Visualization of Network Simulation through a Web Based Interface

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

It is well known that the global Internet is growing into the ubiquitious communication medium for everything from e-business to online entertainment to networked household appliances. New uses of the Internet are being found everyday. As a consequence, Internet traffic is notoriously dynamic, heterogenous, and unpredictable. Observability of Internet traffic is also difficult because of the network size, large traffic volumes and distributed administration. Yet at the same time, we would like to better understand and measure traffic characteristics for the ultimate purpose of controlling traffic to ensure quality of service (QoS) to all applications. This is indeed a fundamental problem which we want to address by our approach of evaluating network performances by means of simulation.

1.2 Motivation

Our long term goal of the project is the evaluation of performance parameters by means of simulation on a realistically modelled network. We shall model networks with different configuration and find out how the network performs under several constraints. From the generation of network models to analysing the simulation results, everything has to be carried out through a well defined interface.

The project is a distributed one, and the responsibilty of this part is to provide a robust, platform independent interface for network topology generation to be used by a network simulator. In our architecture the main interface will be presented to users through java applets in a web page while the simulator will reside in a remote machine running a web server and is capable of running huge simulations. The user is required to create a network in his own machine interactively in the applet interface and submit it for simulation to the Transaction Server and then waits for the result to come as a dump file. The client-side interface is then capable of interpreting the results in visual forms e.g, in the form of bar/pie charts or tables.

1.3 Ideas

Different concepts and ideas are borrowed from network traffic measurement experiments published in IEEE Communications Magazine, May 2000. These include:

The NLANR

The National Laboratory for Applied Network Research [1] is creating a network analysis infrastructure to support network research and engineering of high performance research networks. The NAI includes a passive monitoring project, an active monitoring project, and collection of network management and control data.

The NTOP

NTOP [2] is a Web-based traffic measurement and monitoring application.It was initially written by the authors to tackle performance problems of the campus network backbone because available traffic monitoring tools were not satisfactory. Similar to the UNIX *top* tool that reports processes' CPU usage, the authors needed a simple tool able to measure network traffic and report information about captured packets. Ntop then evolved into a more flexible, extensible and powerful tool as people over the Internet downloaded it and reported problems and suggestions. Its main design goals include:

- Portability across UNIX and non-UNIX (e.g, WIN9x) platforms
- The ability to monitor and manage a network from a remote location without the need to run specific client applications to analyze traffic information
- Minimal requirements (bare operating system), but capable of exploiting platform features if present (e.g, kernel threads)
- The ability to present data in both a character-based terminal and a Web browser
- Traffic analysis output rich in content and easy to read
- User extensibility via dynamically loadable software components (plugins)

The idea is to develop a simple and efficient application kernel able to handle general tasks including packet capture, basic traffic analysis and measurement of network flows. An important feature is that it has an embedded HTTP server for visualizing traffic data without the need to use an ad hoc client application.

Unlike all these tools we have resorted to simulation instead of actual network probing and data collection for input. For this purpose a simulator has been developed which will make the network produce realistic data based on some input parameters. We have borrowed some important concepts from NTOP's design goals, the most important one being the visualizability of network traffic data without ad hoc client applications.

To make a working model we have naturally chosen Java to be the platform of development. But because there are now plugin development kits available on the Internet for free, it is easy to write specific browser plugins for any type of visualizer application. One can write the application in X, and can make it accessible in a web page by writing and installing a custom plugin.

1.4 A novel web-based approach

The complete project is divided into two major parts - the interface and the simulator. The interface is implemented in Java whereas the simulator in C++. The simulator requires a big library like LEDA and also requires huge memory in order to run simulations on very large neworks (of the order of millions of nodes). The interface also is very scalable and subject to modifications quite often. So it is neither possible nor desired that one has to install the whole system in his machine to use it. Instead we can think of a web-based service where the user will be presented the interface in a web page which he can interact with its full potential, create networks in it and submit it to be sent to the simulator through network sockets. More discussion on this issue is done in chapter 4.

Chapter 2

Design issues

2.1 Top-level view

In the topmost level of our objectives we have a rather simple view of the whole system.

