we believe broadly characterize a wide range of "tactical" applications in emergency response, public safety/law enforcement, and military operations:

- Seamless interoperation of wide-area wireless overlay communications for operational units on the move, able to exploit higher quality communications inside buildings yet still able to provide useful functionality over wider-area, lower bandwidth, higher latency wireless technologies;
- Access by mobile applications to the full computation and information resources of in-building infrastructures, e.g., distributed processing, visualization, and digital library access;
- Rapid exchange of media-rich data, including video images, audio communications, annotated maps and graphics, handwriting and text, all in support of decision makers on the move, in real time where possible, and in near real time otherwise;
- Type-sensitive data exchange adapted to the available communications quality, exploiting mechanisms like bandwidth-adaptive compression, latency-adaptive mobile host frontend processing and wireline backend processing.

3 APPLICATIONS VIEWPOINT

Wireless data networks, beyond simple paging and messaging, have been considerably more promising than successful to date. Some experts believe that there are no more than 100,000 wireless data users in the U.S. (compared to 25 million cellular telephone users!). No single technology offers the ideal combination of network parameters: wide-area coverage, high bandwidth and low latency, with support for vehicular as well as pedestrian mobility. To yield flexible connectivity over wide areas, what is needed is a wireless internetwork formed from multiple wireless overlays interconnected by wired segments.

The marketing literature for wireless is fixated on bandwidth, but there is little discussion of other equally important quality metrics, such as latency, expected packet loss rates, probability of packet retransmission, and available bandwidth per cubic foot (effectiveness of frequency reuse). In addition, it is important to understand user and applications needs, and how well these are met can be met by a wireless network. For example, a 10 mbps Ethernet typically supports 50 users; a 1 mbps in-room IR network designed to support 5 users collocated in the same room might be just as effective. Certain applications, such as World Wide Web browsing, are reasonably tolerant of latency. Others, like Eudora-style decoupled mail access, can operate quite well even in a network environment characterized by high latencies. Critical issues for wireless network technologies are scaling performance with increasing user densities, handling

asymmetric bandwidth (it is frequently the case that downlink bandwidth is higher than uplink), supporting hybrid networks (different technologies for the downlink and the uplink, as is the case in satellite direct broadcast systems), support for roaming and mobility (e.g., Mobile IP), and the relative power consumption of alternative wireless technologies.

Consider the following set of wireless technologies and their relative performance advantages and disadvantages:

- In-Room Infrared: Infrared technology is attractive for providing wireless connectivity for well defined physical spaces like offices and meeting rooms (typically 30 m x 30 m). Commercially available diffuse IR devices achieve 1 mbps (e.g., Photonics). Observed latencies are comparable to wireline networks. Research devices have been successfully demonstrated at 50 mbps [11]. Even more modest megabits per second should allow reasonable quality compressed video and high quality audio. Directed IR interfaces are expected to be inexpensive and ubiquitous in future PDAs and laptop computers (Apple Newton, Sony Magic Link, HP palmtops, newest generation of IBM Thinkpads, etc.), although they are primarily being contemplated for interfacing to peripherals and docking stations. Nevertheless, these could be adapted to networking uses.
- In-Building Radio Frequency: Numerous wireless local area networking products are commercially available using spread spectrum in unlicensed ISM bands. They yield bandwidth in the 1-2 mbps range, but whether this bandwidth is really reused among adjacent cells varies among the product implementations (for example, the popular AT&T WaveLAN devices cannot reuse frequencies because they do not use orthogonal spreading codes). Good quality compressed video and high quality audio can be achieved as long as user densities are kept modest. A typical cell size is a several hundred feet — enough to cover perhaps half the floor of a modern university building. Picocellular architectures with much smaller cell sizes are more interesting in terms of aggregate bandwidths, but today's economics makes the approach somewhat expensive. The newest generation of WLANs support PCMCIA interface cards, making these useful for truly portable devices like PDAs and laptops. While typically implemented with a base station/mobile node architecture, the radios can easily support packet radio techniques. It is simply a small matter of programming!
- Campus/Metropolitan Area Packet "Relay" Networks: Metricom is an example of a deployed packet radio network that supports multihop portable radios, geographical routing, and pedestrian mobility. The routing

problem is easier than general packet radio networks since the poletop infrastructure radios do not move. 100 kbps radio-to-radio transmission rates are supported, with modem-to-poletop bandwidths in the 20-30 kbps range. The available bandwidth must be shared among all users in the cell, which could be 1 to 5 miles in diameter (though for high traffic areas, they could be as small as 100 feet). Latency is a function of the number of radio hops before reaching wireline access to the Internet. Assuming an uncongested network, the latency is about 40 ms between modem and poletop, and 20 ms for each additional hop. 2-3 hops are typical. More poletop radios help to increase the aggregate bandwidth, but additional wireline connections are also needed to keep the number of hops less than four (also, poletop radios around the wireline access point are likely to be "hot spots," leading to increased congestion).

- **Wide-Area Packet Switched Data Networks:** Cellular Digital Packet Data (CDPD) is a wide-area data overlay to the analog cellular system now being deployed. The system is designed for instantaneous transmission rates at 19.2 kbps, but this will be difficult to sustain, especially during times of peak demand for cellular voice services. Pricing remains high, in the range of 10–18 cents per KByte, making the service most cost effective for short messages and data transactions like credit card authentication. Latencies will be highly variable, depending on the competing cellular voice traffic. Other wide-area data systems currently deployed include ARDIS and RAM Mobile, providing two-way paging-like services under 10 kbps.
- **Regional-Area Satellite Data Networks:** There are numerous low earth orbiting (LEO) satellite proposals on the drawing boards for the latter part of this decade, planning to offer low data rate services (Teledesic claims it will support multi-megabit rates, but cost is unclear). Deployed Direct Broadcast Satellite (DBS) and Very Small Aperture Terminal (VSAT) services provide shared high data rate downlinks. However, these incur relatively high latencies due to geosynchronous orbit transit times. Furthermore, for DBS the uplink is over non-satellite links, adding some complexity to the system design. The combination of asymmetric communications over hybrid links yields special challenges for the applications developer.

It may appear that the concept of multiple overlay networks will be seriously impeded by the mobile host's need for multiple transmitter/receiver systems. Yet even today it is possible to simultaneously configure a laptop with network adapters for in-room diffuse IR, in-building RF, campus-area packet radio (connected to the serial port) and wide-area CDPD (replacing the floppy drive). The rapid development of multimode radios, for example as described in [15], will

likely yield radios that can integrate such alternatives into a more convenient package.

4 GATEWAY-CENTRIC NETWORK MANAGEMENT

4.1 Introduction

We believe that IP and TCP/UDP will provide the basic network and transport layers upon which overlay and wireline networks will be integrated. Nevertheless, new interfaces are needed that reveal more about the underlying performance capabilities of the networks to applications. Effective management of overlay networks requires protocol extensions that reach down to the network layer. To build the extendible and scalable mobile information systems, able to deliver high quality connectivity to mobile applications over the wide area, the following challenges must be addressed:

- *Seamless Integration of Overlay Networks:* No general network management architecture exists for effectively integrating multiple overlay networks. Mobile applications roaming across overlays requires network intelligence to determine that the mobile has moved from acceptable coverage in one network to better coverage in another. But a global network management algorithm is still needed to control handoffs across overlays based on current mobile connectivity. Link quality is only one metric that determines handover; priority of access, applications needs, and relative cost are equally important. Since overlays may not cooperate with one another to render such decisions, it is crucial to support *mobile assisted handoff* in which the mobile host must be an active participant in handoff processing.
- *Support Services for Mobile Applications:* Handover across overlays will change an application's network bandwidth and latency. Needed is a *new applications interface to the network management layers* to allow them to initiate handovers, to determine changes in their current network capabilities, and to gracefully adapt their communications demands. Mobile applications and scalable wireline processing and storage capabilities could be better integrated through agent processing architectures that exploit data type specific transmissions to manage the communications demands over dynamically varying wireless links.

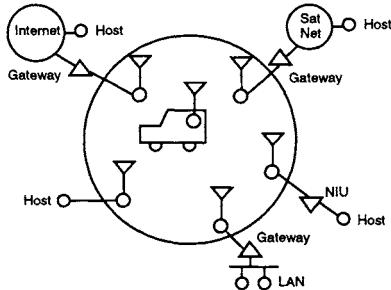


Figure 2 Network-Centric View

- *Managing Mobile Connections to Support Latency-Sensitive Applications:* Handoffs must be executed with lower latency than is now possible if (near) real-time multimedia applications are to be well supported. One strategy moves the routing and resource allocations to local subnets. For example, roaming authentications can be cached locally to avoid repeated remote, latency-intensive transactions. Needed are *algorithms that exploit information* about the location of mobile devices, the geographical adjacency of cells, and the likely routes devices might take, to improve handoff processing. End-to-end strategies like Mobile IP provide routing, but fall far short for latency-sensitive connection-oriented services. More hierarchical approaches, which localize information collection to the region or the subnet containing the users, are more scalable.
- *Load Balancing for Scalable Mobile Processing:* Repositioning within future wireless networks will be a common event. Traffic patterns will not be uniform, with high correlations between mobile host location, their repositionings, and their requests for service. Needed are network management *architectures that build on decentralized algorithms* to allocate network and processing resources on demand, avoiding the static and centralized schemes of the past. Furthermore, overlay networks provide an opportunity to share bandwidth and processing across networks; current network load is one reason to initiate internetwork handoff.

4.2 The Gateway-Centric Architecture

The design of conventional wireless networks places the mobile host at the center, with gateways to other networks placed around the boundaries of the wireless subnetwork. Figure 2 illustrates this concept for a packet radio net-

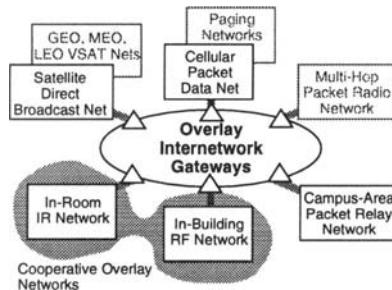


Figure 3 Gateway-Centric View

work. Such an architecture requires that the wireless network be homogeneous in terms of technology, i.e., air interfaces or network protocols, an assumption that is certainly not operative in commercial wireless networks, nor one that is likely to be the case in many hastily deployed tactical environments.

We describe our approach, on the other hand, as *gateway-centric* (see Figure 3). In the gateway-centric view, the subnet-to-subnet gateways are placed at the center of the architecture. Diverse wireless (and wired) networks are integrated through software that mediates between the mobile and the networks it could possibly connect to, supporting the mobile as it roams among the multiple wireless networks. The figure shows specific instances of commercially available wireless networks we are building into our gateway-centric architecture; those in black are networks we currently have access to, those in gray are other networks of interest. We are considering these other networks, such as the militarily important multihop packet radio networks and the emerging multiple satellite systems (MSSs) scheduled for deployment in the latter part of this decade, during the design of our overlay internetworking architecture.

The figure also introduces the important concept of *cooperative* wireless overlay networks. Most wireless networks, deployed by competing service providers (or in the military context, by somewhat distrusting coalition “allies”), provide little controllability above the subnet for routing, network management, or applications support purposes. We call such networks *black pipes*, because packets transfer between the wireline gateway at one end and a wireless link to/from the mobile at the other with no control over routing, priority, quality of service, etc. Cooperative networks, on the other hand, provide greater visibility of the network control functions to the gateway as well as application-level software, enabling management software to balance the network load among

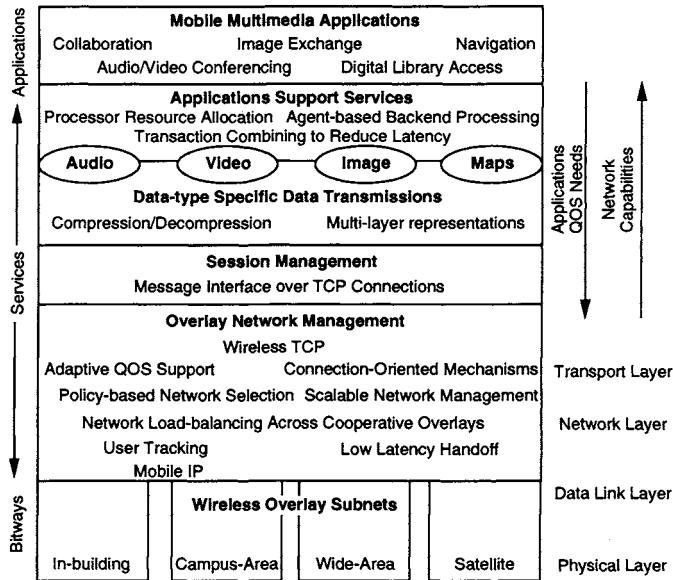


Figure 4 Layered Architecture

the cooperating networks or to choose a particular subnetwork for priority applications, etc.

4.3 The Architecture in More Depth

Figure 4 provides a conceptual architecture for wireless overlay internetworking and mobile applications support services. Although the functions are arrayed in layers, we do not believe that a strictly layered architecture is possible. For example, applications support is likely to need hooks into the transport and network layers (see Section 5 for more details). Working bottom up, the layers and the functionality they provide are described in the following subsections.

Wireless Overlay Subnetwork Layer: This corresponds to the physical, data link, and network layers in the OSI terminology. It represents the independent wireless networks we are integrating into our overlay architecture. These networks interface to the rest of the architecture through the standard Internet IP protocol suite. In general, the details of the underlying physical channels, media access protocols, link-level protocols, and routing protocols are not visible

above this layer, because these are provided by a commercial network service provider, our so-called “black pipe” networks. There is no way for us to take advantage of them at the next level up, the Overlay Network Management Layer. However, for in-building RF and IR networks, we have the ability to fully expose the underlying protocols, including those that control handoff decisions.

For example, IR provides advantageous cell confinement, finer location tracking, better spectrum reuse, and higher aggregate bandwidth with the potential for higher raw bandwidth. IR is particularly well suited for confined spaces, like offices and meeting rooms. However, handoff between IR cells is not easily accomplished, especially when crossing into large open spaces like corridors. An RF overlay advantageously provides connectivity for these gaps in IR coverage, as well as providing an alternative source of bandwidth for areas of heavy traffic. We can exploit this kind of exposed information, such as the degree to which network resources are already allocated to mobile hosts, to demonstrate the capabilities of cooperative management of overlay networks.

Overlay Network Management Layer: The design and implementation of this layer are crucial for the success of the wireless overlay network concept. This corresponds to the network and transport layers in the OSI terminology, with extensions to manage the multiple overlay networks. It builds an internetwork on top of the independent wireless (and wired) subnetworks. It must handle the complex issues of routing packets across heterogeneous subnets, while also choosing the most appropriate wireless network to provide end-to-end connectivity to the mobile host given the application’s specifications for quality of service.

Existing protocols, like Mobile IP, provide an initial solution for the mobile host routing problem. However, Mobile IP was not designed to support continuous media streams nor to provide low latency handoffs. The protocols must still be extended to exploit user tracking and local geographic information, to better support low latency handoffs within and between networks with exposed routing control, and must be better integrated with the emerging protocols for supporting quality of service concepts for real-time and near real-time data streams.

Tracking users and managing their handoffs in regions of high user density generates considerable additional processing load. The key to maintaining low latency in the network management algorithms is to design these algorithms to exploit scalable processing techniques, such as harnessing network of workstations (NOWs). This layer of the architecture should take full advantage of

the distributed processing power of NOW clusters, especially for in-building networks.

Existing handoff algorithms have been designed for homogeneous wireless sub-networks. Algorithms to support handoff between heterogeneous subnets are considerably more complex. They will need to base the handoff decision on application-specified policies as well as periodic measurements of the quality of the underlying network connectivity by the mobile host.

These policies can constrain the feasible handoffs. For example, an application might specify that high priority traffic traverse the available connections with the lowest latency, regardless of cost. Less critical traffic might be constrained to travel over the lowest cost connections currently available. Clearly the goal is to get the data through, so issues like signal quality, bit error rates, and the resulting probability of packet loss and retransmission must be considered as part of the handoff decision process. Under certain conditions, the handoff algorithm might defer switching its connection to a higher latency channel, even though it has better signal quality due to stronger signal strength, from a low latency channel with a weak signal.

Maintaining multiple powered-on network interfaces is an expensive proposition for the mobile host. For reasons of extending battery life, we assume that only one network interface is powered on and selected for providing connectivity to one overlay network at any point in time. Other interfaces will be in a standby mode, and powered up from time to time to determine the quality of connectivity to their respective overlays. A trade-off exists between the frequency of determining link quality, the power consumed by the operation, and how up-to-date the information is about the network state to drive handoff decision making. When the mobile is low on power, a good strategy is to connect to the network with the lowest transmitter power demands, and the handoff algorithm in the mobile may scale back the frequency of probing other overlays. System context may also dictate the frequency of probes. For example, as the mobile moves towards the fringe cells of its current network overlay, a good strategy would be to increase the probes to alternative overlays.

We are defining a rules-based languages for describing network selection policy. Such a specification will identify the application's relative priority of characteristics to drive the choice of a particular network for connectivity: highest raw bandwidth, lowest raw latency, lowest usage cost, best overall reliability of delivery (i.e., lowest packet loss), lightest network load, etc.

Within the network management layer, we must extend existing protocols to improve reliable transport over wireless links and exploit connection-oriented strategies to provide better support for real-time and near real-time media streams. Existing real-time protocols achieve their performance guarantees through admission-control procedures. Once a link becomes fully subscribed, no new connections are allowed to use that link. New schemes for link sharing are being developed to better utilize link bandwidth [7]. Yet even in these schemes, once a connection is granted to an application, it does not need to tailor its demand. These approaches are of limited use in wireless overlay networks, where applications must be able to adapt dynamically to dramatic changes in the quality of their connectivity as they move between networks.

Since entire subnets are “black pipes,” we must depend on end host adaptation to changes in the network characteristics, rather than strong support from the network, such as resource reservations. We must extend support for adaptation beyond the applications-independent protocol stack to the actual applications running on the mobile host. We are developing a strong coupling between the underlying networks’ capabilities and application demands. This is yielding new quality of service protocols, methods for characterizing network quality and availability of alternative connectivity, and call-back service alert mechanisms to the application.

Session Management Layer: This layer provides applications-independent session mechanisms. One example of a very useful session level mechanism is a message-oriented interface built on top of a single TCP connection. Without such a capability, applications like the World Wide Web that use TCP, but need a message-oriented interface, are forced to use a separate TCP connection for each message. The inefficiencies of the standard approaches are considerable.

Data Type Specific Transmission Layer: This layer provides specialized functions for applications to manage the transport of data type specific objects. Multimedia applications process audio, video, image, and other semantically rich data types. We are implementing mechanisms to support alternative representations and compression/decompression schemes for commonly encountered data types for multimedia applications to allow a flexible trade-off between bandwidth and latency, based on the network selected for connectivity and the relative cost of using that network, such as network load and billing costs.

Applications Support Services Layer: This layer provides resource management and distributed processing services that allow applications to migrate computation between the mobile’s environment and the backend processing environment. The quality of network connectivity, in terms of bandwidth and latency,

as well as the mobile host's processing capabilities and available battery life, determines how aggressively to partition the application. The management of backend processing, choice of data representation, and level of compression are determined by agents running on behalf of the mobile host in the backend environment.

Certain application-specific mechanisms can also yield improved efficiency. One idea is to combine multiple TCP connections into a single connection to reduce round trip latency. For example, all image link references within an HTML document could be packaged into a single TCP connection, thereby incurring a single round-trip network delay for a document spanning multiple images. This is particularly effective when the network exhibits highly asymmetric communications paths or particularly long latencies, which is often the case for satellite communications channels.

Mobile Multimedia Applications Layer: This layer consists of the actual mobile applications exploiting the underlying services and network management functions. We expect to reuse major pieces of code across a diverse collection of applications of conferencing, collaboration, and navigation. These code fragments will form the basis for a library of mobile computing modules.

5 OVERLAY NETWORK MANAGEMENT

We are developing new protocols, algorithms, and interfaces to enable continuous connectivity, low-latency handoffs within and between overlays, network load sharing between overlays, and dynamic reallocation of network resources to areas of high user density. This is being achieved by user tracking, vertical handoffs, overlay cooperation, and connection-oriented strategies.

5.1 User Tracking, History, Geography

Network management exploits the location of users and their physical environment to yield low latency handoffs and better allocation of resources to high traffic areas. Particularly inside buildings, it is possible to use the layout to localize the collection and analysis of tracking information, and to drive the handoff algorithms. Wide-area overlay networks are not as precise in their ability to locate users; nevertheless cell sites fall into a natural hierarchy con-

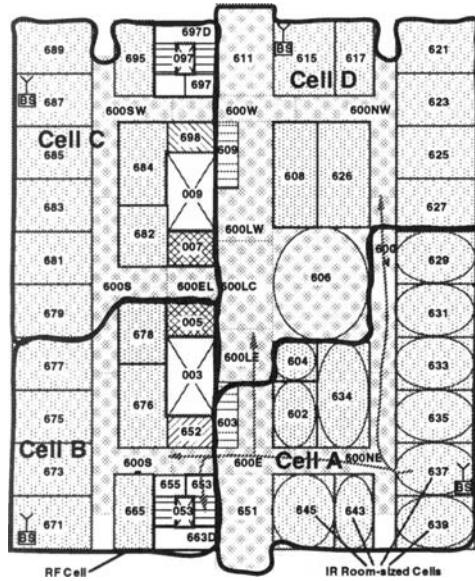


Figure 5 Exploiting Geographic Constraints to Assist Handoff Processing

strained by geographical proximity. This can be exploited for limiting the regions over which user information is collected.