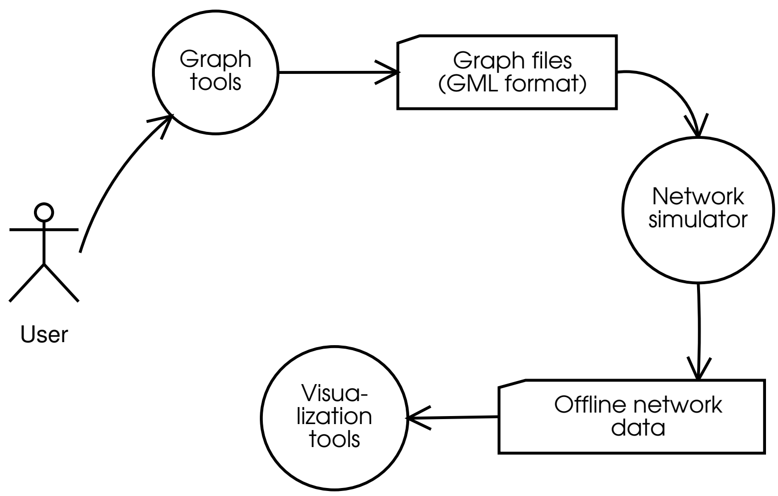

Figure 2.1: Top level view of the simulator system

2.2 Requirements

2.2.1 Network model

The prime requirement of the software is to provide a very flexible, user-friendly and scalable interface for generation of network topologies. We shall often need to create a very large network topology that should be manageable within the limited size of the screen. It is almost impossible to fiddle around with more than say, 30 nodes on the standard screen. So it is highly desired that we organize the network in a hierarchical fashion for visually managing it in an efficient manner. In the current in implementation, the hierarchical network support is fully implemented which means an arbitrary large network can be created simply by means of recursively defining a set of nodes (or supernodes) under a supernode.[1]

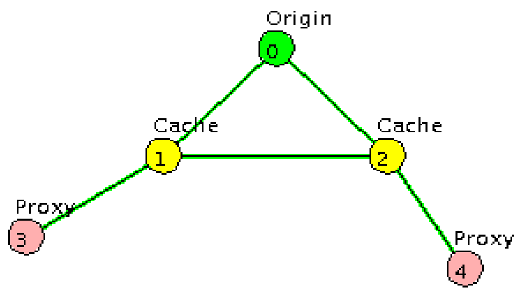

Figure 2.2: A network model

Node

A node in the network is represented by a vertex in the graph. Each vertex has a unique ID and location coordinates. Other network specific parameters embedded in a node are:

- IP Address (String) : May be any alphanumeric name but must be unique
- Server Type (int) : Denotes the server type (1 = Cache, 2 = Origin, 3 = Proxy)
- Data Handling Rate (long) : In Bytes per second
- Transmission Time (double) : In seconds
- Cache Size (long) : In Bytes
- Window Size (int) : Denotes the window size for the sliding window protocol
- Cache Policy (int) : Denotes the caching algorithm (1 = FIFO, 2 = LFU, 3 = LRU)
- Diffusion Policy (int) : Denotes the diffusion algorithm (1 = D4, 2 = Harvest)
- Number of Route Table entries (int) : Size of the route table. Normally $n-1$ for a connected network of n nodes.
- Route Table (Vector) : A set of <dest_IP>:<first_IP> pairs

[1] A supernode is just a collection of nodes that has no attributes except its ID and coordinates. It is just to manage the topology visually and not written in the GML file as a node.

Link

A link in the network is represented by an undirected edge in the graph. The network specific parameters embedded in a link are:

- Bandwidth (long) : Denotes the maximum data rate along the link in Bytes per second
- Propagation Delay (double) : Denotes end-to-end propagation delay in seconds
- Error Probability (double) : Probability of link error

2.2.2 Geographical model

Once a network model is created we can visualize the network situated in a physical area e.g, a state or a country. The hierarchical construction of the network proves to be a very efficient way of representation here as we can put a supernode containing many nodes (and supernodes) in a small bounded area. When we wish to explore a supernode we click on it to expand it to reveal all the underlying nodes including supernodes which are again expandable by clicking on it.