Consider a mobile host receiving a continuous media packet stream while moving through a building. Its path is constrained by the building's physical layout. Figure 5 shows a floor of a typical building, covered by four RF cells. It is not possible to traverse directly from Cell A to C.

Knowledge of cell adjacency and likely routes enables “soft handoff.” Packets routed to the mobile in cell A is multicast to the cells it can reach, i.e., B and D. As the mobile comes within range of one of these, the network chooses when to shift the routes from A to the new cell. The new basestation will have already been “pre-charged” with the packet stream. See Figure 6. At the modest expense of additional buffering, backbone network bandwidth, and inter-basestation control communications, the network control is given flexibility in choosing when to handoff.

The tracking and handoff control processes run on processors in the backbone network and potentially cover multiple cells. To reduce the processing load, these could be allocated to highly interconnected “regions” of the building, such

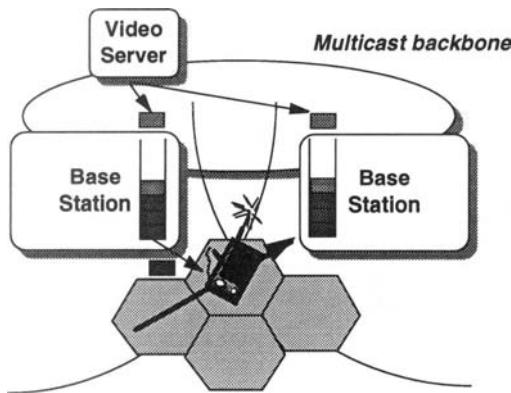


Figure 6 Multicast-enabled Soft Handoff

as a single corridor and adjacent offices, chosen so that inter-regional handoffs are significantly less frequent than intra-region handoffs. One could statically assign a tracking/handoff process per floor, with multiple processes allocated to different backend processors for those floors with particularly high traffic areas (e.g., conference rooms, classrooms). More generally, we are developing load balancing algorithms for network management that allocate processing resources based on the network's dynamic traffic patterns. For example, the algorithms provide more processing cycles to manage handoff at the entrance and egress areas of the building at the beginning and end of the day, the classrooms in the morning, and the seminar/meeting rooms in the afternoon. These algorithms must be extensible, accommodating new users and cell sites dynamically, without requiring global reconfiguration.

Support for "Vertical" Cooperative Handoffs: Conventional wireless handoffs are *horizontal*, i.e., within a homogeneous wireless subnet. The mobile host or associated basestation detects a degraded signal as it reaches the fringe area of its cell. In *mobile-assisted handoff*, the mobile listens for beacon signals from basestations in adjacent cells, choosing to register with the cell with the strongest signal.

Vertical handoffs are new, allowing mobile hosts to roam between overlays (see Figure 7). The mobile host, or higher level network management, determines when to switch the connections to an alternative overlay network, driven by signal quality, network load, or the costs of using one overlay versus an alternative. A rules-based mechanism for allowing applications to define their preferred connectivity is one avenue we are exploring.

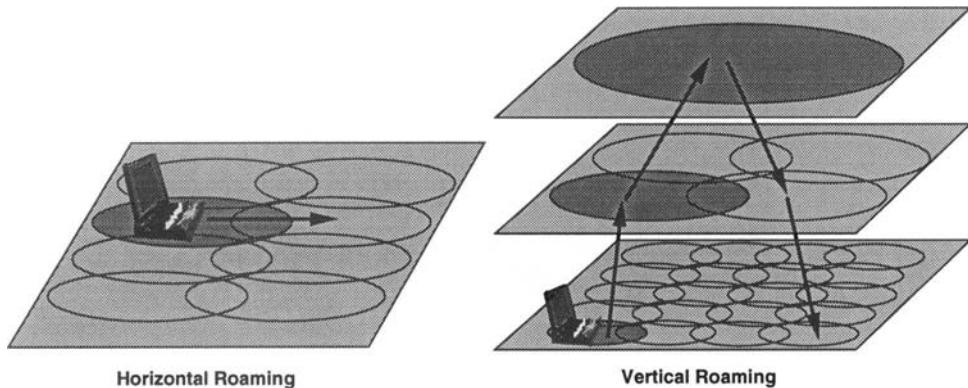


Figure 7 Roaming in Wireless Overlay Networks

When moving between overlays, registration and authentication latencies can be considerable. This can be reduced if they are cached, for a limited time set by policy, within the subnet gateways, thus reducing multiple, high latency authentications back to mobile's home system.

For in-building networks, we selectively violate strict network layering to expose information from below the network layer to higher level overlay management software. This allows wireless subnets to cooperate in order to reduce vertical handoff latency as well as to better share connectivity and network loading between alternative overlays. Our cooperative overlays are formed from independent in-building IR and RF physical networks. They are treated as logically independent networks that can only be integrated through overlay management.

Consider a group of mobile users in a meeting room. The in-room IR network provides the preferred connectivity. But if its bandwidth becomes oversubscribed, resulting in poor service to the mobile hosts in the room, some hosts can be forced to handover to the RF network. Furthermore, since the overlays are cooperating, new connections can be set up in the RF overlay in advance of actual handoff, thereby ensuring that the handoff is executed with low latency. Several management mechanisms are possible. The IR network could refuse new connections to mobiles in the high density area, generating mobile-initiated handover to the RF network. Or the IR network management algorithms could "request" assistance from higher level management layers, with a more global view of the condition of the subnets under its control. Certain mobiles could then be selected for overlay handoff based on their current network performance needs.

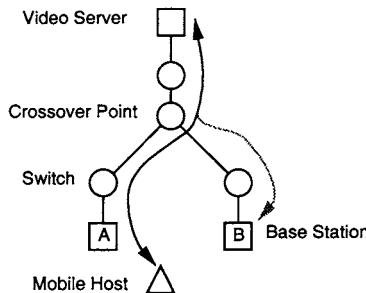


Figure 8 Connection-Oriented Strategies

Connection-Oriented Services for Mobile Hosts: Connection-oriented services with performance guarantees provide useful network support for multimedia applications [6]. The guarantees are typically achieved by reserving network resources in advance. The dynamic nature of mobile handoffs introduce complications; it is impractical to reserve all future channels. Rather than tear-down connections only to rebuild them, we pursue an incremental strategy that modifies existing connections by partially re-establishing them after a horizontal handoff. This method exploits the locality of logically adjacent cells to limit the amount of work involved to re-establish the connection. Figure 8 shows the situation for adjacent cells A and B. Since the established channel from the packet stream source to either cell largely are along the same route, only the tail of the connection needs to be rebuilt.

For “black pipe” networks with no performance guarantees, it is still desirable to attempt to characterize end-to-end network performance. The network management layer will cooperate with the mobile hosts to inject periodic measurement packets into the subnet to characterize the route’s latency and variability. This source-based rate control mechanism will be used by the application support layers in the mobile host to determine how aggressively it can pursue buffering, prefetching, and compression strategies. Wireless spectrum is far from “free,” so readahead should be applied only when the application can expect a high hit rate in cached data.

Connection-oriented TCP performance can also be improved for wireless links. It is well known that the standard protocol interprets lost packets as congestion, employing mechanisms that unnecessarily decrease the rate of transmission. Our approach caches the unacknowledged TCP packets in the basestation while performing aggressive local retransmissions over the wireless link by monitoring duplicate acknowledgements from the mobile host [2]. It requires no changes

to TCP's semantics nor to any implementations of TCP in the wireline portion of the network, except at basestations under our control.

6 APPLICATIONS SUPPORT SERVICES

6.1 Network Servers

Mobile applications will demand access to the same remote computing power now available from desktops. An attractive, cost-effective model views the aggregated computing resource as shared among a community of mobile and desktop users. We believe that the Berkeley NOW server architecture is the most flexible way to organize the shared, scalable remote computing resource.

How best to exploit this backend computing resource varies with the performance and reliability of the wireless link (see the next subsection). Nevertheless, we believe that shared computing power can enable very powerful mobile services. For example, the Berkeley InfoPad exploits pooled computing power to permit a small number of workstations to support a large number of end users. And unlike a distributed system cobbled together from laptops, it is much easier to share processing and memory resources on behalf of many users in a NOW.

For cases where the network performance is poor, we will use the network servers to execute agents on behalf of the mobile units, and to perform application- and data-specific compression before transmission over the wireless link. Agents mitigate poor latency by reducing the number of link crossings, while compression increases the link's effective bandwidth. Both of these strategies can require substantial cycles from the network servers.

6.2 Uniform Architecture for Applications Support

The support architecture and development toolkit will support "data type aware" mobile applications, characterized by access to multiple databases, media-rich documents, and global network access. Initial approaches are already being investigated by the InfoPad research project, but these assume high-quality connectivity. We are developing new mechanisms that dynam-

cally adjust the application's bandwidth and latency expectations. Each mobile device has a *proxy*, which is a process running on the NOW, responsible for managing the wireless connection, deciding on its appropriate level of compression and encryption, and performing computation on behalf of the mobile client both interactively and in the background. The proxy has the same access rights and security privileges as the mobile end user.

The architecture relies on strongly typed transmissions. A dynamically extendable type system enables type-specific compression levels and abstraction mechanisms for progressive object transmission, thus husbanding link bandwidth. For example, depending on link quality, we could send a raw, compressed, or lossy/highly compressed bitmap to a mobile client. As an extreme example, we can perform edge detection and OCR on the bitmap of a topographic map, only transmitting the text and edges. This *distillation* process produces a summary version that can be used to evaluate the value of the full bitmap; in the case of the topographic map, the distilled version is usually sufficient for making decisions. These techniques exploit the backbone compute servers to trade software latency to compress for increased effective bandwidth, thus reducing overall latency. The compressed forms can also be cached to further increase their benefit.

The development kit is based on Tcl/Tk. We are extending Tcl/Tk to support (1) strongly typed connections with data-specific compression / abstraction, (2) continuous media, and (3) Internet, FAX, WWW and SGML access. For example, we are developing a range of bandwidth/computation trade-offs for each data type, distinguishing between *versions*, which are independent, from *layers*, which are part of one version and form a sequence of increasing detail useful for progressive transmission of objects. For example, a PostScript object has at least two versions: ASCII text and full PostScript description. Images in the document, such as a map, use a corresponding type known to the application support subsystem. A map data type could exploit layering for progressive transmission, first sending the major roads and landmarks, and adding layers of increasing detail with each subsequent transmission. Layers/versions can have meaningful names; an application can ask for specific layers, such as a map's "elevation lines" or "county boundaries."

We are also developing cost models based on link instrumentation and statistical model fitting. These techniques, which we first developed for mapping parallel programs onto a variety of platforms [3], can accurately predict the overall latency for each transmission option for a given data type. The models include the link latency and the compression/decompression time, and exploit knowledge of the cache contents. We can decide the optimal format by sim-

ply evaluating the models for each level and picking the one with the lowest predicted latency. The models typically pick the first version to transmit; subsequent versions are controlled by the user. We will also experiment with predictive models to minimize access charges, by selecting the cheapest network and data representation.

The application architecture also provides for prefetching (also known as *speculative downloads*) and client-side caching. Prefetching reduces latency in exchange for bandwidth, which is especially useful for asymmetric communications like satellite downlinks. Client caching is advantageous for high-latency or low-bandwidth links avoiding round trips at the expense of more client memory. In fact, the user code on the mobile device will be cached using these mechanisms, so that the clients can dynamically download missing decompression or decryption functions as needed.

We can reduce required bandwidth and round trips by allowing applications to move computation across the wireless link via agent programs. An agent is just an instance of a small application, so it has access to the same strongly typed connections as any other, and can use the same mechanisms to adjust its behavior depending on dynamic link quality. In fact, user code is just another transmission type and can be sent in either direction. A key issue is the ability to download untrusted code safely.

We will employ two approaches to code transmission. First, we support scripts written in Safe-Tcl, a subset of tcl that is safe because it has very limited OS and file-system access. Safe-Tcl is good for simple agents written by the end user.

Consider building an application that combines information from multiple heterogeneous databases, using a full-text web indexer and on-line library card catalog. Typically the end user must do the combining. Alternatively, an agent running on the representative can access, combine, and filter the requested information. The agent uses the card catalog to find relevant documents and the web-indexer to determine which are on line. The end user then receives an annotated reference list with hypertext links for the documents. The wireless link is only traversed to start the search and receive this list; alternatively the results could be returned along an alternative path such as e-mail or FAX. The agent could prefetch abstracts or whole documents should the end user decide to view an on-line document. Other examples include combining maps and intelligence reports from multiple sources, and combining patient information with medical databases.

The second approach for code transmission involves dynamically translating and linking agent code written in a canonical assembly language. We use tokenized MIPS assembly as our download format, since it is easy to generate quality code from C/C++ using gcc as a cross compiler. Given code in this format, we then perform two transformations (in one pass). First, we convert the code to native assembly for the mobile device; this is manageable in part because of the simplicity of MIPS assembly. Second, we use *software fault isolation* to add native instructions that prevent the code from reading or writing outside of its assigned memory regions, or from executing unauthorized OS/library procedures. Thus untrusted code can affect only a predetermined range of memory and set of procedures. The transformation can occur on the proxy if the downlink is trusted (possibly through public-key authentication).

This approach enables arbitrarily sophisticated agents, since we can write efficient agents in C/C++ or other languages. This is crucial for adding new decompression/decryption modules dynamically, since such modules tend to be CPU intensive. For example, it would not be practical to write an MPEG decoder in Safe-Tcl. There is a second more subtle advantage, which is that only the protocol skeleton needs to be permanently installed on the mobile device. Thus, sensitive code such as custom decryption software can be downloaded as needed, which implies that lost or stolen mobile devices need not contain any proprietary or classified information.

Finally, the proxy architecture allows uniform handling of disconnection, since it maintains state across sessions. The network management layer will notify the applications support services layer of a disconnection, whether intentional or not. This allows the services layer to store results until the user reconnects without any help from the application. This is particularly useful if a connection is expensive or unreliable, since the user can start an agent (on the proxy) and reconnect later to get the results. This also gives us more freedom in implementing fast handoffs; if we temporarily lose the connection during a handoff, there are no adverse effects and applications will in fact not even know (unless they asked to be notified).

7 RELATED WORK

7.1 Packet Radio Networks

Driven by the military's need for communications in regions with poor infrastructures, ARPA initiated its pioneering packet radio program in the early 1970s [12, 13]. The original program focused on algorithms for self-configuring, self-managing packet switch networks with support for mobile nodes. The ARPA Survivable Adaptive Networks (SURAN) program of the 1980s focused on eliminating vulnerabilities, such as network partitioning, in packet radio networks. [14] reviews the critical packet radio network design issues at the physical, data link, and network layers, while [10] describes the implementation of the routing and network management protocols. As part of these efforts, algorithms were developed leading to the Reconstitution Protocol (RP), which manages the partitioning and coalescing of networks while maintaining communications to mobile hosts. Researchers at SRI have recently proposed RP as an alternative to Mobile IP to support internetwork routing to mobile nodes [5].

Packet radio algorithms for link determination, discovery of network topology, packet routing, and route dissemination assume a homogeneous wireless technology. Wireless overlay networks introduce the additional flexibility (and complexity!) of alternative physical links, with varying performance, along which to establish network connectivity and to route packets. For example, the low cost ARPA packet radio could downgrade its transmission from 400 kbps to 100 kbps when operating over poor quality radio links. With overlay networks, an alternative strategy is to switch connectivity to a different wireless subnetwork with higher signal quality even though the network exhibits worse performance metrics, such as lower bandwidth or longer latencies. Furthermore, overlays may be able to reconstitute partitions in one network by providing connectivity through an alternative overlay network. The gateway-centric approach described in [14] has influenced our conceptual architecture. New kinds of network management algorithms, along the lines proposed in this paper, are needed for the heterogeneous environment of overlay networks.

7.2 Wireless Data Technologies

We review some of the wide-area data services we have access to in the San Francisco Bay Area. Metricom is bringing to market a wireless IP packet relay network, based on wireless modems, poletop radios, and wired access points to

the Internet. Cellular Digital Packet Data (CDPD) is a commercially available cellular data overlay delivering a medium rate, "moderate" latency IP packet service [17]. It is offered by GTE Mobilenet in the S. F. Bay Area. Nextel Communications, Inc., whose underlying technology is based on the Motorola Integrated Radio System (MIRS), offers enhanced special mode radio (ESMR) services throughout the West Coast, the Northeast, Florida, and Texas. The technology supports digital voice and data, including acknowledgment alphanumeric paging (i.e., messages are held and retransmitted until acknowledged by the mobile handset). Although the initial data offerings are circuit-switched, future packet-switched services, likely to be similar to CDPD, are being planned. The service providers understand the importance of providing connectivity to customer enterprise networks via the Internet.

Pacific Telesis Corporation has recently acquired PCS spectrum licenses in the San Francisco Bay Area, as well as in Southern California. The evaluation of alternative technologies is currently underway, with initial network services to be deployed in 1996. A likely candidate technology is based on the European digital cellular standard, GSM. Packet data services for GSM, called GPRS, are still in the early stages of definition. Existing proposals from Motorola and Nokia are similar to CDPD, but provide somewhat lower data rates.

The International Telecommunications Union (ITU) is moving forward with its effort to define Future Public Land Mobile Telecommunications Services (FPLMTS), which are meant to seamlessly integrate cordless and cellular telephone overlays with emerging satellite communications services. A similar Universal Mobile Telecommunications System (UMTS) is being defined by researchers in the European Community countries [4]. Both efforts are focused on voice rather than data services, but eventually packet data will no doubt be included in these evolving standards.

The Hughes Direct Broadcast Satellites are currently in operation, providing coverage to most of North America. It is *highly asymmetric*, with a shared 12 mbps satellite downlink, a wireline Internet gateway uplink, and latencies greater than 500 ms. Individual users may achieve up to 400 kbps on the downlink. The system is wireless, but not mobile; its 24-inch satellite dish, though rapid to deploy, will not support communications on the move. DBS is attractive for wide-area data distribution or other applications in which more information can be usefully broadcast on the downlink than on the uplink. Many applications can take advantage of broadcast-based and asymmetric access, e.g., distribution to field depots of updates to maintenance manuals, maps, intelligence reports, etc., and access to the World Wide Web.

7.3 Mobile IP

Mobile IP addresses the mobile routing problem, providing connectionless packet service as nodes move through internetworks. Existing implementations introduce intolerable latencies during handoff, making it difficult to support continuous media streams across such repositionings. The protocols take no advantage of local information to reduce handoff latency or to support media streams across handoffs. Variations of Mobile IP have been proposed to improve its performance, using a variety of short cut and tunneling schemes (e.g., [9, 16, 18]), but these have not as yet been included in the IETF Mobile IP protocol specifications.

7.4 Scalable Network Processing and Handoff Algorithms

The cellular telephone system, with the network intelligence embedded in large-scale switching offices, represents the kind of centralized network model that is inherently difficult to scale. The Intelligent Network (IN) architecture attempts to package services like handoff management as processes, decoupling them from the network switching nodes. We believe that a more general distributed networking architecture, based on NOWs, will be inherently more scalable.

Algorithms for cellular handoff have recently gained in attention by the research community, motivated by the need to scale up handoff processing for the emerging PCS networks characterized by higher user densities, smaller cells, and increased frequency of cell crossings. For example, [1] has proposed a “region- alized” scheme for managing handoff in cellular networks. When a connection is established for a given mobile host, virtual circuits are created and associated with all nearby basestations within its roaming region. The mobile host then has the freedom to choose among any of these to establish its connectivity, without requiring the intervention of any call processor. The latter only becomes involved when the mobile crosses between regions. The scheme implements regions as a static concept, as a layer just above the basestation leaves. Load balancing is achieved only through coarse-grained admission control at the regional boundaries. We believe that a fully hierarchical approach, allowing arbitrary nesting of regions that could be defined dynamically based on user densities, would be much more attractive. Furthermore, load balancing at the leaf level of basestations will be crucial for reducing latency, since in our model, mobile hosts make heavy use of wireline processing resources.

The above approach focuses primarily on reducing handoff bottlenecks by reducing the frequency of interaction with the call processing or connection management components of the network. It makes no attempt to exploit local information or mobile tracking to reduce the latency of handoff. Initial work specifically focused on latency reduction for handoff has just begun to emerge. For example, [8] describes a scheme similar in concept to our approach: dynamic formation of multicast groups based on adjacent cells through which a mobile is likely to move, with continuous media packet streams buffered at all basestation members of the group. Knowledge about the physical structure of the space is used to define the groups, i.e., only physically reachable cells can be combined into a group. Although the proposed buffer management schemes do consider mobile velocity, coarse trajectories, and time within cell, the protocols do not attempt to exploit likely routes through the space of cells, which may vary based on the time of day. Furthermore, the proposed algorithms have no scheme for load balancing or processing resource allocation, which can become a serious issue in areas of high traffic and user densities.

7.5 Adaptive Applications and Network Performance Guarantees

Applications built on top of overlay networks must be able to adapt to dramatic changes in performance as the mobile client moves among alternative networks. Existing network support is not flexible enough. Early work on performance guarantees (e.g., [6]) used admission control algorithms to deny service when applications requirements could not be met. Recent research on packet scheduling and controlled link sharing [7] provides mechanisms for better utilization of links, but a dynamic renegotiation of quality of service, either down as more applications demand access to link or up as contention for the link is relieved, is still not supported. Because the bandwidth- and latency-adaptive applications we will develop for our overlay networks, we are in a strong position to develop more dynamic mechanisms for specifying and negotiating quality of service guarantees in the underlying networks. Upcall mechanisms from the network layer are clearly needed, to alert applications to changes in their network connectivity.

8 SUMMARY AND CONCLUSIONS

In this paper, we have described the wireless overlay network concept as a way to combine the advantages of wide-area coverage while simultaneously achieving the best possible bandwidth and latency for mobile devices at any point in time. If we can demonstrate effective, low latency handover between overlays, we believe that whole new classes of compelling wireless applications will be enabled. Furthermore, we are developing support mechanisms that make it possible for applications to adapt to the changes in the quality of their network connectivity, either being changing data type representations or by moving the place where computations are performed. In our model, agents moderate these changes in network connectivity on behalf of applications.

Our network management architecture is being designed, deployed, and evaluated in a wireless overlay testbed being fashioned in the San Francisco Bay Area, with wireless technology and network services from AT&T, DEC, GTE Mobilenet, Hughes Aircraft Corporation, IBM, Metricom, Nextel, and Pacific Telesis. We call this testbed BARWAN or *Bay Area Research Wireless Access Network*. To the extent possible, we hope to discover what happens inside the “black pipes,” to better understand how they might be more cooperatively controlled by our overlay internetwork management algorithms.

We are exploring the possibilities of integrating BARWAN with the Bay Area Gigabit Network (BAGNet) or other high speed network testbeds being deployed in the Bay Area, such as the ARPA-sponsored 10 Gbps Fiber “Ring Around S. F. Bay.” We are seeking collaboration with other Bay Area sites mutually interested in mobile/wireless technology, such as SRI, Stanford University, Xerox PARC, and UC Santa Cruz, to integrate their research networks into the testbed, further validating its internetworking capabilities.

We hope to be able to make this testbed available to other members of the research community, to demonstrate their evolving mobile and wireless technologies. If successful, it will provide an ideal environment for experimenting with technology interoperability, especially between operational and experimental networks.

REFERENCES

- [1] A. S. Acampora, M. Naghshineh, “An Architecture and Methodology for

Mobile-Executed Handoff in Cellular ATM Networks," *IEEE J. on Selected Areas in Communications*, V 12, N 8, (October 1994), pp. 1365-1375.

- [2] H. Balakrishnan, S. Seshan, E. Amir, R. H. Katz, "Improving TCP/IP Performance over Wireless Networks," ACM Mobile Computing and Networking Conference, Berkeley, CA, (November 1995).
- [3] E. A. Brewer, "Portable High-Performance Supercomputing: High-Level Platform-Specific Optimization," MIT Ph.D. Dissertation, September 1994.
- [4] J. S. DaSilva, B. E. Fernandes, "The European Research Program for Advanced Mobile Systems," *IEEE Personal Communications Magazine*, V 2, N 1, (February 1995), pp. 14-19.
- [5] B. Denny, J. Escobar, Presentation on "Mobility" at DARTNet II Meeting, Xerox PARC, (March 1995).
- [6] D. Ferrari, "Client Requirements for Real-Time Communication Services," *IEEE Communications Magazine*, V 28, N 11, (November 1990), pp. 65-72.
- [7] S. Floyd, Presentation on "Packet Scheduling Result" at DARTNet II Meeting, Xerox PARC, (March 1995).
- [8] R. Ghai, S. Singh, "An Architecture and Communications Protocol for Picocellular Networks," *IEEE Personal Communications Magazine*, Third Quarter 1994, pp. 36-46.
- [9] J. Ioannidis, D. Duchamp, G. Q. Maguire, Jr., "IP-based Protocols for Mobile Internetworking," Proceedings 1991 ACM SIGCOMM Conference, pp. 235-245.
- [10] J. Jubin, J. D. Tornow, "The DARPA Packet Radio Network Protocols," *Proc. IEEE*, V 75, N 1, (January 1987), pp. 21-32.
- [11] J. M. Kahn, *et al.*, "Non-Directed Infrared Links for High-Capacity Wireless LANs," *IEEE Personal Communications*, V 1, N 2, (Second Quarter 1994), pp. 12-25.
- [12] R. E. Kahn, "The Organization of Computer Resources into a Packet Radio Network," *IEEE Transactions on Communications*, V COM-25, N 1, (Jan 1977), pp. 169- 178.
- [13] R. E. Kahn, *et al.*, "Advances in Packet Radio Technology," *Proceeding IEEE*, V 66, N 11, (Nov 1978), pp. 1468-1496.

- [14] B. M. Leiner, D. L. Nielson, F. A. Tobagi, "Issues in Packet Radio Network Design," *Proc. IEEE*, V 75, N 1, (January 1987), pp. 6-20.
- [15] J. Mitola, "The Software Radio Architecture," *IEEE Communications Magazine*, V. 33, N. 5, (May 1995), pp. 26-38.
- [16] A. Myles, D. Skellern, "Comparing Four IP-based Mobile Host Protocols," *Computer Networks and ISDN Systems*, V 26, (November 1993), pp. 349-355.
- [17] K. Pahlavan, A. Levesque, "Wireless Data Communications," *Proceedings IEEE*, V 82, N 9, (September 1994), pp. 1398-1430.
- [18] C. Perkins, "Providing Continuous Network Access to Mobile Hosts using TCP/IP," *Computer Networks and ISDN Systems*, V 26, (November 1993), pp. 357-369.

THE DIANA APPROACH TO MOBILE COMPUTING

Arthur M. Keller, Tahir Ahmad,
Mike Clary*, Owen Densmore*, Steve Gadol*,
Wei Huang, Behfar Razavi*, and Robert Pang

Stanford University, Computer Science Department, Stanford, California 94303

* *Sun Microsystems, Mountain View, California
USA*

ABSTRACT

DIANA (Device-Independent, Asynchronous Network Access) is a new application architecture to solve two major difficulties in developing software for mobile computing—diversity of user interface and varied communication patterns. Our architecture achieves display and network independence by de-coupling the user interface logic and communication logic from the processing logic of each application. Such separation allows applications to operate in the workplace as well as in a mobile environment in which multiple display devices are used and communication can be synchronous or asynchronous. Operation during disconnection is also supported.

1 INTRODUCTION

People want to access their applications not only while in the workplace but also while they travel, and while using a variety of devices. Although much current computer software can provide sophisticated functionality to ordinary users, the software rarely addresses the special needs of mobile users.

A number of issues limit the usability of software for mobile users. We consider two particular issues: user interface and communication. Conventional software is usually developed with implicit assumptions about the kinds of display devices users have and the communication media available between them and the users. For example, applications written for the X-windows protocol can only be used by those users who run X-windows across high-speed computer

networks. Such software is useless to mobile computer users with a different kind of communication environment.

Various computing devices tailored for mobile users are currently available, such as notebook computers, palmtop computers, and personal digital assistants (PDA). The user interface on these devices differs greatly. To make software accessible to these devices, application developers currently have to customize their software for each individual type of device. We anticipate an even greater variety of such mobile devices in the future and such diversity can put a significant burden on software developers.

On the other hand, the communication networks available to mobile users are still relatively slow and unreliable, as compared with the high-speed local area networks that connect workstations, servers, and mainframes. Also, various communication protocols are used for mobile computing and handling all these different protocols can be a significant software development cost. Moreover, mobile users may need to communicate intermittently because of the high cost or unavailability of communication.

To overcome these two hurdles in software development for mobile computing, we propose a new application architecture for mobile applications. The DIANA architecture—Display Independence and Asynchronous Network Access—decouples the display and communication logic of applications from their processing logic. The resultant applications will enjoy the benefits of being display and transport independent. Not only will new applications be able to take advantage of this architecture, but also we envision that existing applications designed for directly connected users on particular devices will migrate to our approach.

1.1 Background

Our work on this project started by looking at workflow systems closely and trying to understand why people had failed to use them extensively in the industry. Our study raised the following issues as hindrances in the applicability of such systems.

Platform Bindings. One desirable aspect of these systems is that they should be able to make legacy applications work together in a broader context than they were originally designed for. However, most of these legacy applications are bound very strictly to the platform for which they were designed. Porting

these applications to multiple platforms incurs significant development and maintenance costs.

Network Management. Both the legacy applications and the workflow management systems have embedded assumptions about the underlying networks and the available communication protocols that they will be using. These assumptions provide significant inflexibilities and limitations. Furthermore, networking across hardware platforms introduces complications such as security. Also, there may be the unavailability of similar operating system features on different platforms, e.g., a workflow system relying on RPC (remote procedure call) to implement triggers becomes ineffective if one participant system does not provide support for RPC.

Design Complexity. An inherent problem with workflow systems is the complexity of designing working workflow models. Most of the systems, in their bid for providing the designers maximum flexibility, put a lot of responsibilities on the designers. Designers have to manage the conceptual entities in their workflow model, such as roles, tasks, jobs. Designers also have to be aware of the heterogeneity of the physical resources their system will be using and the restrictions this heterogeneity will impose on their model.

Different Data Representation. Multiple applications comprising a workflow system often have different views of data, which makes it more difficult to develop a generic solution for inter-application workflow.

To address these issues, we developed the DIANA architecture. DIANA uses a universal, dynamic language understood by all the entities in the system (applications, users, and system components) that is based on the semantics of interaction. This language can be interpreted on multiple platforms, so that the same core application becomes accessible from a variety of heterogeneous devices. The network components of DIANA are capable of switching between synchronous (e.g., direct connect) and asynchronous (e.g., e-mail based) communication modes. Applications can access the responses from clients by using a simple information-level Application Programming Interface. By providing a single DIANA client to legacy applications, they can be made to interact with all other entities using the DIANA architecture.

1.2 Related Work

Considering display and network independence, we observe that Mosaic and Telescript [3] have close similarity with DIANA.

DIANA has a significant overlap with Mosaic as far as the interface is concerned. Mosaic uses Fill-Out Forms to express user interactions and query certain information sources distributed throughout the network. We have implemented the sample travel authorization application using Fill-Out Forms for a comparison between DIANA and Mosaic. From our experience of using Fill-Out Forms, we observe that one key difference between the DIANA approach and the Mosaic approach is that DIANA uses an interface language based on the semantics of the user interaction. On the other hand, Fill-Out Form language part of HTML assumes the presence of an access device with certain layout characteristics. This semantic representation of interaction in DIANA makes the system extensible to non-conventional access devices, such as telephones. However, we can extend FDL by incorporating layout preferences, such as those in HTML, without sacrificing display independence.

Considering network communication, the World Wide Web, and browsers such as Mosaic, adopts stateless communication between users and the application. Where transaction states are needed, they are kept by the application. In contrast, DIANA's Courier provides connection-based communication, and it keeps the states for the client applications. We believe this stateful communication can not only reduce setup time and bandwidth requirements but also facilitates caching information locally. In addition, the Courier provides both synchronous and asynchronous communication while Mosaic provides only synchronous communication currently.

General Magic's Telescript promises network and device independence. Unfortunately, very little information about Telescript is publicly available to permit a meaningful comparison with DIANA. A simplistic comparison is that DIANA handles the user interface similar to Mosaic and the network interface similar to Telescript.

Kazman et al., [9] present an overview of some user interface architecture models. The Seeheim model [13], the Arch/Slinky Metamodel, the Presentation-Abstraction-Control (PAC) model [4] and the Serpent model [1] have some overlap with the DIANA model in that they all attempt to separate the presentation of a user interaction from its functionality within the application logic. DIANA proposes a form based solution to implementing a user interface

architecture in which the core application logic is de-coupled from the interface logic. The applications in DIANA use a form based approach in which applications do not have to be responsible for fine grained rendering details as user's input or retrieve information to and from the forms. The fact that existing interfaces require most applications to control the presentation on a very interactive and fine scale is regarded as one of the reasons why human computer interfaces are hard to design and implement [12].

Imielinski and Badrinath [8] identify some of the issues involved in mobile wireless computing, including location management, configuration management, disconnection, cache-consistency, recovery, scale, efficiency, security and integrity. The paper also presents a model of a system to support Mobility. In this system, Mobile Units are assumed to have wireless e-mail access to Mobile Support Stations (fixed network hosts with a wireless e-mail interface to communicate with the mobile units). Most of our discussion assumes the presence of a similar system for the purposes of wireless e-mail communication.

The Coda File System [16] is a highly available file system designed to suit distributed and mobile computing. It uses an optimistic replica control strategy to provide high availability and relies on a dynamic cache manager to provide disconnected operation. DIANA proposes to use a similar caching strategy to minimize the network utilization and improve efficiency in addition to support disconnected operation. Details of various aspects of the Coda File System are available in [14, 15, 17].

Java [6] is a language for portable applications that may be disseminated among a variety of platforms. HotJava is a browser that allows dynamic fetch of Java applets over the Internet in a manner that extends the notion of the World Wide Web. HotJava works best in a connected network environment; it has no special features to handle disconnected or intermittently connection operation.

1.3 Outline

In this paper, we will describe the proposed architecture and explore the advantages of using DIANA architecture in mobile computing. We will also report on the status of our implementation and our experience in using DIANA. The organization of the rest of this paper is as follows. Section 2 describes the overall architecture of DIANA along with an illustrative application. Section 3 discusses how DIANA addresses the display independence problem. Section 4 focuses on the connectivity issues in DIANA. Section 5 describes the applica-

tion development methodology. Section 6 discusses the current implementation of DIANA. Section 7 discusses future work, followed by concluding remarks in Section 8.

2 DIANA—THE OVERALL ARCHITECTURE

This section presents an overview of the DIANA architecture. We define the key components of the system, describe their responsibilities and explain how they work together. Subsequent sections of the paper discuss these components in further detail.

2.1 User Interface Logic

In order to de-couple the user interface logic from the processing logic of applications, we define a component called the *User Interface Logic* (UIL) to handle generic user interface operation on the client on behalf of applications. As a component of the DIANA system, this UIL is independent of the specific applications. There is a different UIL for each different type of display devices. Each different UIL will implement the semantics of forms using the display characteristics appropriate for the particular display device. For example, applications will be able to communicate with the UIL for a X-windows display device as they will do so with the UIL for a PDA. The end result will be that users can use different display devices as clients to the same application without requiring the application to handle each of those devices separately.

In order to allow applications to communicate with the UIL in a generic way, we design a *Form Description Language* (FDL) for applications to express the types of user interface to UILs. This language is based on the form paradigm and focuses on the semantics of the information exchange between the applications and the users rather than its aesthetics. A Form or script written in FDL is called an *inFORM* (standing for *info-form*, hence the capitals). Since FDL focuses on the semantics of information exchange rather than the characteristics of display devices, applications which use FDL to describe their user interface can truly be display independent.

2.2 The Courier

The network manager in DIANA is called the *Courier*. It supports network independence through the support of multiple network protocols and both synchronous and asynchronous modes. The synchronous communication mode represents direct communication with potentially high bandwidth and low latency. On the other hand, asynchronous communication is used in the situation when communication is intermittent or has high latency. For asynchrony, a store-and-forward information exchange paradigm is more appropriate. Courier makes this kind of asynchronous communication transparent to applications. We will discuss these modes in more details in a subsequent section.

There are two components of the Courier: one resides on the network with the application (Application Courier) and the other resides on the access device (User Courier). The purpose of having these couriers is to provide an encapsulation of the underlying communication medium so that both users and applications can have the same processing logic independent of whether they are communicating in synchronous or asynchronous modes.

2.3 The Application Replay

In asynchronous communication mode, users may work asynchronously with applications during disconnection. In order to deliver inFOrms from users to applications in the sequence as they are generated, we define *Application Replay*, an agent on the applications' side that presents the inFOrms that were collected during disconnection to applications as if they were generated while continuously connected.

2.4 Other components

In order to reduce communication traffic, we add an inForm cache on the user device to store those inFOrms which are frequently used. It can also cache those inFOrms which the user may need during disconnection, thus allowing the user to continue to work during those periods.

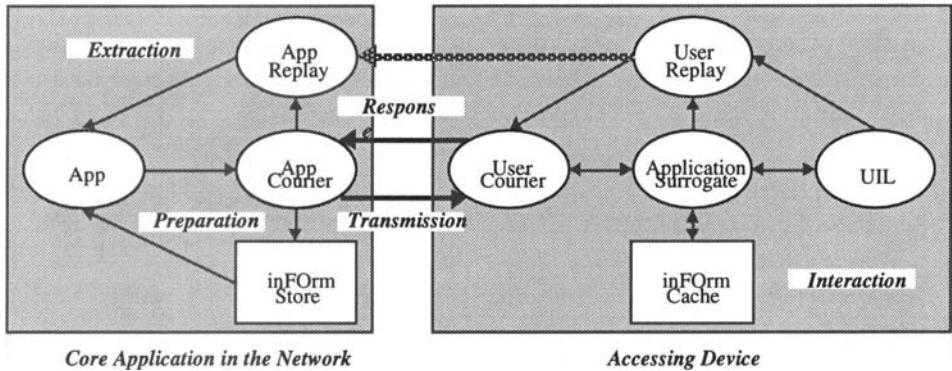
If the application has application-specific processing not encodeable in FDL that should occur on the client during periods of disconnection, then such processing can be embodied in an Application Surrogate that resides on each client device. We will discuss the application surrogate later.

The inForms for all applications are identified in a central registry and are stored in an inForm store. This approach allows users to search for inForms for specific purposes.

2.5 The Life Cycle of an inForm

Figure 1 describes the overall DIANA architecture. The following are the different phases an inForm goes through during its life cycle from preparation, transfer and processing.

Figure 1 DIANA: Overall Architecture and inForm Life Cycle



Preparation. When a transaction starts, the application first obtains an inForm from a repository (inForm Store) and optionally fills in some default data on it such as the user's employee number and his or her department code.

Transmission. The next phase involves the physical transmission of the inForm from the application to the user by the Courier. The Courier is responsible for connection detection and switching between directly connected operation and disconnected operation using e-mail. The details of the mechanisms used during this transmission phase are hidden from the application.

Interaction. The user interacts with the inForm in this phase, providing the necessary information by filling in the form. This process involves the UIL interpreting the inForm script and controlling the user interaction. The user

responds to the application by entering his inputs in the inForm rendered by the UIL on the user's display device.

Response. This phase is similar to the transmission phase, except that the flow is from the user's device to the application's machine. It involves transmitting the inForm replies to the Application Replay Agent.

Extraction. In this phase, the application receives the inForm reply from the Application Replay Agent and extracts the information from that form using the DIANA API. The transmission and Extraction phases are asynchronous from the core application's perspective. The application may then proceed to process the reply inForm and optionally reply to the user through another inForm cycle.

2.6 Application Example

Jeff Jones is a sales representative of ABC corporation. Due to the nature of his job, he often travels to meet prospective customers. Before going on a trip, Mr. Jones must use travel authorization software to submit a travel request. He uses the same application to view the results of his requests and clarify on his travel details if necessary.

By using the travel authorization software (we will simply call it the "application" in the following discussion), he first submits his travel request using a workstation running the X-windows system. The application first sends a travel request inForm to him via the Courier. In this situation, the Courier uses TCP/IP to transfer the inForm from the application to the client.

The UIL for X-windows on his workstation interprets the inForm and graphically renders it. He then enters his travel details and returns the reply form back to the application via the Courier using TCP/IP.

When he leaves his office to travel to his customer, he disconnects his User Courier from the Application Courier. The Application Courier, noticing this disconnection, forwards the inForms to him via electronic mail without affecting the application.

While away from his office, he uses a palmtop computer, a character-only device, to retrieve the status of his travel request. The User Courier first retrieves those inForms sent to him during disconnection from his mail box. They are

interpreted by the UIL for this character-only device. Then the User Courier resumes normal communication with the Application Courier over a serial line.

In this example, we have seen how a user access the same application with multiple display devices and connects to the application sometimes continuously and sometimes intermittently.

3 USER INTERFACE AND DISPLAY INDEPENDENCE

We consider applications that interact with users through a question and answer dialog. In this question-and-answer paradigm, applications are considered to send out questions organized as forms, called inFOrms in DIANA, and users reply to these forms. The notion of question forms provides an abstraction of user interfaces to applications. Instead of seeing the heterogeneity of access devices and user interface, applications see only an abstract interface for information exchange, but not the actual display devices. This form-based interaction helps to separate display mechanism from applications so that they are display-independent.

In fact, form-based interaction is analogous to form-based service provided by public service offices like the Department of Motor Vehicles. Someone applying for a driver's license will fill out an application form. In a similar way, users are requested to fill out inFOrms to provide inputs to applications that use the DIANA framework.

To accomplish display independence, inFOrms have no relationship with the display capability of user devices. They focus on the semantics of the information being exchanged, rather than the representation of the information on users' devices. By removing any ties they may have to any display devices, they become truly display-independent. The final representation of the information is left to the User Interface Logic which will be discussed in details later.

3.1 Form Description Language

To achieve display independence through the use of generic forms, DIANA provides FDL for applications to define their user interface as inFOrms. In Table 1, we show the list of input items provided by FDL for applications

to choose the items which best suit their input needs. More specifically, they specify the questions and the nature of the replies they expect. For example, an application may require text input to one question and an integer input within a range in another one. Notice the difference between the interaction items listed below and their representation by the GUI. For example, a one-from-many interaction describes the semantics of a question while a radio button is a possible implementation for displaying such a question. Figure 2 shows a travel request form as it appears on an OpenLook interface and Figure 3 shows the input that created this form.

Notice that some of the interaction items might not be renderable on some devices. Users who wish to see or hear such items will need to use suitable devices.