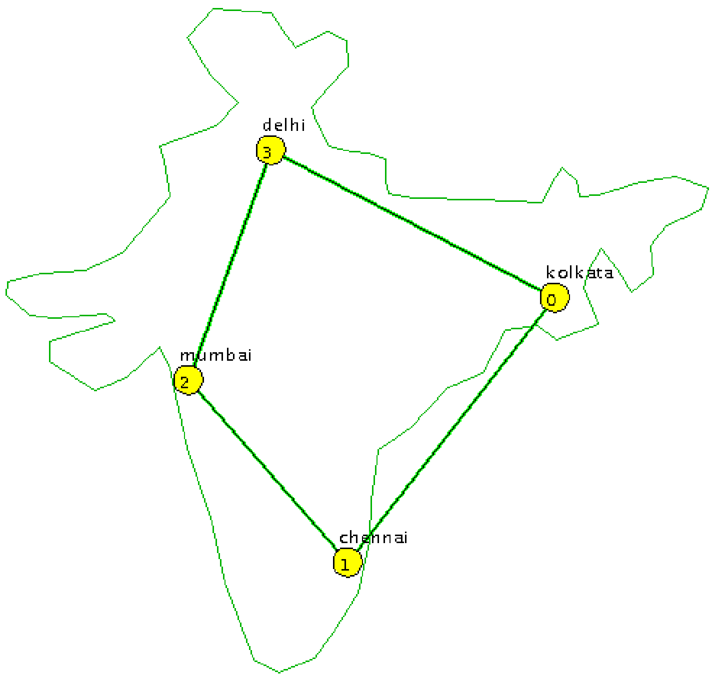

Figure 2.3: A network embedded in a geography

2.3 Essential functionality

The essential functions that the software should perform are:

- Loading and Saving network (Graph) files in either custom format, object format[2] or GML format.

[2] Java serialization format

- Loading and Saving geography (Map) files in object format.
- Concatenating network models created earlier to form a larger network.
- Automatic route table generation for a newly created network.
- Modification of network topology/attributes in an efficient manner.
- Zooming in/out through the network hierarchy or geography hierarchy.
- Printing network/geography/simulation results to printer/file.

2.4 General framework

The framework of the software is shown in figure 2.4. The whole system is developed as an independent Java application. There is an embedded client server model in the communication between the interface and the simulator. The interface is separated out from direct access to the simulator so that the two major components can reside in separate, networked machines.

The simulator should be ideally placed in one powerful machine with huge RAM whereas the interface can be installed in client machines with low resources. But the interface being a highly scalable one, it is desirable that a user has access to it without having to install it in his machine so that whenever a new version is released, all the users can enjoy the new features at the same time. We can think about putting the interface in a web page which will offer the use of the software by means of on-the-fly automatic downloading of program code without the user ever having to bother about it. Thus we introduce the approach of using a software through a web based service.