Figure 2 A Travel Request inForm on an OpenLook Interface

The screenshot shows a window titled "Travel Request Form". Inside the window, there are several text fields and a set of radio buttons. The text fields contain the following information:

- Employee Name:** Jeff Jones
- Employee Number:** C-43382
- Date of departure:** 9/27/94
- From:** San Francisco, CA
- To:** Austin, TX
- Cash Advance Required:** \$300

Below the text fields is a row of radio buttons labeled "Type of Travel" with options: Business, Training, Conference, and Other. The "Conference" button is highlighted. At the bottom of the window are two buttons: "Apply" and "Cancel".

3.2 User Interface Logic

Rendering. The User Interface Logic (UIL) interprets inForms received from applications on behalf of end users. UIL renders inForms on users' devices according to the display characteristics of the devices. UIL is also responsible

Type	Properties	X-windows Interface	DOS Interface	Telephone Interface
<i>Text input</i>	Allows user to enter a string	Text Input Widget	String printf and scanf, or text input fields	Touch-tone encoding (or speech rec.)
<i>Command</i>	UI Logic returns to application when user chooses a command	Command Buttons in Xview	Initial character represents the command	Use dial buttons (or speech recognition) to choose command
<i>Menu</i>	A group of commands for user to execute	Menu in Xview	Initial character as command	Dial buttons (or speech rec.) to choose command
<i>One-from-many input</i>	User can choose only one option. Options may change dynamically	Radio Button in Xview for short list, scroll list for long list	Radio Buttons on an ASCII screen for short list, scroll list for long; highlight chosen option	Use dial buttons (or speech recognition) to choose option
<i>Multiple-options input</i>	User can choose multiple options; options may change dynamically	Option Button in Xview for short list, scroll list for long list	Option Buttons for short list, scroll list for long list; highlight chosen options	Use dial buttons (or speech recognition) to choose option
<i>Yes/No</i>	User answer yes or no	Yes and No buttons	Y or N key input	2 special dial buttons
<i>Bounded Input</i>	User can only input a value within a range	Scroll Bar in Xview	Text input w/bound, or graphical scroll bar	Dial buttons – “#” as “enter” (or speech rec.) with range-checking
<i>Date/Time Input</i>	Enter date/time; UI Logic check for validity of entry	Text Input, with arrows, or menus	Input fields with arrows, or menus	Use dial buttons to choose from a list (or speech recognition)
<i>Simple Display</i>	For display only	Text display in Xview	Text display	Speech Synthesis

Table 1 Device-specific interpretations of the types of user interaction

Figure 3 A Travel Request inForm

```

BEGIN FORM newRequestForm
  HEADING "Travel Request Form"
  BEGIN SIMPLEOUTPUT empName
    HEADING "Employee Name"
    DATA <TEXT> "Jeff Jones"
  END SIMPLEOUTPUT
  BEGIN SIMPLEOUTPUT empNum
    HEADING "Employee Number"
    DATA <TEXT> "C-43382"
  END SIMPLEOUTPUT
  BEGIN ONEFROMMANYFIXED travelType
    HEADING "Type of Travel"
    BEGIN OPTION business
      HEADING "Business"
    END OPTION
    BEGIN OPTION training
      HEADING "Training"
    END OPTION
    BEGIN OPTION seminar
      HEADING "Conference"
    END OPTION
    BEGIN OPTION other
      HEADING "Other"
    END OPTION
    DEFAULT business
  END ONEFROMMANYFIXED
  BEGIN SIMPLEINPUT date
    HEADING "Date of departure"
    TYPE <DATE>
  END SIMPLEINPUT
  BEGIN SIMPLEINPUT fromPlace
    HEADING "From:"
    TYPE <TEXT>
    DEFAULT <TEXT> ""
  END SIMPLEINPUT
  BEGIN SIMPLEINPUT toPlace
    HEADING "To:"
    TYPE <TEXT>
    DEFAULT <TEXT> ""
  END SIMPLEINPUT
  BEGIN SIMPLEINPUT cashAdv
    HEADING "Cash Advance Required"
    TYPE <MONEY>
    DEFAULT <MONEY> $0.00
  END SIMPLEINPUT
  BEGIN COMMAND apply
    HEADING "Apply"
  END COMMAND
  BEGIN COMMAND cancel
    HEADING "Cancel"
  END COMMAND
  BEGIN COMMAND quit
    HEADING "Quit"
  END COMMAND
END FORM

```

for getting responses from users and sending these back as user inputs to applications. In other words, UIL can be considered as a local translation tool of FDL for display devices while FDL is the universal language for information exchange. On the other hand, inFOrms are the scripts written in FDL to express a unit of interaction between two DIANA agents.

Each UIL is device-specific. There is an instance of a UIL on each category of access device. When UIL renders inFOrms, it maps each different type of interaction item to the corresponding user interface feature available on that display device. For instance, UIL may use radio buttons to represent questions that present to users a list of options from which to choose one. The implementation of UIL for each display device is independent of which applications the UIL serves.

Grouping. Besides rendering, UIL is also responsible for managing the interaction process with the user. Applications may send multiple inFOrms together in a single interaction, with dependencies described among the various inFOrms. For example, a travel authorization application may send a top-level menu form with another travel request form to the user in one interaction. When the user chooses the travel request submission function in the top-level menu form, she is directly given the travel request form. Thus, the flow of interaction between the user and the application can be specified within the inFOrms. When UIL interprets inFOrms, it understands the flow of interaction and it guides users through the process.

Extensibility. When applications are being developed in DIANA, they make no distinctions regarding the particular display devices to be used. Therefore, when a new access device is used, all that needs to be done is to implement a UIL for this new device. After that, all applications can work with this device requiring neither modification nor recompilation. This portability is achieved by having the applications and UIL be separate entities that communicate using FDL, which is device independent.

3.3 Implementation

Broadly speaking, the structure of a UIL can be divided into 3 parts: a parser which parses inFOrms into internal structure, a renderer which maps the input items listed in Table 1 into the user interface available on the target display device, and a communication module which talks with User Courier. The parser and the communication module are the same for all instances of UIL.

To implement a UIL for a new display device, the only implementation needed is of a new renderer for the user interface on that device.

4 THE DIANA NETWORK ARCHITECTURE

The network components of DIANA are the Couriers and the Replay Agents. Each Courier is responsible for the actual transmission of information across the network, while each Replay Agent deals with managing the inFORM replies and passing them to the applications when desired. Before delving into the architectural details, we will explain our approach of hiding the network details from the applications.

A significant part of the design and development cycles of many existing distributed applications is spent on dealing with networks and communications. The network logic is deeply embedded in these applications, which incurs significant costs for maintenance, upgradability and porting of these applications to other user platforms.

For example, in a client-server model, the basic client is designed to work with particular network protocols, such as TCP/IP, UDP, and so on. Even though the flexibility of choosing a particular network protocol is critical for some applications, there are many applications that are not time critical and would benefit from independence from the heterogeneity of the networks. These are the applications that fit very well into our model. We must also mention that in the DIANA architecture, the performance penalty incurred by a reliable protocol is mitigated through caching to reduce network traffic. By grouping multiple one inFORMs in a single interaction, we further reduce network utilization.

Another reason behind our decision to de-couple communication logic from applications was the commonality of communication logic among a variety of applications. The DIANA architecture allows application developers to model their applications at a higher level—information exchange—and relieves them from having to deal with the intricacies of network communication.

Our approach also takes into account mobile computing and intermittent connectivity. Many computer users are moving to smaller, portable devices. This migration has led to the evolution of a large variety of portable computing

devices from notebooks to palmtops to PDA's. Some of these devices are connected to the network through wireless communications, such as wireless e-mail. Such connectivity is often asynchronous. Other mobile communication can support a continuous connection but with high latency. In this case, if responses do not arrive in a timely fashion, DIANA will fall back to the intermittently connected case transparent to the application. By hiding how the information was transmitted between the user device and the application, DIANA supports a variety of network protocols between nomadic devices and network resident applications.

4.1 Issues in Network Management

We begin the design discussion with some of the questions that we answered in the process of modeling the network components.

Naming. The first question we considered is how different entities in our system address each other. There are two type of entities that interact with DIANA - end user clients and applications (we will use the term "agent" unless we wish to distinguish between these two). These agents can appear at different places in the network.

Everything directed to a user is directed to the UIL component of DIANA residing on the user's device, and everything directed to an application is directed to the Application Replay Agent at the application's host machine. Also, all connections between two nodes are made between the User Courier and the Application Courier, which in turn forward the inFOrms to the appropriate agents. We use the existing `port@host` naming conventions by assigning well known ports to the DIANA components and using the Internet addressing mechanisms to identify the hosts (when the application and user device are directly connected over the Internet).

The entities interacting with DIANA can address a particular agent using a notation similar to `agent@host`. This information is stored in the meta-information headers of the inFOrms discussed later in this section. The Courier can look at the meta-information and route the forms locally to the appropriate DIANA component.

DIANA resolves the issue of the heterogeneity of application naming in the following manner. In DIANA, the sender application must obtain an inForm script from the inForm Repository before sending it to the destination. Ap-

plications can obtain only those inFOrms, which they are prepared to interpret and for which they have already registered the scripts with the Courier. These applications provide their local identifiers at the time of registration and these identifiers are stored in the meta-information of the inFOrms. Hence the Courier at the sender's machine resolves the host part of the address and the Courier at the receiver's machine resolves the agent part.

Delegation. In a corporate environment, the end users of a system perform a lot of routine work, such as generating weekly progress reports and scheduling meetings. In many cases, it is desirable to write an application that does some regular work on the users' behalf.

This notion requires that a system like DIANA provide mechanisms for sorting inFOrms according to *roles* or functionalities. The end users should be allowed to *delegate* inFOrms to other authorized agents to do some work on their behalf. This delegation is transparent to the agent sending the inForm.

We suggest that the end users provide delegation information to the Courier. The Courier goes through an extra indirection before determining the actual destination of an inForm. As an example, suppose that the corporate VP has an automatic meeting scheduler that sends inFOrms to other meeting attendees requesting that they confirm the schedule. The receivers of these inFOrms can either complete the inFOrms by themselves or delegate this job to their secretaries or their own meeting schedulers.

Ultimately, supporting a system based on $\langle \text{agent}, \text{role} \rangle$ pairs to resolve the delegation of inFOrms seems promising. This convention is very important from the point of view of corporate computing where different people can have different responsibilities under different roles. The details of such a system have been left for future consideration.

4.2 Meta-Information on InFOrms

Each inForm contains a meta-information header, which is designed so that different components of DIANA can monitor the flow of an inForm through the system. The source and destination addresses are included in the meta-information header. It is desirable to keep this header as small as possible. Table 2 presents some of the logical fields that are an essential part of the meta-information. This list is not complete; we expect it to evolve with time.

Name	Function	Agent Access
<i>TransferID</i>	identifier specifying the flow of a particular inForm	read only
<i>Source</i>	address of the sender	read only
<i>Destination</i>	address of the receiver	read/write
<i>Delegated to</i>	address of the actual destination	read only
<i>Time</i>	time fields to monitor flow intervals	read only
<i>Priority</i>	urgent/normal/... to determine how time critical this transfer is	read/write

Table 2 The Meta-Information Headers for FDL

4.3 The User and Application Couriers

In DIANA, the Courier is divided into Application Courier and User Courier, which handle communication issues for application and users respectively. The communication between the Application Courier and the User Courier can happen in two basic modes: connected and disconnected.

The Courier components always attempt to establish a direct connection if possible. The Courier has the capability to run on top of multiple network protocols, such as TCP/IP and UDP, in connected mode. Once the connection is established, the Courier components exchange a series of commands as a result of which the desired action is performed, e.g., the inForm is delivered to the UIL from the application, or an inForm reply is sent from the UIL to the Application Replay agent.

In the disconnected mode, the Courier currently uses e-mail as the medium of communication. The e-mail messages contain special subject headers and can be filtered and passed onto the corresponding Courier component at the destination. Time-critical applications which rely on the network for a guaranteed minimum transfer rate are not compatible with this mode of communication. However, other protocols for disconnected or intermittently connected operation and may be implemented in the DIANA architecture without change to applications or other user device software. The Courier provides a mechanism to determine the communication mode (i.e., connectedness) at any particular time.

4.4 Application Surrogates

It is important to explore how much useful work can be done on the user's access device while the accessor is disconnected from the network, or when the system is operating in the asynchronous mode.

The introduction of an application surrogate to deal with the disconnectivity logic of the user interaction is presented as an attempt to address this problem. A careful study of the behavior of these surrogates will reveal the benefits of this technique. By caching the inFOrms required for an application and having a surrogate for that application, the users will be able to do useful work while disconnected. The Coda File System relies on caching of objects (including the applications which users are expected to use during the disconnected period) to support disconnected operation [16]. We expect that an application surrogate-based approach will be able to use the limited resources on the nomadic devices in a more efficient manner. Also, the use of application surrogates facilitates support for multiple platforms as only the surrogates will have to be implemented on each user platform, whereas the bulk of the application logic will be network resident and most of the user interface will be handled by the UIL. A scripting language, such as TCL, appears useful for constructing Application Surrogates.

Ideally, we would like to be able to generalize the functionality of the application surrogates and include them in the UIL. Whether the applications should be aware of the presence of application surrogates on the remote devices also remains to be answered.

4.5 Implementation

The Application and the User Courier consist of 2 parts: a communication manager which handles both synchronous and asynchronous communication modes, and a connection table which records the information on the pairs of communicating parties. At this stage, the Replay Agent simply maintains a first-in-first-out queue of reply forms. In subsequent work, we will explore the operation of the Replay Agent and its use in disconnected operation.

5 APPLICATION DEVELOPMENT METHODOLOGY

Application development using DIANA framework differs from conventional application development approach. In conventional application development, the developer first designs the functionality and infrastructure of the applications. Then she works out the finer details as well as the interface to the external world. But in the DIANA approach, when the developer designs an application, she first designs the application's interaction with users in terms of inFOrms. These inFOrms define inputs requested from users as well as the application's responses to those users.

After defining inFOrms, the developer can then design the structure of the application and code it. When coding the application, the developer should assume only an asynchronous communication channel with the users, even though the Courier might establish a synchronous channel with the user.

This approach of first defining the interface with the users before coding allows the developer to mock-up the application and test the correctness of the interface independent of the functionality of the application. Testing can be carried out at an earlier stage so that some design flaws can be detected earlier.

5.1 Benefits of DIANA to Application Developers

The goal of DIANA is to achieve display independence and communication independence in software development. These are achieved by the separation of display logic and communication logic from applications. When communicating with human users or other applications using DIANA, applications do not see user interfaces and communication channels. Instead, they merely send to the Courier inFOrms, which contain the information being exchanged, and the Courier will take care of the delivery of the inFOrms. In summary, DIANA provides an abstract message exchange mechanism (the Courier) with a universal language for describing the messages (FDL).

There are significant advantages to using DIANA in application development. First, by separating display logic and communication logic from applications, developers can be freed from handling display and communication issues and

can concentrate their efforts instead on the design and coding of the application logic. Thus, application development is speeded up and the quality is enhanced.

Second, with the provision of these display and communication facilities, even novice programmers can develop sophisticated applications with GUI and communication. Without DIANA, the development of these applications will require knowledge of multiple user interfaces and communication mechanisms to have the same flexibility.

Third, DIANA provides application portability across heterogeneous display devices. While a number of portable GUI facilities currently available also provide portability, they limit the usability of applications to bitmap-based and character-based display devices. These portable GUI facilities focus on the presentation format of information exchanged between human users and applications, and enforce strict representation of such information using graphical objects like buttons and scroll bars. On the other hand, DIANA focuses on the semantics of information exchanged but does not presume its representation. Therefore, it is able to support a greater variety of devices with a varied level of display capabilities.

Note that it is easy for DIANA to support future display devices that are presently unforeseen, as DIANA focuses on the semantics of information being exchanged. Once a new device is supported, all applications using DIANA can work with that new device, once the common UIL, User Courier, etc., are developed. Application Surrogates are not required for initial use of the new device.

As FDL focuses on the semantics of information being exchanged, it serves as a universal language for applications. Applications can use FDL to talk with one another. The consequence is that FDL provide a common ground for applications to communicate and collaborate. FDL can be used as a communication language for distributed applications. Applications share a common protocol, namely inFOrms, and thus can achieve a greater degree of cooperation not achievable without a common protocol.

One of DIANA's main goals is to facilitate mobile computing, where users have only intermittent communication channels with network-resident applications. The asynchronous communication mechanism provided by DIANA favors such intermittent connectivity. Consider the situation when a mobile user has only 5 minutes to connect to a host application but the whole interaction will take 10 minutes. Without DIANA, the user will be unable to continue work after the first 5 minutes. However, with DIANA, the user can use the first 5 minutes

to send an inForm to request some job to be done by the host application. He can then check the results of his work in another inForm at a later time.

DIANA supports disconnected operation. When users are disconnected from host applications, they can still work with the inForms stored on their client devices. There are agents which will record their operations and replay the whole operations to the applications automatically once the users are connected again.

Finally, DIANA minimizes network traffic by using inForms to exchange information. The inForms do not contain extra data on how to represent the information. Without DIANA, applications will have to send extra detail to a mobile display device, for example, to render the windows and buttons. The saving in communication traffic is especially important to mobile computing as the communication channels have limited bandwidth.

5.2 DIANA's Application Programming Interface

From an application's perspective, the whole processing of communication with the users can be divided into a series of steps. First, the application prepares the inForm it wants to send to the user. At this point, the application can update the inForm by adding a dynamic data part to the inForm. Then it sends the inForm to the user. After that, the application can continue with other operations and asynchronously wait for the reply form from the user. Alternatively, the application can choose to wait synchronously. After receiving the reply form, it can access the user's replies from the reply form.

Table 3 describes the API provided by DIANA to do the above actions.

Readers should notice that the list of API calls exported by DIANA is simple and small. This simplicity compares favorably with the complexity of the API required to for applications to handle GUI and communication directly. This simplicity reflects the design goal of DIANA: to provide application developers with a simple and uniform interface to a great variety of access devices.

Call	Description
<code>initDiana()</code>	initializes DIANA and should be called before any DIANA facilities are used.
<code>prepareForm()</code>	reads in an inForm given its name. The <code>prepareForm()</code> procedure should be called first before any dynamic data can be apply to the inForm and the inForm can be sent.
<code>getItem()</code>	gets a pointer to an item within another item, given a string specifying the item from the other item. This procedure is useful for navigating through an inForm.
<code>setAttr()</code>	sets the attribute of an item. Various attributes of an item can be set, for example, the name, the heading, the options, the default value. The <code>setAttr()</code> procedure is particularly useful for setting the dynamic data part of an inForm before the inForm is sent.
<code>sendForm()</code>	sends an inForm to an agent, either a human user or an application.
<code>getReplyForm()</code>	waits for a reply from the agent. Applications can specify whether the waiting should be blocking or non-blocking. A handle to the reply form is returned, which the application uses to access the user's reply.
<code>getAttr()</code>	gets a particular field inside the reply inForm given the field name. The return type is a pointer to void, which the application should cast to the proper pointer type before accessing the value.

Table 3 The description of the DIANA API

5.3 Procedural vs. Object-Oriented Interface

Currently, DIANA uses a conventional procedural programming approach in both the view exported to application developers and its implementation. Alternatively, an object-oriented approach can better model the whole information exchange mechanism tackled by DIANA. It is possible to develop an object hierarchy to represent the interaction items and inForms; however, recompilation of application programs might be required if the object hierarchy is changed to encompass new types of data potentially being transmitted. The overloading feature of some object-oriented programming languages like C++ avoid the typing problem in the return value of the `getAttr()` procedure mentioned above.

Based on the type of interaction items passed as the parameter, the right type of return value can be returned.

6 CURRENT IMPLEMENTATION

At the current stage, we have implemented only a portion of the whole DIANA framework. We have implemented a UIL for OpenLook environment. We have also implemented another UIL for the telephone, which is simulated by a software whose interface consists of 12 buttons mimicking the dial buttons of a telephone for input and dialogs are spoken out by a speech synthesizer for output. The Courier currently supports only TCP/IP synchronous communication. The Replay Agent is a simple first-in-first-out queue of reply forms.

To test and verify DIANA, we have also implemented a travel authorization application which allows users to submit travel requests, review request status and approve requests. This application has successfully worked with both OpenLook and telephone interface without special adaptation to either display device. The OpenLook interface is a good representative interface in the workplace while the telephone interface is common in a mobile environment. Though tested in a simulated environment, we believe that the idea of display independence for mobile computing has been successfully demonstrated in this exercise.

We have implemented and tested disconnected operation, but the results of this experiment are beyond the scope of this paper.

7 FUTURE ISSUES

This section discusses some of the outstanding issues we have not yet addressed.

7.1 Workflow Support

The DIANA infrastructure has important features to support business workflow processing. We feel that one dimension of the evolution of DIANA can lead us into studying workflow systems and incorporating basic features of workflow systems into DIANA as default services provided by the Courier. e.g. the

Courier already has the notion of delegation based on different roles of end users (see section 4.1).

The device and network independence supported by DIANA is essential for workflow systems where a variety of heterogeneous systems need to interact with each other.

The universal FDL on the other hand can make all the applications understand each other. Legacy applications can be incorporated into a workflow system by providing a single DIANA client which can translate between the FDL and the applications input and output parameters.

However, minimal work has been done in this direction so far and further study of workflow systems and their overlap with DIANA is essential.

7.2 Disconnectivity Logic

It is important to explore how much useful work can be done on the user's access device while the accessor is disconnected from the network, or when the system is operating in the asynchronous mode.

The introduction of an application surrogate to deal with the disconnectivity logic of the user interaction is presented as an attempt to address this problem. A careful study of the behavior of these surrogates will reveal the benefits of this technique. By caching the inFOrms required for an application and having a surrogate for that application, the users will be able to do useful work while disconnected. The Coda File System relies on caching of objects (including the applications which users are expected to use during the disconnected period) to support disconnected operation [16]. We expect that an application surrogate-based approach will be able to use the limited resources on the nomadic devices in a more efficient manner. Also, this approach seems more suitable to support multiple platforms as only the surrogates will have to be implemented across platforms, whereas the bulk of the application logic will be network resident and most of the user interaction will be handled by the UIL.

Ideally, we would like to be able to generalize the functionality of the application surrogates and include them in the UIL. Whether the applications should be aware of the presence of application surrogates on the remote devices also remains to be answered.

7.3 Replaying Disconnectivity Interaction

A simple scheme can be adapted to record interaction occurring between the application surrogate and the end user while the network is down. This logged interaction can be “replayed” to the host application after the network connection is re-established or when this interaction is forwarded through e-mail.

The user may complete several inForms on a disconnected device before they can be transmitted to the application. In this case, the inForms need to be given to the application in the order that the application expects them. It is possible that an exception arises, so that the application next requests information not contained in the inForms already completed. In particular, the Application Surrogate has made a different decision from the actual Application due to a lack of information. The User Replay and Application Replay agents will allow the additional information to be supplied and the communication to be resynchronized. If necessary, the inForms already completed may be edited. We call this synchronization process *zippering*. We have implemented zippering, but the results of this experiment are beyond the scope of this paper. The goal of zippering is for the application not to be able to detect such inconsistencies, particularly when it does not even know about the existence of application surrogates.

7.4 Customizing Interfaces

An interesting problem is to let the applications customize their interface, but without re-introducing the device-dependence problem.

According to application developers, 70% to 95% of their efforts are concentrated on developing and customizing the look and feel of their applications. This statistic implies that DIANA must address the interface layout and customization issue. One approach can be that the UIL provide a big percentage of the above functionality and let the developers write their own surrogates to customize the remaining part as desirable. In particular, the UIL could utilize user and application profiles to support user-, device-, and application-specific customizations.

Extending FDL to include some layout hints about the grouping of several items on an inForm is also possible.

8 CONCLUDING REMARKS

DIANA addresses the issues of platform dependence, network binding, and interoperability by proposing a simple, semantic-based system. DIANA will eliminate the effort required to implement application-specific user interfaces and communication modules, thus significantly reducing the design and implementation cycle of applications. Applications can be implemented and tested on a simple text interface and then plugged into the DIANA interface. Using a complete and reliable implementation of DIANA, the application developers will be able to concentrate on testing their applications' processing logic without having to worry about the intricacies in user interface code or communications code. Also, the design of the user interface can proceed in parallel with core application development and can easily be changed to adapt to any new requirements placed on the application. We believe that the ideas introduced in this paper will address some of the drawbacks in current workflow systems, as well as enhance interoperability between incompatible systems and increase the customer base of applications.

A current prototype of the system exists which supports the inFOrm life cycle. The network component supports direct connectivity using a TCP/IP connection and a UIL for the OpenLook environment exists. We have developed a telephone interface to FDL. We have also experimented with disconnected operation and will describe that experiment in future writing. We expect that many of these ideas will be very important in the world of mobile computing and heterogeneous systems in the near future.

Acknowledgements

This research was funded in part by Sun Microsystems. We acknowledge the support and encouragement of Terry Keeley, Bert Sutherland, and Emil Sarpa.

We thank Bob Sproull, and Xinhua Zhao for their thoughtful comments. We thank Marianne Siroker for her help in preparing this paper.

REFERENCES

- [1] Bass, L., Clapper, B., Hardy, E., Kazman, R. and Seacord, R., "Serpent:

A User Interface Management System," Proceedings of the Winter 1990 USENIX Conference, Berkeley, CA, January 1990, pp. 245–248.

- [2] Bennet, A., "FormIKA: A Form-Based User Interface Management System for Knowledge Acquisition," Knowledge Systems Laboratory, Stanford University, June 1990.
- [3] Caruso, D., "General Magic Got Quite a Start," Digital Media, March 29, 1993.
- [4] Coutaz, J., "PAC, An Implementation Model for Dialog Design," Proceedings of Interact T87, Stuttgart, September, 1987, pp. 431–436.
- [5] de Souza C. S., "A Semiotic Approach to User Interface Language Design," Center for the Study of Language and Information, Stanford University, May 1992.
- [6] Gosling, J., and McGilton, H., "The Java Language Environment, A White Paper," Sun Microsystems Computer Company, May 1995.
- [7] Guarna, V. A., Jr. and Gaur Y., "A Portable User Interface for a Scientific Programming Environment," Center for Supercomputing Research and Development, University of Illinois, Urbana-Champaign, February 1988.
- [8] Imielinski, T. and Badrinath, B.R., "Mobile Wireless Computing: Solutions and Challenges in Data Management," Department of Computer Science, Rutgers University, 1993.
- [9] Kazman, R., Bass, L., Abowd, G. and Webb, M., "Analyzing the Properties of User Interface Software," Department of Computer Science, Carnegie-Mellon University, October 1993.
- [10] McBryan, O. A., "Software Issues at the User Interface," Department of Computer Science, University of Colorado, May 1991.
- [11] Myers, B. A. and Rosson, M. B., "Survey on User Interface Programming," Department of Computer Science, Carnegie-Mellon University, February 1992.
- [12] Myers, B. A., "Why are Human-Computer Interfaces Difficult to Design and Implement?" Department of Computer Science, Carnegie-Mellon University, July 1993.
- [13] Pfaff, G., (ed.), User Interface Management Systems, Newyork: Springer-Verlag, 1985.

- [14] Satyanarayanan, M., Kistler, J. J., Kumar, P., Okasaki, M. E., Siegel, E. H. and Steere, D. C., "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE Transactions on Computers*, April 1990.
- [15] Satyanarayanan, M., "Scalable, Secure, and Highly Available Distributed File Access," *IEEE Computer*, May 1990.
- [16] Satyanarayanan, M., Kistler, J. J., Mummert, L. B., Ebling, M. R., Kumar, P. and Lu, Q., "Experience with disconnected Operation in a Mobile Computing Environment," Department of Computer Science, Carnegie-Mellon University, June 1993.
- [17] Steere, D.C., Kistler, J. J. and Satyanarayanan, M., "Efficient User-Level Cache File Management on the Sun Vnode Interface," In Summer Usenix Conference Proceedings, Anaheim, June 1990.
- [18] Tso, M., "Using Property Specifications to Achieve Graceful Disconnected Operation in an Intermittent Mobile Computing Environment," Xerox Corporation, Palo Alto Research Center, June 1993.
- [19] Watanabe, T. and Oketani, I., "Functional Design of Cooperatively Integrated Information System," Data Processing Center, Kyoto University, October 1986.
- [20] Zarmer, C. and Canning, P., "Using C++ to Implement an Advanced User-Interface Architecture," HP Laboratories, Hewlett Packard Company, April 1990.

THE CMU MOBILE COMPUTERS AND THEIR APPLICATION FOR MAINTENANCE

Asim Smailagic and Daniel P. Siewiorek

*Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213*

ABSTRACT

This paper describes the use of CMU's VuMan wearable computers for maintenance of military vehicles and their communication mechanisms with the outside world. The key concepts involve: portable, hands-off access to information; mobility of users and uploading information yielding increased productivity of maintenance operation. VuMan is used as a Referential System, replacing large volumes of printed materials such as maintenance manuals. The main tasks are Limited Technical Inspections (LTI) and Trouble Shooting Flow Charts for an Amphibious Motor Vehicle. This effort included the development of the VuMan Hypertext Language (VHTL), using a forms-based hypertext paradigm that provides quick access to manuals. The VHTL considerably simplifies the task of creating document systems that integrate forms, references (hyperlinks), images and complex control structures (such as nested menus). The User Interface has been designed to provide access to the information in an intuitive and natural manner. Both the User Interface and the document structure are centered around the concept of fields. When selected by the user, these fields invoke some action, such as bringing up a menu, following a reference or toggling a check mark. The User Interface can accommodate two types of input devices: a three-button mouse and a multiposition rotary dial. Our estimate is that the time required to perform some typical Marine Maintenance procedures using VuMan will be cut in half.

1 INTRODUCTION

Mobile computers deal in information rather than programs, becoming tools in the user's environment much like pencils or reference books. The mobile

computer provides automatic, portable access to information. Sensors make the mobile computer an active part of the environment. Information can be automatically accumulated by the system as the user interacts with and modifies the environment, thereby eliminating the costly and error-prone process of information acquisition. Several new application areas have been made possible by this paradigm shift:

- Maintenance
- See-through reality for manufacturing
- Self-guiding navigation
- Medical

In this paper we focus on the application of the VuMan wearable computer for maintenance. As systems become more reliable and more complex, maintenance and repair become increasingly challenging problems. In maintenance, the individual field service engineer will not encounter enough failures of the same type from which to perceive a pattern and thereby develop a shortcut to diagnose the problem. Rather, each problem has to be time-consuming diagnosed from first principles. A portable information system can use highlighting and animation to identify components and procedures necessary to replace them.

The mobile user requires substantial processing power, but packaged so that its weight, size, and thermal properties are almost unobtrusive. The information that the system provides must be easily and comfortably accessible. Communication between the mobile computer and distant computers must be continuously available.

Carnegie Mellon University has built four generations of mobile computers, Table 1, and work is close to completion on a fifth generation. One class of applications is referencing large amounts of information while performing an activity and generating summary status information. The status information is transmitted to other computers for further processing. This paper describes the use of mobile computers in the application of vehicle maintenance. First, the next section gives an overview of the capabilities of the mobile computers.

2 CMU MOBILE COMPUTERS AND THEIR APPLICATIONS

VuMan 1 [1], conceived in 1991, allows a user to maneuver through the blueprints of a house using three buttons for input, much like the mouse of a desktop computer. Output is provided on a commercially available head-mounted display, the Private Eye [4], which gives the illusion of viewing a personal computer screen from about five feet. The VuMan 1's electronics include an 8 MHz 80188 processor, 0.5 MB of ROM, and a Private Eye controller board. Table 1 shows the characteristics and attributes of the CMU mobile computers.

The original application of VuMan 2 [2], built in 1992, was to allow a user to navigate the Carnegie Mellon campus. It has a database of buildings, departments, and people, so that a user unfamiliar with the campus can find the location of an appointment, get information on a faculty member such as a phone number or office number, or locate a specific building on the campus. Like VuMan 1, VuMan 2 uses the Private Eye for output and three buttons for input. Unlike VuMan 1, however, VuMan 2 is not dedicated to a single application. New applications are loaded via a Flash memory card. A second application developed for VuMan 2 is an electronic maintenance manual for an alternator. A user can scan through manual pages and then access the corresponding diagram. VuMan 2 has also been used as a testbed for developing the third application, maintenance of amphibious vehicles, which is ported to the VuMan Maintenance Assistant (VuMan MA). Table 1 illustrates that VuMan 2, which has over twice the functionality of VuMan 1, is at least a factor of four better in terms of volume, weight, and power consumption. The savings were a result of replacing the Private Eye controller board with a single programmable logic device, replacing the glue logic with a FPGA, and replacing EPROMs with a Flash memory card.

The initial application of the third generation mobile computer, Navigator 1 [3], built in 1993, is as a campus navigation tool, similar to VuMan 2. Unlike VuMan 2, Navigator can use speech as input, allowing completely hands-free operation. The Navigator's speech recognition system is speaker-independent [5], has a 200 word vocabulary, and currently runs at about eight times real time. A mouse is also available, in case the speech recognition rate is low or speaking is undesirable. A second major difference between Navigator and VuMan is that Navigator is a general purpose computer while the VuMan computers are embedded. Navigator runs the Mach operating system [6], allowing applications to be developed on a Unix workstation and then transferred to the Navigator platform. Software developers can use the standard Unix environ-

ment, such as X Windows [7] and Shell scripts, in their applications. A third difference is that the Navigator architecture is modular, so that the hardware can be re-configured based upon the application.

For example, in the campus tour application, a Global Positioning System (GPS) unit is used to locate the user with an accuracy of several meters over an area of a few square kilometers. In some manufacturing applications, however, a different positioning system could be used to locate the user with an accuracy of a few millimeters over an area of a few square meters. Other parts of Navigator that can be re-configured depending upon the application are the telecommunications and the display.

VuMan MA incorporates a new housing design to withstand shock, temperature, water, and dirt. VuMan MA has been field tested in the Marines vehicles maintenance environment. It uses an input interface combined of a rotary dial and a single push-button. The speed and ease for a user to scroll through many options that may appear on a screen are the reasons for the use of a rotary dial. A link is provided between VuMan MA and a Logistical Maintenance Computer (LMC) so that results from vehicle inspection check lists can be uploaded for scheduling and planning. VuMan MA includes hardware power management, and two PCMCIA slots. In addition to a Flash memory card, another PCMCIA device can be supported in a modular fashion, such as a radio. The introduction of programmable micro-controllers for input/output (to allow reconfiguration of the electronics with mouse, dial, etc.) and power control (to selectively turn off unused chips) provided a higher degree of flexibility than our previous designs.

3 VUMAN AS A MAINTENANCE ASSISTANT

In [8] we have identified several classes of applications for mobile computers, requiring different capabilities: Referential Systems, Augmented Reality, and Information Sharing. The first class, Referential Systems, requires access to a large, relatively static data base for reference in completing a complex task. Typical referential applications include maintenance and operation procedures. During the past ten months, site visits were conducted to an oil refinery, an aircraft maintenance facility in Pittsburgh, and two US Marines vehicle maintenance operations. A striking similarity was observed between the referential requirements for maintenance. Based on walk-throughs of a variety of mainte-

Artifact	VuMan 1	VuMan 2	Navigator	VuMan MA
Delivery Date	Aug 91	Dec 92	June 93	Dec 94
Number of Units	30	7	3	20
Embedded/GP	embedded	embedded	general purpose	embedded
Design Style	semi-custom	fully-custom	composition	fully-custom
Custom Boards	1	1	3	2
Off-the-Shelf Boards	1	0	5	0
Chip Count	24	5	14	10
Lines of code	1800	4700	38000	12000
Processor	80188 - 8MHz	80C188 - 13MHz	80386 - 25MHz	80386 - 20MHz
RAM	8KB	512KB	16MB	2MB
Nonvolatile Storage	512KB	1MB	85MB	40MB
Input	3-button	3-button	speech/mouse	dial
Display Resolution	720 x 280	720 x 280	720 x 280	720 x 280
Dimensions (inches)	10.5 x 5.25 x 3	7.75 x 4.5 x 1.37	7.25 x 10 x 3	5 x 6.25 x 2
Power (W)	3.8	1.1	7.5	2
Weight (lbs.)	3.3	0.5	9	1.75
Design Activity				
Magnitude of Design	innovative	innovative	innovative	innovative
No. Designers	4	6	21	16
No. CAD Tools	16	16	7	7
Person-Mo. Effort	12	6	28	42
Quantity Fabricated	30	6	3	20
Part Count	45	12	8 boards	2 boards
Housing Fabrication	vacuum-forming	SLA/molding	pressure forming	molding/machining

Table 1 Attributes of CMU Wearable Computers

nance activities, the following model was generated. Upon receiving a trouble report or during a periodic maintenance check/upgrade, an initial inspection checklist is used by experienced maintenance personnel to identify problems. Information from the checklist is used to schedule personnel and issue task orders. Task orders are either accompanied by detailed printed instructions or require personnel to look up procedures in a set of maintenance manuals. For example, a typical manual for a DC-9 class aircraft contains over 110,000 pages, half of which are obsoleted every six months. These manuals frequently cannot leave the manual library, so maintenance personnel must often return to the

library, resulting in much lost time. Often only a small number of manual pages are required to effect a repair, but these pages may be spread out over several volumes with several hundred pages per volume. In addition, maintenance activities are often performed in confined places. The Marines may assign two people to a maintenance activity in an Amphibious Tractor, one person to read the procedures from the manual and one to perform the maintenance activity. A referential mobile computer which provides access to maintenance information while leaving hands free to perform the required physical operations could be an effective way to improve productivity. The steps to be performed in a general maintenance activity, for which VuMan MA is developed, are depicted as follows:

- Vehicle brought in for routine or emergency maintenance
- Limited Technical Inspection (LTI)
 - Visual inspection
 - On-board test systems
 - Service checklists
- Equipment Repair Order (ERO)
 - Specifies maintenance actions to be performed
- Trouble Shooting
 - Logic charts
- Repair/Replace Parts (R&R)
 - Order replacement parts
- Testing
 - Verify repair properly performed

An example of a maintenance document is the Limited Technical Inspection (LTI). An LTI is a 50-page document including:

- Processing a check-list of over 500 items, one for each part of the vehicle (e.g., left track, suspension, turret)

- Selecting one of four possible options about the status of the item: Serviceable, Unserviceable, Missing or On Equipment Repair Order (ERO)
- Entering further explanatory comments about the item as needed, for example the part is unserviceable due to four missing bolts.

A sample of the LTI application, using screen dumps, is illustrated in Figures 5 to 10. The startup screen, Figure 5, is immediately followed by the Main Menu, Figure 7, which lists all documents available in the system, as well as data transfer and system control options. All the options are reference control points. Selecting any one of the three documents brings up a table of contents (TOC) of the document. The TOC contains references to all the sections of the document. The standard screen consists of a command area and data area, as in Figure 6. The command area contains control points that allow the user to scroll between screens, follow references, return from references, etc. The data area of the screen displays the text or graphics. The following are some illustrative examples of LTI Application functionality:

- Online assistance and tutorials, Figure 6, allow for rapid user training times.
- Selecting LTI: AAPV781 option from the Main Menu, Figure 7, brings up the screen containing the table of contents for the Personnel Transport LTI document.
- User customization options, such as those depicted in Figure 8, allow Vu-Man to accomodate a wide variety of preferences (including left/right eye dominance). Also configurable is an indicator of available battery lifetime.
- The status for a given item can be entered by selecting the Status control point beneath that item (Figure 9). The user then categorizes the status of the item, entering comments as needed. The Status control point will be highlighted to indicate that the item is completed.
- Figure 10 illustrates LTI items which have been inspected and whose status has been noted.

Our experience with the Marines vehicle maintenance application indicates the importance of user involvement in the design process. Based on that feedback from our users, we have been making further improvements in the user interface.

4 APPLICATION SOFTWARE

4.1 A Forms-Oriented Hypertext Language

The application software for maintenance of military vehicles is based on a hypertext document system for the VuMan wearable computers. The technical inspection of a vehicle should determine its status accurately and rapidly. The inspection involves tedious cross-referencing of information and schematics from several manuals. A hypertext system providing quick access to the manuals is very useful in these situations. The user performing an inspection is required to enter data about different parts of a vehicle by filling out a form. This form needs to be integrated into the basic hypertext system which contains the manual information. The data entered by the user can be treated as independent from the form document itself. Some hypertext languages allow such a mix of forms and hypertext links. In the HTML language of NCSA Mosaic, for example, a form can be embedded into a regular hypertext document. The form filled out by the user can be submitted to a remote computer for an evaluation and response. This type of client-server approach to form processing leads to a high computational and network load overhead when dealing with entire documents that are forms. The VuMan Hypertext Language (VHTL) considerably eases the task of creating document systems which integrate forms, references, images, and complex control structures (such as nested menus that are used repeatedly). These control structures allow the document to change in response to the user inputs. For example, a reference could be followed only if the user had previously selected some menu option. VHTL has been used to implement an extensive LTI.

The LTI checklist consists of a number of sections, with about one hundred items in each section. The users have to manually go through each of these items by using the dial to select “next item,” or “next field.” The Smart Cursor feature represents built-in intelligence in the user interface, and was designed to help automate some of this navigation. This approach is accomplished with the use of two features:

- An input pattern recognizer, which keeps track of what fields the user selects on a given screen, which we can call a “working set.” If the working set remains the same over two or three screens, the navigation system starts moving the cursor automatically to the fields in the working set. In essence, this is a macro recorder that runs continuously during the user’s work session, and uses a heuristic about when to repeat recorded keystrokes.

- A domain-specific heuristic, developed through studies of how users usually navigate through LTI hypertext documents (e.g., their behavior in the presence of multiple options). A high-level navigation pattern was found, which the input pattern recognizer could not identify. The knowledge about this high-level pattern was encoded in a navigation template. The system then uses a heuristic to decide when to apply this navigation template.

4.2 Communication with a Logistical Maintenance Computer

The VuMan wearable computer communicates with a Logistical Maintenance Computer (LMC) by uploading data via an on-board serial port. As an LTI or checklist is being processed, an user can set toggles, select menu items, and make comments. After the user is done, all the entered data need to be uploaded to the Maintenance computer. The option ‘Transfer Data’ on the main menu of the LTI application, Figure 7, allows the user to perform actual data upload. By selecting this option the instructions are given for setting up the communication program on the LMC which is to receive the status information. Our current effort is to employ wireless communication between Vuman and the LMC. Therefore, one of the two PCMCIA card slots on VuMan 3 provides a radio link. The wireless LAN capability is based on Proxim’s RangeLAN2/PCMCIA adapters, and Access Point. Compared with the other wireless LAN systems that we considered, the RangeLAN2 has the lowest power consumption in transmit and sleep modes, and the second lowest in receive mode (after Xircom’s Credit Card Netwave Adapter). Also, it has doze and sleep modes that significantly reduce power consumption when the device is idle, fairly good bandwidth (1.6 Mbps), the longest range available, and solid roaming capability.

5 CONCLUSIONS

The VuMan wearable computers provide portable, hands-off access to text, images, maps, and other information needed to perform maintenance operation. One of the user interface main design goals was to provide a minimal training time, less than five minutes. As a maintenance assistant and advisor VuMan can provide:

- Help in determining vehicle status accurately and rapidly
- Easily accessible and extensive expert information for on-the-job support
- Job aid to ensure accuracy and diminish need for extensive training
- Status information to a fast automated system for problem sorting and generating work orders

The VuMan wearable computer communicates with a Maintenance computer by uploading data via an on-board serial port, yielding increased productivity in maintenance operations. The wireless LAN capability based on spread spectrum PCMCIA adapters is currently being added.

6 FIGURES



Figure 1 VuMan 1 used with construction blueprints

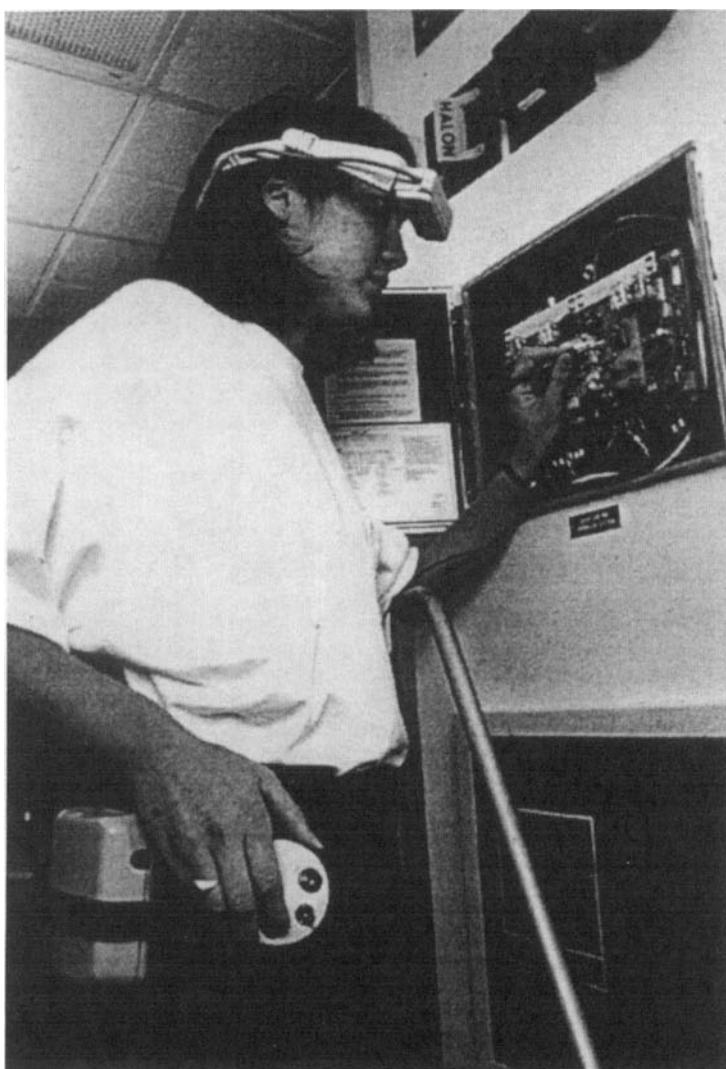


Figure 2 VuMan 2 configured for an electronic maintenance application

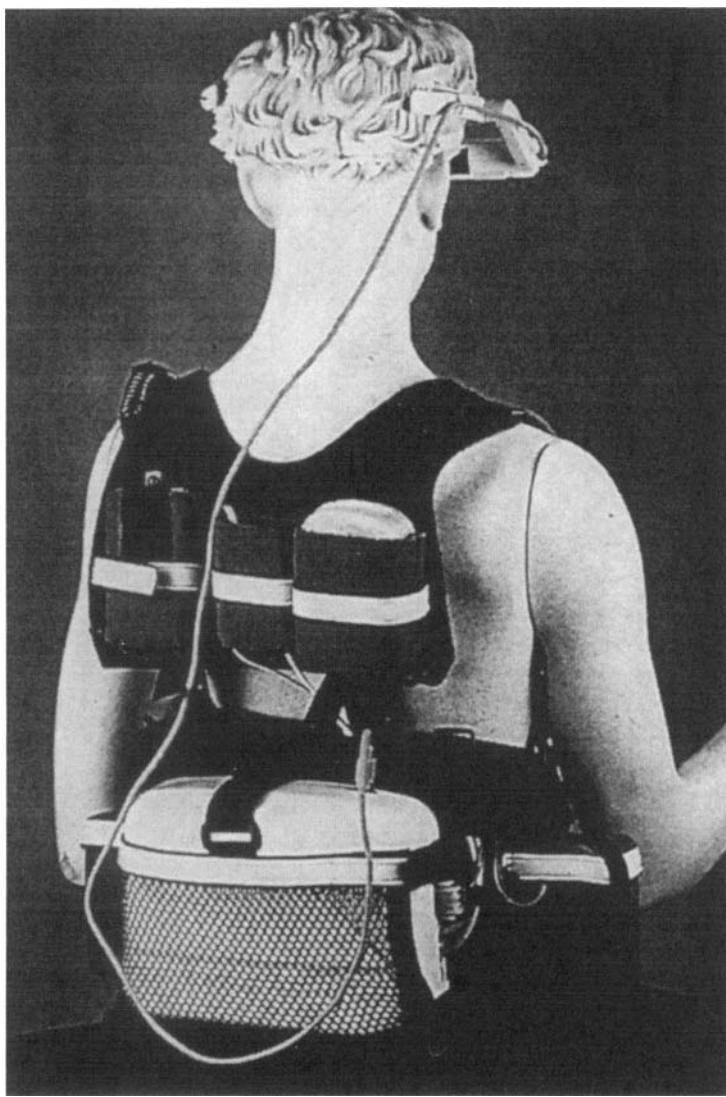


Figure 3 Navigator, a modular wearable computer with speech recognition, GPS and cellular phone.



Figure 4 VuMan MA used for an aircraft inspection

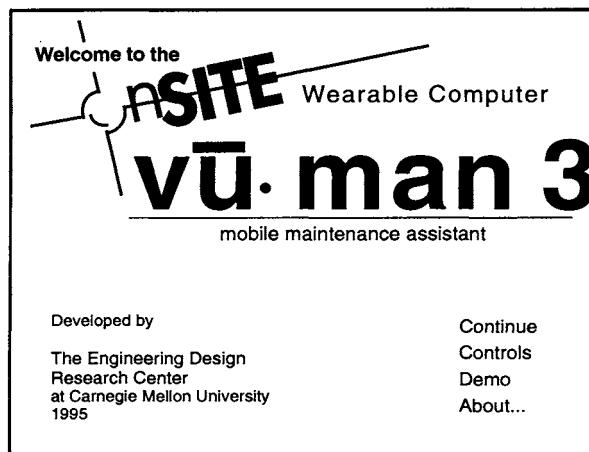


Figure 5 The VuMan 3 (MA) Startup Screen

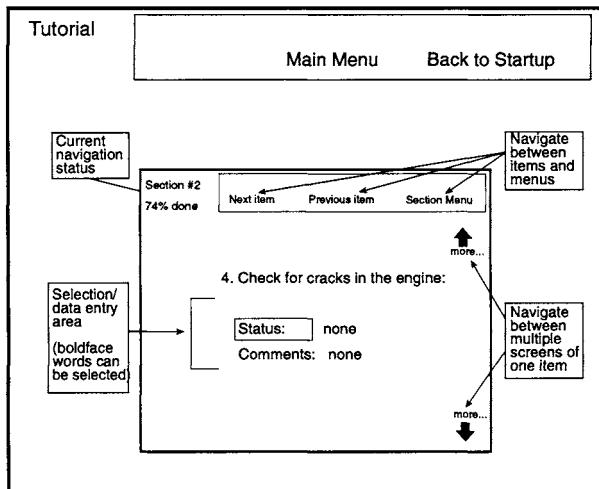


Figure 6 A page from the online user interface tutorial

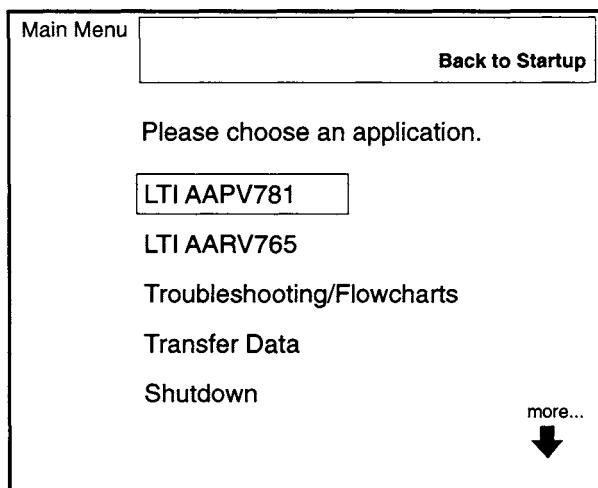


Figure 7 The LTI Main Menu

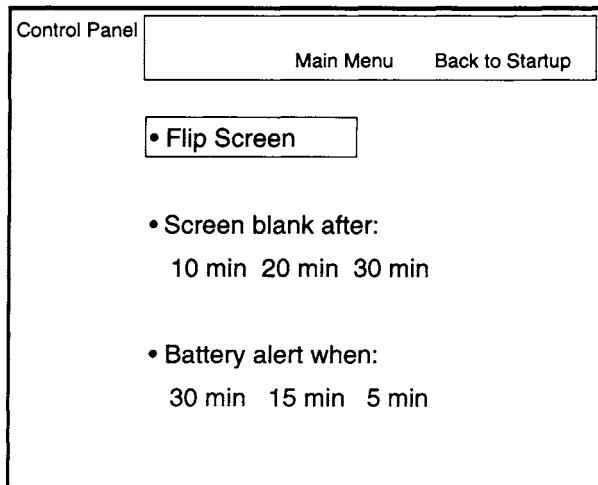


Figure 8 Customization options available through the Control Panel

LTI - sec. 1 15% done	next	previous	section menu
<p># 7. Bow Pod Port Side. Remove drain plug check for water.</p> <p>Status: none Comment: none</p> <p>7b. Port Final Drive</p> <p>Status: none Comment: none</p> <p>7c. Outer Housing</p> <p>Status: none Comment: none</p> <p style="text-align: right;">more... </p>			

Figure 9 Inspection information for the Personnel Transport vehicle LTI

LTI - sec. 2 75% done	next	previous	section menu
<p># 43 b. Check crank shaft</p> <p>Status: Serviceable Comment: none</p> <p># 43 c. Check engine</p> <p>Status: Serviceable Comment: none</p>			

Figure 10 Another item from the Personnel Transport vehicle LTI

Acknowledgements

For their valuable contributions to the wearable computers project we thank: Drew Anderson, John Stivoric, Chris Kasabach, Gautam Valabha, Alex Amezquita, Michael Malone, Nathaniel Barlow, Arijit Biswas, John Dorsey, DeWitt Latimer, Tara Taylor, Eric Zesztarski, and Elizabeth Zimmerman.

REFERENCES

- [1] Akella, J., Dutoit, A. and Siewiorek, D. P., "Concurrent Engineering: A Prototyping Case Study," Proceedings of the 3rd IEEE International Workshop on Rapid System Prototyping Research Triangle Park, N. Carolina, June 1992.
- [2] Smailagic, A., Siewiorek, D. P., "A Case Study in Embedded Systems Design: The VuMan 2 Wearable Computer," IEEE Design and Test of Computers, Vol. 10, No. 3, pp. 56-67, September 1993.
- [3] Siewiorek, D.P, Smailagic, A., Lee, J.C. and Tabatabai, A.R.A., "An Interdisciplinary Concurrent Design Methodology as Applied to the Navigator Wearable Computer System," Journal of Computer and Software Engineering, Vol. 3, No. 2, 1994.
- [4] Becker, A., "High Resolution Virtual Displays," Proc. SPIE, Vol. 1664, Society of Photooptical Instrumentation Engineers, Bellingham, Wash., 1992.
- [5] Li, K.F., Hon, H.W., Hwang, M.J., Reddy, R. "The Sphinx Speech Recognition System," Proceeding of the IEEE ICASSP, Glasgow, UK, May 1989.
- [6] Rashid, R. et al. "Mach: A System Software Kernel," COMPCON Spring '89, San Francisco, CA, March 1989.
- [7] Kantarjiev, C.V. et al., "Experience with X in a Wireless Environment," Proc. of the Usenix Symposium on Mobile and Location-Independent Computing, Cambridge, MA, August 1993.
- [8] Smailagic, A., Siewiorek, D.P., "The CMU Mobile Computers: A New Generation of Computer Systems", Proceedings of COMPCON '94, IEEE Computer Society Press, Los Alamitos, CA, February 1994.

GENESIS AND ADVANCED TRAVELER INFORMATION SYSTEMS

Shashi Shekhar and Duen-Ren Liu*

*Department of Computer Science
University of Minnesota, Twin Cities, Minnesota*

** Institute of Information Management
National Chiao Tung University, Hsinchu, Taiwan
Republic of China*

ABSTRACT

Genesis and ATIS are being developed under the umbrella of the Intelligent Vehicle Highway Systems¹ to facilitate various kinds of travel, including daily commuting to work via private/public transportation and visiting new places and points of attraction. Travelers are naturally mobile, and the most effective way to aid travelers is via mobile computing, which is being explored in the Genesis project at Minnesota. The travelers can use personal communication devices including pagers and portable computers (e.g. Apple Newton) to avail themselves of the ATIS services provided by Genesis. This extended abstract presents an overview of the goals and preliminary design of Genesis. We believe that ATIS provides a very important commercial application of mobile computing and can bring mobile computing to mass markets. *This paper focuses on describing the application domain rather than evaluating candidate solutions.*

1 INTRODUCTION

Advanced Traveler Information Systems (ATIS) is one facet of the Intelligent Vehicle Highway System (IVHS) [2], which is currently being developed to improve the safety and efficiency of automobile travel. ATIS assists travelers with planning, perception, analysis and decision-making to improve the con-

¹Intelligent Vehicle Highway Systems are also known as Intelligent Transportation Systems from 1994.

venience, safety and efficiency of travel [7, 18, 6, 21, 16]. ATIS applications create a shared resource for efficient mobile computation and data integration. As shown in figure 1, ATIS obtains information from different sources, including traffic reports, scheduled traffic events, sensors and maps, etc. Periodic sensor data might lead to high update rates. The clients of the database include drivers on the road, mobile persons with handheld or portable personal communication devices (PCDs), and users who access information via computers at home, the office, shopping mall or information center. A large number of travelers will query databases over the wireless communication channel to seek traffic information and driving advisories. Mobile computing is the core of ATIS.

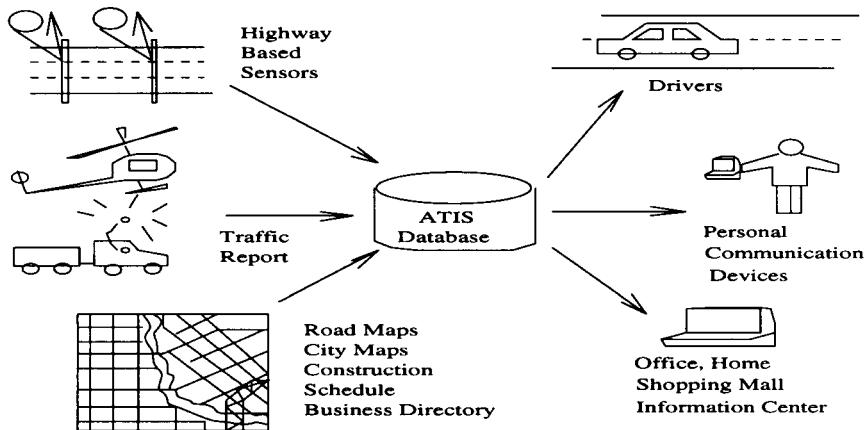


Figure 1 Data Sources and Mobile Computing for an ATIS database

1.1 ATIS Services

ATIS provides up-to-the minute information on weather and road conditions, detours, construction zones, bus schedules and parking. Travel information is available before your trip or en-route. ATIS performs a variety of functions, including navigation using digital maps; route selection and guidance; information on services such as gas stations, restaurants, and hospitals; and real time traffic information through communication between drivers and the Advanced Traffic Management System (ATMS). ATMS collects information primarily via sensors and reporters rather than via communication with drivers. These functions are categorized into five services [1].

(a) The **Traveler Information Service** includes the business directory and travel information databases which are integrated with the road map database to provide information about tourist attractions, hotels, restaurants, etc. (b) The **Pre-trip Travel Information Service** provides information that is useful before the trip begins, such as road and weather conditions. The trip planning service also allows the user to set trip routes using roadway or MTC (Metropolitan Transit Commission) bus routes, plan the trip schedule, and receive estimated travel duration and known advisories for the planned trip. (c) The **Route Guidance Service** recommends the most favorable routes from a starting location to any chosen destination. "Favorable" may represent the shortest distance, minimum travel time, etc. The route guidance service guides the user from a location to the destination. (d) The **En-route Driver Advisory Service** provides drivers (in-vehicle or roadside) with information such as construction zones, traffic congestion, traffic incidents, weather conditions and detours. (e) The **Emergency Notification and Personal Security Service** uses pagers, cellular phones or callboxes for "mayday" purposes. When initiated by the driver, or initiated automatically in case of accident, this feature will provide the capability of summoning emergency assistance and provide the vehicle location.

1.2 Mobile Computing in ATIS/IVHS

ATIS will assist many kinds of travel, including the daily commute to work via private/public transportation, occasional visits to points of attraction in a familiar/new geographic area, trips involving multiple points, etc. Since travelers are naturally mobile, mobile computing plays a central role in assisting different kinds of travel and tasks related to travel. For example, driving to a point of attraction in a new city can be facilitated by a portable route-guidance device and navigator (see Table 1 for products) that contains information about road maps, points of interest, traffic restrictions, etc., and is available on a portable mobile computer coupled with a global positioning system and wireless communication. Now that coupling to real-time traffic information, business directories and other ATIS services is possible, mobile computing issues are becoming more important. Several functions and services of ATIS, including route guidance, en-route driver advisory and emergency notification, require mobile computing. Other functions of ATIS, such as the traveler information and pre-trip travel information service, might also require mobile computing.

While many of the current products for route guidance are special-purpose computers, the next generation ATIS products are being designed for general

Product	In-Vehicle Route Guidance	Business Directory
Zexel's GuideStar, Oldsmobile	*	
Siemen's Ali-Scout	*	
Motorola's Arrow, Envoy	*	*
ARDIS	*	*
RAM Mobile Data	*	*
GTE Personal Communication Services	*	*
Sony Mobile Navigation System	*	

Table 1 Navigation and Routing Guidance Products

purpose mobile portable computer platforms. The market for ATIS products is likely to grow rapidly in the US, and a widening market for route guidance systems already exists in Japan [4]. ATIS is an application naturally suited for mobile computing technology.

1.3 ATIS Projects

Many on-going projects are being carried out to develop and test ATIS features [2, 3, 7]. Some ATIS operational test projects have been completed [2, 3]. We describe some of the representative projects. More comprehensive survey of ATIS projects can be found in [2, 3]. TravTek was a joint public sector-private sector project to develop, test and evaluate an ATIS in Orlando, Florida. TravTek provided traffic congestion information, tourist information and route guidance, etc. Route guidance reflected real-time traffic conditions in the TravTek traffic network. A traffic management center collected real-time traffic information from various sources and transmitted it to test vehicles via digital data radio broadcasts.

Pathfinder was a cooperative effort by California Department of Transportation, Federal Highway Authority (FHWA), and General Motors. In the Los Angeles Smart Corridor, Pathfinder provided drivers of specially equipped in-vehicle navigation systems with real-time traffic information, and suggested alternate routes. A control center managed the communication, detected traffic density and vehicle speeds, and transmitted congestion information to the special vehicles in the form of an electronic map on a display screen or digital voice.

TravInfo is a joint project between the California Department of Transportation, the Bay area ad hoc IVHS committee and FHWA. TravInfo will provide comprehensive traveler information both before and during trips. A multi-modal transportation information center will collect transportation information from various sources and make the information available to the general public, public agencies and commercial vendors.

SWIFT (Seattle Wide-Area Information for Travelers) is a joint public sector-private sector project to test an ATIS in Seattle, Washington. SWIFT will test the delivery of traveler information via three devices: the Seiko Receptor Message Watch, an in-vehicle FM subcarrier radio, and a palm-top computer. SWIFT will also expand ATIS service currently being developed under the Seattle Smart Traveler project.

A variety of projects are also underway in Europe and Japan [2, 3, 7]. ALI-SCOUT/LISB uses infrared transmitters and receivers to transfer navigation information between roadside beacons and equipped vehicles. The Autoguide being developed in British is similar to ALI-SCOUT. Other collaborative projects being developed in Europe are the DRIVE with the goal of supporting such needs as route guidance, improved travel safety and efficiency, etc. CACS, RACS and AMTICS are projects being developed in Japan.

1.4 Genesis, Related Work and Scope

Genesis [1, 22] is an ATIS operational test project currently being developed in Minnesota. It is a partnership between the University of Minnesota, the Minnesota Department of Transportation and private sector companies. The goal of Genesis is to test the effectiveness of an advanced portable traveler information service to provide comprehensive, real-time travel data. Traveler information, including trip planning, transit, traffic and parking information, etc., will be provided via a family of fully portable personal communications devices (PCDs). Initially, three types of devices are being evaluated: an alphanumeric pager, a notebook computer and a personal digital assistant. PCDs are being evaluated in a multi-phase operational test throughout the Minneapolis - St. Paul metropolitan area. The first phase is a pilot project providing reports on incidents only. Later phases will examine the provision of additional information via PCDs.

The TravLink project will implement an ATIS and Advanced Public Transportation System (APTS) along the I-394 corridor in downtown Minneapolis.

lis. TravLink will provide real time traffic information and transit schedules through a combination of kiosks and terminals at home, work, transit stations and shopping centers.

Genesis is likely to open up an application requiring mobile database technology. Mobile database technology such as caching, data replication, indexing strategies, and tracking strategies including location updates and queries for mobile users, etc. [12, 5, 10, 14, 13, 11], are relevant to the design of Genesis. We note that Genesis is still in the preliminary design phase, so it is premature to compare it with other related work in mobile databases. This paper attempts to describe an important application which may showcase mobile database technology. We focus on describing the application rather than evaluating the technology.

Outline : Section 2 defines the ATIS system components. Section 3 presents data management in Genesis. Query processing issues in Genesis are discussed in section 4. Finally, section 5 summarizes our discussions.

2 GENESIS SYSTEM

In this section, we first describe the components of the Genesis system and then present the Genesis system functions. Genesis is a dynamic project such that, as the program progresses, the functions and capabilities of Genesis will need to continually improve to accommodate advances in technology and changing user requirements. To provide a more comprehensive understanding of the ATIS system, we not only describe the current functions of Genesis, but also include other ATIS functions which will be the future goals of Genesis.

2.1 Components of Genesis

The Genesis system includes data collection stations such as the Traffic Management Center (TMC) and the Metropolitan Transit Commission (MTC), along with the fixed-end ATIS database server, the wireless communication service provider and mobile PCDs (Personal Communication Devices). Figure 2 shows an operational view of Genesis [1].

The TMC is the communications control and computer center for managing traffic on Twin Cities Metropolitan Area Freeways. The TMC is equipped

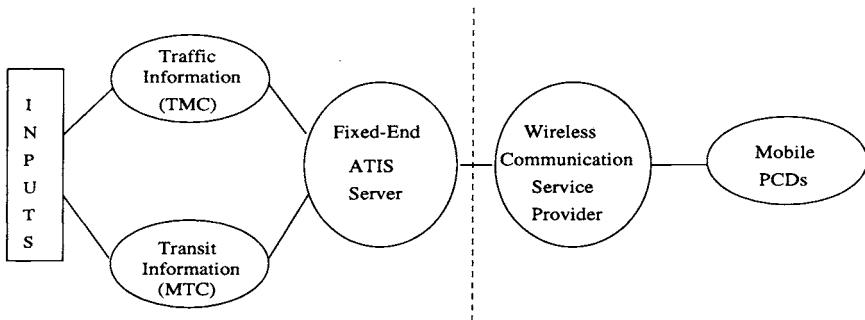


Figure 2 Genesis Operational View

with video and radio monitoring and broadcasting equipment, as well as traffic management workstations. The TMC uses networks of sensors, communication channels, and information sources to acquire traffic data. The MTC collects data about current locations of busses (public transportation) and schedule exceptions.

The fixed-end ATIS server is the functional data processing element. Real-time data is collected, converted, processed and stored by the fixed-end server. Data is transmitted to the mobile clients via a communication service. Data transmission occurs at the request of mobile clients, or is sent by the fixed-end server as a result of a pre-determined data transaction.

The communication service provider has the responsibility of transmitting data from the fixed-end server to mobile clients (PCDs) and transmitting requests from the clients to the server. PCDs will, through a tightly coupled interface, communicate to the Genesis fixed-end server via the communication service provider. PCDs are divided into three categories, an alphanumeric pager, a hand-held PDA (Personal Device Assistant) and a PDA off-the shelf information device.

2.2 Genesis System Functionality

Figure 3 shows the functions of the Genesis system, including application functions, communications, user interfaces, I/O drivers and operating system, and other ATIS functions. Genesis application functions include the user profile,

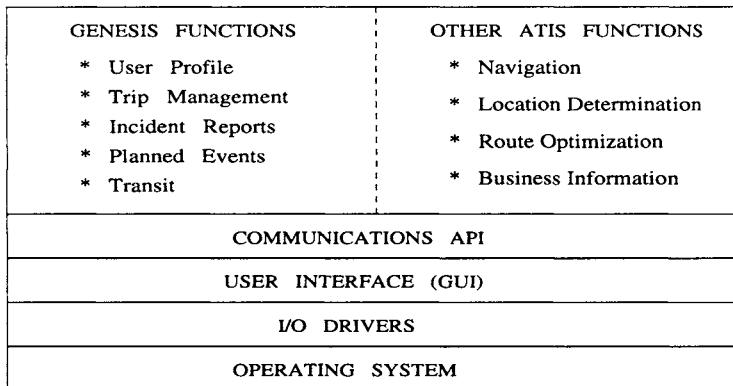


Figure 3 Detailed Functional Decomposition

trip planning, trip scheduling, trip status, travel time advisory, real-time advisory, planned events and transit applications [1].

The **User Profile** function allows the user to maintain default travel time preferences, alarm parameters and advisory recognition preferences. Travel time preferences include desired starting or stopping time of trip, travel duration, etc. Advisory parameters include travel duration, bus exceptions, real-time, and planned event advisories. User-specific information such as preferred routes, trip origin and destination, etc., are also maintained.

The **Trip Management** function provides trip planning, trip scheduling, trip status, and route layout. Trip planning allows the user to plan trips using primary and alternate routes to destinations, receive estimated travel durations and known advisories for planned trips, etc. Trip scheduling allows the user to select a planned trip as the active trip, receive current and estimated travel duration and known advisories for the active trip, etc. Trip status allows the user to receive travel duration, real-time, planned event, and bus exception advisories, etc. Figure 4 shows the Trip Management function [1] in Genesis.

The **Incident Reports** function receives and displays real-time location-specific incident information from the fixed-end server. Incident information includes potential hazards or delays due to unusual occurrences such as traffic accidents or excessive congestion. Incident reports include travel duration advisories and real-time advisories. A travel duration advisory is the dissemination of the manipulated difference between the current travel duration of a link and the historical average travel duration of the link for the current time period. A real-

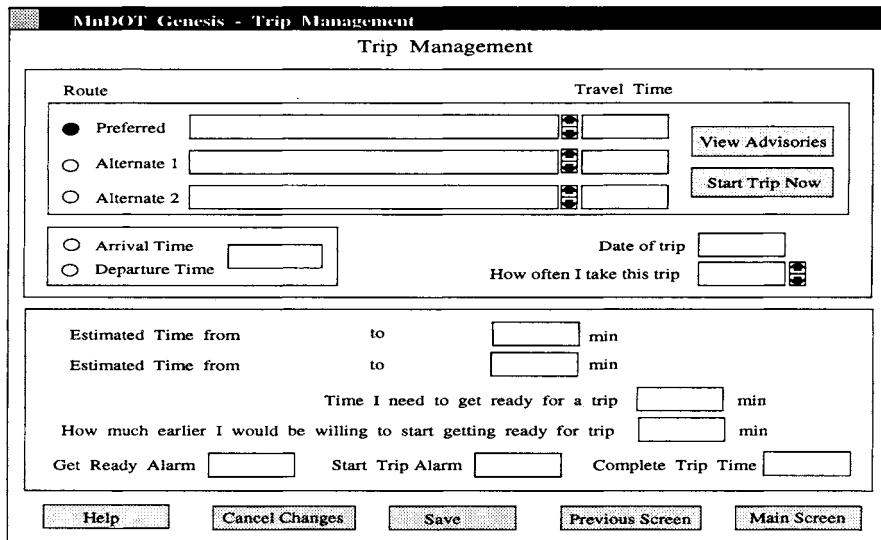


Figure 4 Trip management function in Genesis

time advisory is the dissemination of current abnormal conditions for specific links.

The **Planned Events** function provides the user with a planned event advisory. A planned event advisory is the dissemination of planned abnormal conditions for specific dates and times. The types of conditions to be reported by the fixed-end server are construction, lane closures, maintenance and special events.

The **Transit** function provides bus exception advisories and fixed bus information. A bus exception advisory is the dissemination of the current delay for a specific bus route. Fixed bus information includes the bus schedule, fare and bus route information.

The **Communications** function supports the data transmissions between the fixed-end database server and PCDs. Several approaches to wireless communication are being explored, including Infrared and RF beacon technologies, FM sideband technology, Mobile satellite services, Cellular phone technology and Radio-Frequency (RF) data communication networks [16].

The **User interface** function communicates with the user (driver), accepting requests for service and delivering driving directions and other information.

Direction screens present simple, highly stylized graphics that can be taken in at a glance. The system can be augmented with digitized or synthesized voice for traffic information and route guidance. Alternatively, head-up displays provide more flexible and useful guidance.

The **Navigation** function displays digital road maps. First, the navigation method needs to determine the vehicle's location and path of travel. Then, as the vehicle moves, its position on the screen remains fixed and the map moves around it, providing a view of the roadway ahead. The driver may view a particular area in greater or lesser detail by zooming the display in or out.

The **Location Determination** methods include dead-reckoning, map-matching and GPS (Global Positioning System). Dead-reckoning with map-matching uses a magnetic compass, wheel speed sensors, and algorithms which match the vehicle's motion to the map network. A GPS receiver in the car triangulates the vehicle's position on the basis of transmissions from a satellite network deployed by the U.S. Department of Defense. GPS is most effective when combined with dead-reckoning sensors and the map-matching algorithm.

The **Route Optimization** function determines "favorable" routes from the vehicle's current position to a chosen destination. Route evaluation capabilities are also supported to evaluate a set of alternate routes between origin and destination, based on the current travel-time, congestion, restrictions and other attributes of the transportation network. The in-vehicle processor is able to receive traffic information on incidents, including travel times on affected routes, from the TMC. The system determines if the driver's selected route is affected, calculates a new route if necessary, and informs the driver that a revised route is available.

The **Business Information** function supports various service functions for the business directory and travel information inquiry, etc.

3 DATA MANAGEMENT IN GENESIS

In this section, we describe the data and queries in the Genesis system. Some of the data and queries are not included in the current stage of Genesis design, but are common in the ATIS system, and thus are very likely to be accommodated by Genesis in the future.

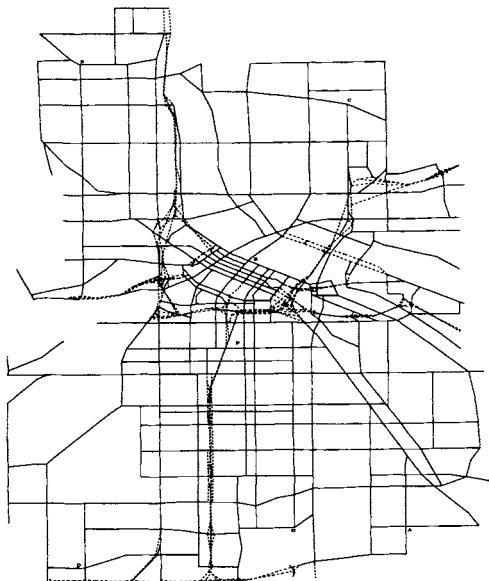


Figure 5 Minneapolis Road Map (major streets)

3.1 Data

We examine the data-set in the Twin Cities area. ATIS information data includes driving and public transportation data that are available to the traveler or commuter. Those data are categorized into the following.

Digital Road Maps are represented as a collection of road segments. Each road segment has a beginning node and an ending node. Since road maps are embedded in a geographical space, each node has x, y coordinates. There are about 100,000 road intersections and 300,000 road segments. Data set sizes are 10 Mbytes for the metropolitan area. Figure 5 shows the major streets in the Minneapolis road map as an example. The road-map of interest to Genesis includes all the roads in the seven counties around the Twin-Cities area.

Transit Information includes Schedules, Route Maps, Fares, Bus Stop Schedules, Real Time Status, Messages, Park and Ride Parking, Downtown HOV parking, Ridesharing, Elderly and Handicapped Service, Transit Itinerary Planning, etc. Data-set sizes are about 30 Kbytes per minute for the metropolitan area.

Traffic and Roadway Information includes Incidents, Construction and Detours, Highway Segment Information, Highway Trip Information, Highway Itinerary Planning, etc. Data set sizes are about 30 Kbytes per minute for simple sensor data and incident information.

The **Business Directory** contains information about hotels, restaurants, and services, etc. Data set sizes are 50 Mbytes for the administrative units of the metropolitan area.

3.2 Queries

User queries include querying traveler service information, pre-trip travel information, route guidance, en-route driving advisories and emergency notification.

Range Queries

Since the transportation networks (road maps) are embedded in geographical space, users might issue a range query. The following example demonstrates a range query. This query utilizes a predefined geographic boundary, the "downtown area", to search for all the road segments within the downtown area which are currently under construction.

Query#1 : Which roads are under construction in the downtown area?

Route Optimization

In a road-map database, the most frequent queries will be route-related queries. A driver might want to know how to get from location A to destination B. This kind of query requires applying path computation algorithms such as the iterative, Dijkstra [8, 15, 19] or A^* [9, 19] to find the routes.

Query#2 : Find the shortest travel-time path from EE/CSci building to the Mall of America?

The sample query demonstrates the use of path-computation algorithms to find routes that satisfy an AGG function. AGG is an aggregate function defined to retrieve the aggregate properties that are specified over the set of routes

in the road networks. Typical AGG functions are shortest distance, minimum travel time, etc. In the IVHS application, there may exist thousands of road intersections between two geographical locations. Therefore, path computation often leads to heavy computation and I/O overhead.

Mixed Queries

Instead of finding routes from a source to a destination, drivers might want to find routes from a source to a set of destinations, as in the following example. The following query first applies range and join queries over road maps and business directory databases to find certain county parks, then applies route computation to find corresponding routes.

Query#3 : Display the locations of all county parks within the Twin Cities area, along with the minimum travel time routes from the EE/CSci building to them.

Route Evaluation

One variant of route computation is route evaluation. A simple example of a query is presented below, where the current travel time for a specific road segment is requested.

Query#4 : Find the travel time on I-35W between 62 and Diamond Lake Road.

The goal of route evaluation is to find the aggregate properties of a given route or set of routes between two locations. These aggregate properties may include travel time, travel distance and traffic congestion information. The set of possible candidate routes are those familiar routes selected by travelers, or the most frequently used routes automatically recorded by ATIS. As shown in figure 6, Genesis provides the route planning function [1], which uses route evaluation to estimate the travel time on routes.

The following two examples illustrate the route evaluation queries. Route evaluation requires less computation and I/O overhead than route computation, since only a very small number of routes are examined.

Query#5 : Evaluate the travel time from Home to the EE/CSci building by my preferred major route I-494.

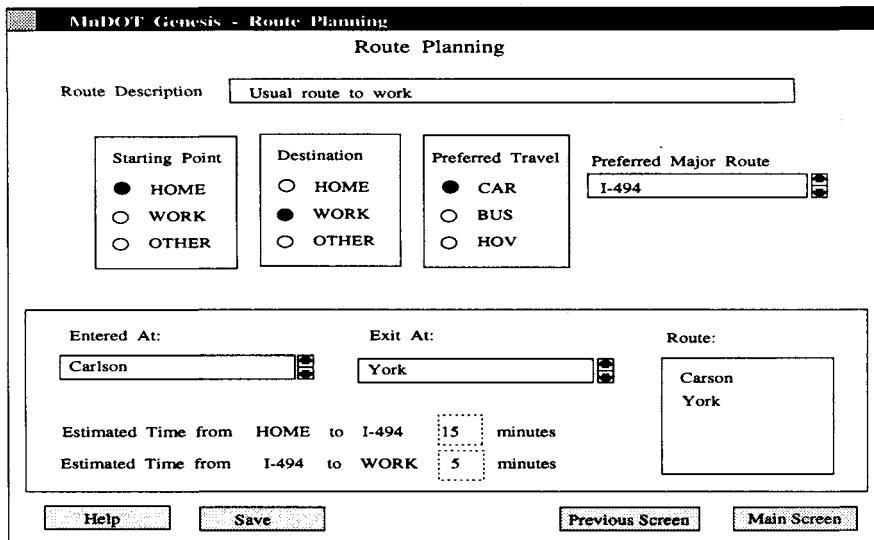


Figure 6 Route planning function in Genesis

Query#6 : Evaluate the travel time for my three preferred routes from Home to EE/CSci building.

Emergency Service

One very important feature of ATIS is its emergency notification and personal security service. Users issue emergency requests by using pagers, cellular phones, or callboxes for "mayday" purposes. The emergency request requires an in-vehicle transceiver to transmit the emergency service request and vehicle location information. The digital messages can be transmitted over the traffic message channels or over the cellular telephone network.

Query#7 : Emergency service request.

3.3 Triggers

In Genesis, events such as traffic accidents, traffic congestion and road hazards need to be persistently monitored. Since events do not occur predictably, it

is more appropriate to model them as triggers so that the system can warn drivers once the system detects an incident. Such Genesis functions as real-time advisory and trip status can be modeled as triggers.

A real-time advisory is the dissemination of current abnormal conditions for specific road segments or a route which is usually the primary route selected by the driver. The type of conditions to be reported are: traffic incidents, traffic congestion, weather conditions and road hazards. For example, the driver would like the system to inform him/her if there is any traffic congestion on his/her primary route home.

Trigger#1 : Whenever any traffic congestion occurs on the primary route, inform me.

In addition, Genesis will provide a route status function that is triggered automatically by a change in traffic conditions on the primary route, or triggered manually by a primary-route status request. Figure 7 shows the trip status function [1] in Genesis.

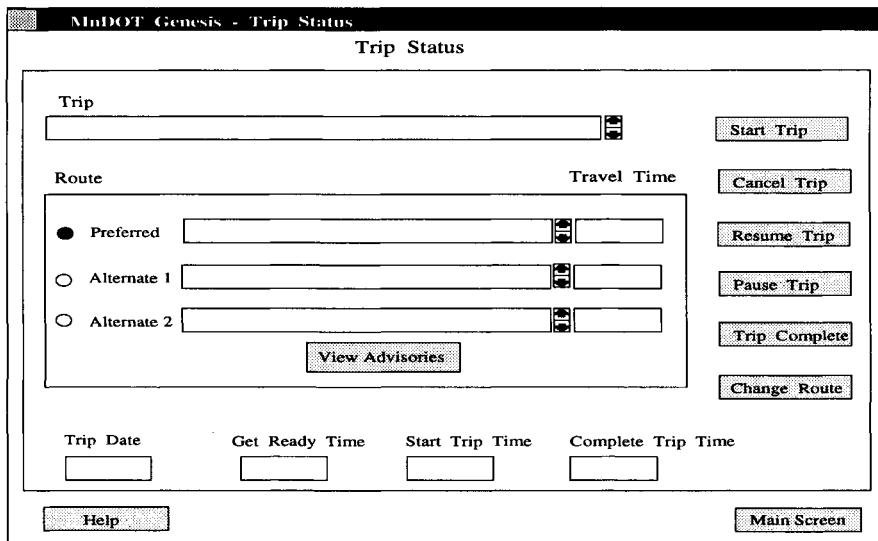


Figure 7 Trip status function in Genesis

A driver might specify the following trigger to change the primary route if the travel duration on the primary route drops below a threshold. This trigger uses route evaluation to find the travel duration for routes.

Trigger#2 : If the travel duration for an alternate route is less than the primary route travel duration by a differential threshold, then suggest the alternate route as the new primary.

3.4 Dynamic Routing Problem

ATIS has to manage constantly changing travel times, which can alter the route of choice during travel. Optimal (e.g. minimum time) route selection by the individual driver, using real-time information disseminated by the TMC, will need to account not only for current traffic conditions but also for predicted traffic conditions and the routes of other drivers in the road network. Drivers provided with real-time traffic information will change selected routes to avoid congestion and incidents of which they are alerted. The current preferred or optimal route might no longer be the best choice as travel proceeds. Path computations need to be continually performed to select the preferred path from the driver's current location to the requested destination. Path computations in ATIS also become time-critical because a lengthy turnaround time may cause the drivers to miss their next turn. For example, path computations must be completed well in advance of the driver reaching the decision point (intersection). The distance to the decision point and the average travel speed may determine the time constraint.

Figure 8 shows a dynamic routing example with given source node A and destination node F. The originally preferred route was A-B-C-D-E-F. Let us assume that an incident occurs on road-segment B-C when the vehicle is at point M, before the driver reaches node B. In order to avoid the incident, the driver may divert to an alternate route. In this example, the system suggests an alternate route B-P-Q-R-F, before the driver reaches the decision point B. The path computation must be performed between point M and node B to alter the preferred route and avoid blockage.

The responsiveness of dynamic route guidance depends upon the cost and efficiency of accessing new information such as incidents and changes in travel-time. It is desirable to explore methods to improve the response-time of incremental route evaluation and incremental route optimization via techniques such as materialization and the maintenance of dependence between events and the ranking of routes.

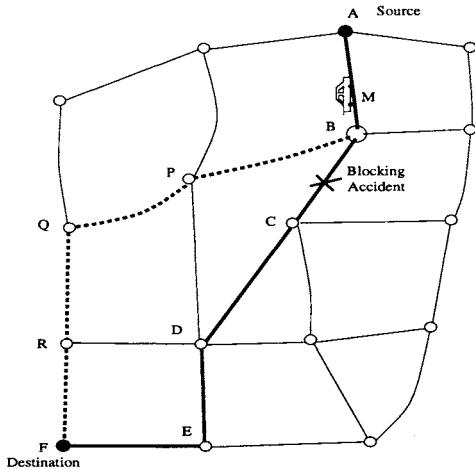


Figure 8 Dynamic Routing Problem in ATIS

4 PERFORMANCE ISSUES IN GENESIS

At present, the preliminary requirement specifications of Genesis has just been finalized. Several different strategies for supporting query processing, advisories, and communications are being considered. We list some of the strategies in this section, without any particular preference or evaluation.

4.1 Metrics of Performance

There are a number of factors that should be considered in designing query processing strategies. Those factors are discussed below.

- **Query Response Time :** Query processing needs to be efficient to guarantee reasonable query response time, especially for en-route real-time advisories.
- **Data Currency :** For example, traffic delay information should not be more than 1 minute old.
- **Scalability :** The system should have the capability of scaling up to accommodate a large number of travelers.

- **Autonomy :** If a communication failure occurs due to weather, tunnels, etc., the system should be able to provide some basic functions.
- **Cost Models :** Should include fixed and variable service charge to the user for communication, the ATIS service, etc., as well as strategies to minimize cost.

4.2 Challenges

In this section, we compare mobile computing in ATIS to other types of mobile computing in the following aspects. Mobile computing involves problems in the access and storage of real-time traffic data, in response to constantly changing traffic information.

- **Highly concurrent real-time access :**
The access to real-time traffic information is highly concurrent. For example, there are 200,000 drivers (on average) on the road during rush hour in the Twin Cities Metropolitan area. Managing a reasonable response time during a large number of concurrent accesses will be a key issue. The other issue will be how to distribute the CPU-process load between the server and the clients. Some ATIS functions such as path optimization queries require CPU-intensive computations. The overhead will be too large for the server to process large numbers of path optimization queries. A more reasonable choice will be to let clients share the load of processing queries such as path optimization queries. However, centralized (server) path optimization can achieve global optimization and avoid having most drivers choose the same routes, which would result in congestion in the near future.
- **Network computations :**
ATIS requires more complex computations such as path evaluations and shortest path computations, besides simple data accesses. Query processing not only needs to consider issues involved in the retrieval of data from wireless limited bandwidths, but also needs to take into account the characteristics of the network data and network computations.
- **Managing triggers for dynamic routing :**
Triggers need to be persistently monitored and reported as soon as possible. Pager networks in Genesis can guarantee the delivery of generic trigger information (e.g. incident) within 2 minutes of occurrence. It would be

interesting to explore techniques to reduce the delay. Another challenge is how to manage custom triggers for the 200,000 drivers during rush hour.

- User interface for drivers :

Drivers have a limited time to interact with the computer during driving. For example, manipulating mouses or keyboards during driving may not be safe. The use of voice recognition, thumb-controlled switches on steering wheels etc., needs to be explored along with headup displays and simple information displays (e.g, icons) to convey the results. Pre-trip information can be more elaborate. Providing effective route guidance and driving advisories through limited-size display screen is also a key issue.

- Bandwidth, caching vs data accuracy :

The communication and caching policies used can take advantage of the data-accuracy requirements of ATIS. Several dynamic attributes in ATIS are continuous functions of time. The frequency of changes in these attributes may be quite high; however, the drivers may not be interested in every change. For example, if travel-time on a road-segment changes by few micro-seconds, the change may not be interesting. Thus, it is not desirable to ensure complete consistency between server database and caches at the mobile clients. Clients need to be informed about traffic changes only when traffic differences exceed some thresholds which might be defined on the basis of various qualities of service provided by ATIS. A higher quality of service will provide more accurate traffic information and require higher communication overhead, which implies a higher charge. ATIS should be able to provide different qualities of service to meet different clients' needs.

- Disconnection :

It is often assumed that mobile clients may frequently be disconnected to save power [5, 12], and effective caching algorithms need to be designed based on the disconnection time. Extended disconnection to save power may not be an issue in ATIS, since the vehicle can provide adequate power. However, disconnection may occur due to tunnels and other obstructions to communications. It would be interesting to characterize disconnection in ATIS environments to select appropriate caching algorithms.

- Power consumption :

Power consumption is not an issue in ATIS because computers can be powered using the electricity supplied by the vehicle. For users carrying portable palmtops, energy efficient data management techniques are often considered to save power consumption [12]. However, this issue is less critical in ATIS.

- While issues in location management, data replication and caching strategies, etc., are being explored now, these have not been linked to important real-world ATIS applications which will be used by a large number of mobile clients such as commuters and travelers. An ATIS application benchmark can be quite useful towards comparing methods in mobile databases.

4.3 Strategies for Query Processing

Standard Client-Server

Query processing is performed on the servers. A driver (client) issues a query to the server. The server processes the query, then sends back the query result. This strategy centralizes the database operations on the servers and distributes only application and interface processing to the clients. Since the query processing is done in the server, the PCD can be simple. For an ATIS with a large number of drivers, this strategy will place a very heavy burden on the communication system and servers. Therefore, fixed-end server design becomes one of the main issues.

The main computations in ATIS are route computation and route evaluation. In a page-based environment, data is stored in the system's secondary storage, which consists of fixed, uniform-size pages. The main overhead during route computation or route evaluation is disk I/O. We have evaluated the performance of path computation algorithms on road map databases [19]. The choice of clustering techniques affects the performance of query processing [20]. Lastly, parallel processing [17, 23] may be used to scale up to a realistic number of traveling events and to provide each with a reasonable response time.

Enhanced Client-Server

Query processing can be executed locally by the clients (PCDs). The work load in the server is reduced by distributing database operations to clients. Clients cache data and query results in local disk storage that is available to clients. This strategy requires the PCD to be capable of processing queries such as route guidance, route computation and evaluation, etc. Caching strategies using broadcasting and signature, etc., are discussed in [5].

In an enhanced client-server architecture, the client will request data from the server if data is not available in the client caches. In a page-based environment, the server will not just return the requested data, but will return all data stored in the same page as the requested data. After the client receives the page containing the requested data, the client can access all the data in that page. A clustering technique that can reduce the number of pages accessed from secondary storage during route evaluation will also reduce the communication load and server overhead [20].

4.4 Strategies to Handle Triggers

As described in section 3.3, triggers are used to model events that need to be persistently monitored. We discuss strategies to handle triggers in the following.

- Triggers are placed at the server site. Once the server monitors any trigger in effect, the server transmits warning or advisory messages to the specific driver or broadcasts the warning/advisory messages.
- Triggers are cached in the client (PCD) site. The server broadcasts incident data and the client caches broadcast data. The client checks the triggers for any effect on the primary route and provides warnings or advisories if the trigger is activated.
- A hybrid of the above two methods is to put common and important triggers (e.g. severe weather conditions or earthquakes) in the server site, and to put driver-specific triggers (e.g. If traffic congestion occurs on the primary route, then suggest alternate routes) in the client site.

4.5 Broadcasting Strategies

When considering the periodic broadcasting of dynamic traffic information, the broadcast period (T) needs to be carefully chosen. A longer T lowers the communication load, but provides less accurate (up-to-time) traffic information to the client. In addition, since ATIS often covers very wide geographic areas and since many vehicles participate, data volume to be broadcast will be very large. We discuss the following possible approaches to reduce data volume and thus reduce the communication load.

Decompose Geographically : People driving in the northern area might not need traffic information in the southern area. Therefore, server coverage could be geographically decomposed into different zones or cells, such that each zone has a particular server site to serve drivers in that zone. Only traffic information in that particular zone then needs to be broadcast. This scheme will need to handle situations where drivers enter different zones or travel across nearby zones.

Checkpointing : Traffic changes are broadcasted at checkpoints. Traffic information is examined at checkpoints to determine traffic changes, i.e., when traffic difference exceeds thresholds. Increasing thresholds will reduce communication load, but at the same time will decrease the accuracy of the traffic information. Checkpointing operates every t times ($t \ll T$).

5 CONCLUSIONS

ATIS requires that large amounts of data be continuously collected, stored and retrieved. Throughput requirements and the capacity of the wireless communications channel will dictate many of the other parameters. Due to the massive number of drivers and the geographic area to be covered, an ATIS will require high throughput demands on the wireless communications system. Channel capacity requirements for real-time traffic information should be investigated. Efficient wireless communication architectures and technologies need to be explored to support full-scale ATIS features.

Communication overhead, caching strategies, data currency, scalability, computer cost and service charges, etc., need to be carefully evaluated in designing ATIS. In addition, efficient routing algorithms should be investigated for minimum travel time (or other selectable criteria) that takes into account current traffic conditions and traffic network travel times.

ATIS is a very promising application requiring mobile database technology. Since it is limited by current technology, the current stage of Genesis will only test the effectiveness of some ATIS features. As the technology advances, the functions and capabilities of Genesis will continually improve to accommodate advances in technology and changing user requirements. In the future, a wide variety of personal and business related ATIS services will be fully provided that are based on the success of mobile database technology.

Acknowledgment

Genesis represents cooperation among the public and private sector and academia [22]. Public sector institutes such as the Minnesota Department of Transportation and the Federal Highway Authority are involved in the project, along with private sector companies such as IBM Federal Systems Company, BRW, JHK, Battelle, Barrientos and Associates, Motorola etc.

REFERENCES

- [1] "GENESIS : Personal Communication Device". GENESIS 191A321 Document, 1993.
- [2] "Intelligent Vehicle Highway Systems Projects". Department of Transportation, Minnesota Document, March 1994.
- [3] "Intelligent Transportation Systems Projects". Department of Transportation, Minnesota Document, January 1995.
- [4] J. Arlook and Randall Jones. "Tracking IVHS : Where It is and Where It is Going". In *Geo Information Systems*, November/December 1993.
- [5] D. Barbara and T. Imielinski. "Sleepers and Workaholics: Caching Strategies in Mobile Environments". In *Proc. of SIGMOD Conference on Management of Data*, pages 1–12. ACM, 1994.
- [6] J. L. Buxton and et. al. "The Travelpilot: A Second-Generation Automatic Navigation System". *IEEE Trans. on Vehicular Technology*, 40(1):41–44, February 1991.
- [7] W. C. Collier and R. J. Weiland. "Smart Cars, Smart Highways". *IEEE Spectrum*, pages 27–33, April 1994.
- [8] T. H. Cormen, C. E. Leiserson, and R. Rivest. "*Introduction to Algorithms*", chapter 25. The MIT Press, 1990.
- [9] D. Galperin. "On the optimality of A*". *Artificial Intelligence*, 8(1):69–76, 1977.
- [10] Y. Huang, P. Sistla, and O. Wolfson. "Data Replication for Mobile Computers". In *Proc. of SIGMOD Conference on Management of Data*, pages 13–24. ACM, 1994.

- [11] T. Imielinski and B. R. Badrinath. "Querying in Highly Mobile Distributed Environments". In *Proc. of Intl Conference on Very Large Data Bases*, pages 41–52, 1992.
- [12] T. Imielinski and B. R. Badrinath. "Mobile Wireless Computing". *Communication of ACM*, 37(10), 1994.
- [13] T. Imielinski, S. Viswanathan, and B. R. Badrinath. "Energy Efficiency Indexing on Air". In *Proc. of SIGMOD Conference on Management of Data*, pages 25–36. ACM, 1994.
- [14] T. Imielinski, S. Viswanathan, and B. R. Badrinath. "Power Efficiency Filtering of Data on Air". In *Proc. of 4th Intl Conference on Extending Database Technology*, pages 245–258. EDBT, 1994.
- [15] B. Jiang. "I/O Efficiency of Shortest Path Algorithms: An Analysis". In *Proc. of the Intl Conference on Data Engineering*. IEEE, 1992.
- [16] A. M. Kirson. "RF Data Communications Considerations in Advanced Driver Information Systems". *IEEE Trans. on Vehicular Technology*, 40(1):51–55, February 1991.
- [17] D. R. Liu and S. Shekhar. "A Similarity Graph Based Approach to Declustering Problems and its Application towards Parallelizing Grid Files". In *Proc. of the Eleventh Intl Conference on Data Engineering*. IEEE, March 1995.
- [18] J. H. Rillings and R. J. Betsold. "Advanced Driver Information Systems". *IEEE Trans. on Vehicular Technology*, 40(1):31–40, February 1991.
- [19] S. Shekhar, A. Kohli, and M. Coyle. "Path Computation Algorithms for Advanced Traveler Information System". In *Proc. of the Ninth Intl Conference on Data Engineering*, pages 31–39. IEEE, April 1993.
- [20] S. Shekhar and D. R. Liu. "CCAM : A Connectivity-Clustered Access Method for Aggregate Queries on Transportation Networks : A Summary of Results". In *Proc. of the Eleventh Intl Conference on Data Engineering*. IEEE, March 1995, (An extended version is also accepted for IEEE Trans. on Knowledge and Data Engineering).
- [21] R. von Tomkewitsch. "Dynamic Route Guidance and Interactive Transport Management with ALI-SCOUT". *IEEE Trans. on Vehicular Technology*, 40(1):45–50, February 1991.

- [22] James L. Wright, R. Starr, and S. Gargaro. "GENESIS - Information on the Move". In *Proc. of Annual IVHS America Conference*, pages 334–336, 1993.
- [23] Y. Zhou, S. Shekhar, and M. Coyle. "Disk Allocation Methods for Parallelizing Grid Files". In *Proc. of the Tenth Intl Conference on Data Engineering*. IEEE, 1994.

INDEX

A

Active badge, 402, 408, 410, 413, 418, 421, 423
Active document, 36, 376
Ad-hoc networking, 19, 33, 192
AFS (Andrew File System), 30, 372, 508
AMPS, 7, 10, 142–143
Antenna diversity, 4
ARDIS, 12, 300, 627, 702
Audio, 51, 233, 271, 277, 399, 413, 623

B

Battery, 2, 15, 49, 184, 201, 184, 201, 446, 633
Bit-error rate, 4, 275
Broadcast, 15, 53, 149, 155, 184, 210, 231, 299, 331, 387, 595, 621, 702

C

Cache, 25, 67, 107, 134, 155, 299, 331, 368, 434, 445, 509, 587, 618, 655, 717
CD-ROM, 16
CDMA, 8
CDPD, 6, 28, 142, 152, 249, 300, 628, 645
Cellular, 1, 52, 104, 129, 207, 219, 248, 253, 272, 278, 332, 410,

427, 440, 538, 563, 621, 645, 693, 701

Client-server, 18, 21, 35, 260, 302, 333, 341, 358, 520, 531, 564, 665, 688, 719
Coda, 29, 32, 37, 364, 507, 537, 655, 669, 675
Color, 25, 57, 104, 184
Compression, 35, 247, 258, 272, 274, 289, 293, 440, 564, 625, 634
Congestion, 5, 19, 29, 34, 39, 92, 156, 163, 176, 207, 214, 223, 243, 248, 260, 627, 640, 701
CSMA, 199, 201

D

Database, 2, 29, 36, 67, 143, 301, 335, 401, 420, 518, 571, 600, 641, 683, 700
DataMan, 26, 36
DBS, 627, 645
Device driver, 428
Disconnection, 22, 29, 230, 250, 266, 326, 364, 444, 508, 540, 587, 643, 651, 717
Downlink, 14, 23, 56, 301, 626, 642
Doze mode, 16, 305, 317

E

Electronic mail, 2, 15, 37, 77, 81, 95, 425, 621, 659

Embarc, 12, 318
 Energy, 3, 27, 80, 146, 302, 399,
 423, 449, 717
 Ethernet, 54, 63, 81, 232, 245, 393

F

Fading, 4, 233, 274, 623
 Failure, 31, 81, 91, 133, 171, 237,
 265, 431, 444, 507, 540, 580,
 682, 716
 FCC, 11, 14
 FDMA, 7
 File systems, 29, 32, 357, 508, 531,
 537, 563
 Flash-memory, 1, 37
 Flow control, 19, 67, 234, 249, 264,
 441

G

GEOS, 13
 GPS, 6, 410, 423, 693, 708
 GSM, 9, 143, 152, 254, 269, 576,
 589, 645

H

Handoff, 10, 107, 143, 207, 238,
 314, 367, 572, 623
 Hidden terminal, 12

I

IETF, 33, 103, 125, 139, 269, 646
 Indirection, 21, 35, 233, 249, 405,
 412, 667
 InfoPad, 42, 640
 Infrared, 13, 46, 63, 82, 90, 104,
 169, 184, 201, 225, 393, 622,
 703

Interference, 4, 83, 156, 225, 274,
 281, 297
 Internet, 103, 125, 151, 180, 186,
 223, 239, 248, 258, 300, 616,
 621, 644
 IP, 18, 103, 130, 167, 18, 103, 130,
 167, 210, 229, 257, 305, 321,
 442, 460, 597, 620
 IR, 38, 52, 359, 366, 416, 622, 625,
 632, 638–639
 ISDN, 82, 146, 426, 538, 650

K

Kernel, 264, 425, 453, 515, 527,
 542

L

LAN, 1, 11, 53, 145, 157, 176, 201,
 249, 254, 300, 328, 366, 426,
 538, 563, 623, 649, 689
 LEO, 14, 627

M

MAC, 12, 156, 160, 171, 176, 192,
 204, 242
 MACA, 156
 Memory, 1, 16, 64, 77, 81, 95, 199,
 219, 232, 333, 358, 426, 435,
 522, 527, 538, 640, 683
 Mobidata, 42
 Mobile IP, 147, 231, 239
 Modem, 6, 10, 302, 377, 440, 538,
 576, 627, 644
 Mosaic, 89, 237, 377, 386, 393,
 654, 688
 MPEG, 35, 273, 643
 Multicasting, 19, 26, 110, 188, 301,
 412

Multimedia, 271, 297, 444, 574, 593, 624

Multipath fading, 4, 272

N

Nano-cell, 93, 403, 421

Nearcast, 26

Newton, 17, 377, 450, 626, 699

NFS, 237, 442, 508, 567

O

Operating system, 2, 49, 393, 425, 449, 566, 653, 683, 705

OSI, 2, 17, 305, 597, 610, 632

Overlay network, 38, 621, 628, 644

P

Packet loss, 5, 19, 34, 146, 207, 215, 233, 243, 258, 625, 633–634

PCMCIA, 37, 393, 425, 626, 684

PCS, 8, 14, 276, 571, 8, 14, 276, 571

Pen, 17, 32, 45, 58, 122, 259, 326, 348, 358, 367, 452, 520, 545, 665

Personal digital assistant, 1, 45, 50, 573, 652, 703

Picocell, 2, 143, 300

Power, 1, 45, 103, 155, 184, 208, 229, 253, 274, 298, 332, 337, 425, 451, 507, 538, 577, 623, 682, 717

Primary copy, 30

Privacy, 8, 117, 178, 278, 397

PRMA, 15, 304

Proxy, 21, 27, 176, 262, 641

Publishing, 23, 299

Q

Query, 21, 31, 38–39, 77, 106, 299, 356, 430, 595, 654, 700

R

Radio, 3, 49, 104, 151, 209, 274, 298, 356, 421, 532, 592, 598, 622, 661, 684, 702

Rayleigh fading, 4

Recovery, 63, 218, 243, 258, 317, 522, 550–551, 579, 655

Relay network, 644

Replication, 3, 30, 37, 132, 310, 358, 509, 540, 704

Resolution, 25, 30, 46, 127, 179, 184, 289, 287, 398, 414, 453, 514, 566

Routing, 34, 39, 63, 104, 130, 154, 183, 210, 229, 258, 387, 412, 597, 606, 627, 714

RPC, 63, 72, 237, 404, 653

S

Satellite, 1, 12, 104, 275, 300, 333, 440, 621, 707

Scheduling, 16, 35, 322, 449, 647, 667, 684, 706

Security, 21, 36, 121, 130, 178, 279, 440, 511, 538, 553, 641, 653, 712

Serial line connections, 37, 442

Shadows, 4

Signal strength, 4, 233, 633

Socket, 236, 255

Speech, 7, 43, 662, 674, 683, 693

Spin down, 37, 456

Spread spectrum, 9, 11, 209, 626, 690

T

TCP, 19, 114, 147, 165, 207, 229,
255, 269, 443, 628, 659
TDMA, 8
Telephone, 7, 130, 142, 248, 253,
278, 410, 440, 575, 625, 654,
674, 712
Telescript, 654
Transaction, 31, 38, 147, 268, 356,
408, 423, 522, 539, 571, 627,
654, 705
Transport layer, 6, 17, 29, 147,
226, 229, 628, 632

U

Ubiquitous computing, 43, 209,
397, 425
Unix, 36, 63, 210, 232, 253, 404,
425, 508, 539, 551, 683
Uplink, 7, 14, 56, 146, 241, 301,
626, 645, 301, 626, 645
User interface, 57, 365, 391, 444,
652, 688, 705

V

Video, 15, 35, 78, 233, 271, 333,
378, 623–624, 705
Voice, 7, 129, 146, 271, 278, 532,
576, 621, 645, 702
VSAT, 627

W

WaveLAN, 209, 225, 303, 626
Wearable computer, 681
World-Wide Web, 1, 17, 266, 270