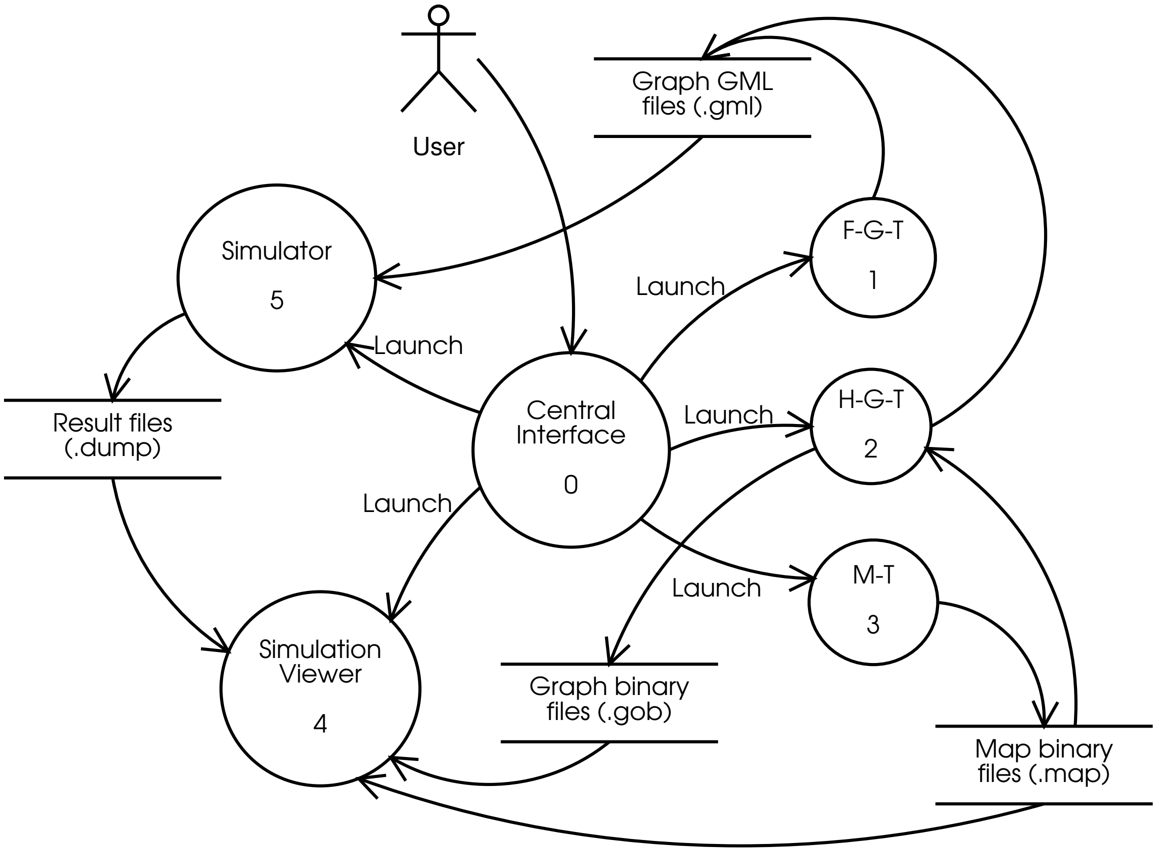

Figure 2.4: Context diagram of the simulator system

Chapter 3

Implementation

3.1 Object oriented design

The software is designed with two very important issues in mind, namely *OOD* and *code reuse*. All common components are designed in a bottom up way, from general to specific. A good software for visualizing network simulations is a big issue and hence should be well designed with some desirable qualities like good code understandability and highly modular design.

3.2 Package classifications

The software is developed by integrating application modules from different packages. We have a *common* package which defines classes used by most other packages, several *tool* packages containing individual component applications, and an *applet* package for applet classes. Figure 3.1 shows the package organization view.

3.2.1 The package `netsim.client.common`

This package defines classes needed by several other packages. The most important classes included here are `Vertex`, `Edge` and `Constants`. It also includes classes like `Utilities`, `Dialogs` offering general purpose functions needed by many other classes of different packages. The class `Utilities` provides the method to automatically generate the route table for a given network (graph) based on the shortest hop computation by Floyd-Warshall's All-pair Shortest Path Algorithm. There are also several abstract classes like `BasicDrawCanvas` and `BasicFileMenu` which forms the basis of the some common components.

3.2.2 The package `netsim.client.fgraphtool`

This package contains classes for the *Flat-Graph-Tool* application used to create simple networks. It defines the class `Graph`, which encapsulates a set of vertices and edges and methods to operate on them. All the mouse events for the drawing are handled by the class `DrawCanvas` which is a subclass of the class `BasicDrawCanvas` of package `netsim.client.common`.

3.2.3 The package `netsim.client.hgraphtool`

This package contains classes for the *Hierarchical-Graph-Tool* application used to create complex hierarchical networks. It defines the class `HGraph`, which is a subclass of the class `Graph`, and encapsulates a set of vertices, edges and hgraphs. All the methods involving a query on a `HGraph` object are inherently recursive because of the structure of the class `HGraph`.

3.2.4 The package `netsim.client.maptool`

This package contains classes for the *Map-Tool* application used to create geographical maps to be shown under a network. It defines the class `GMap` which denotes a hierarchical map and encapsulates a set of boundary points and a set of gmaps. Like the class `HGraph`, all the methods involving a `GMap` object are recursive in nature.

3.2.5 The package `netsim.client.simview`

This package contains classes for the *Simulation-Viewer* application used for the visualization of networks, maps and simulation data.

3.2.6 The package `netsim.client.main`

This package contains classes for the main application.

3.2.7 The package `netsim.client.appletmain`

This package contains the applet class needed to put the main application in an applet.

3.2.8 The package `netsim.server`

This package contains the only class GMLTransactionServer which is a multithreaded server program capable of launching the simulator in a separate process.

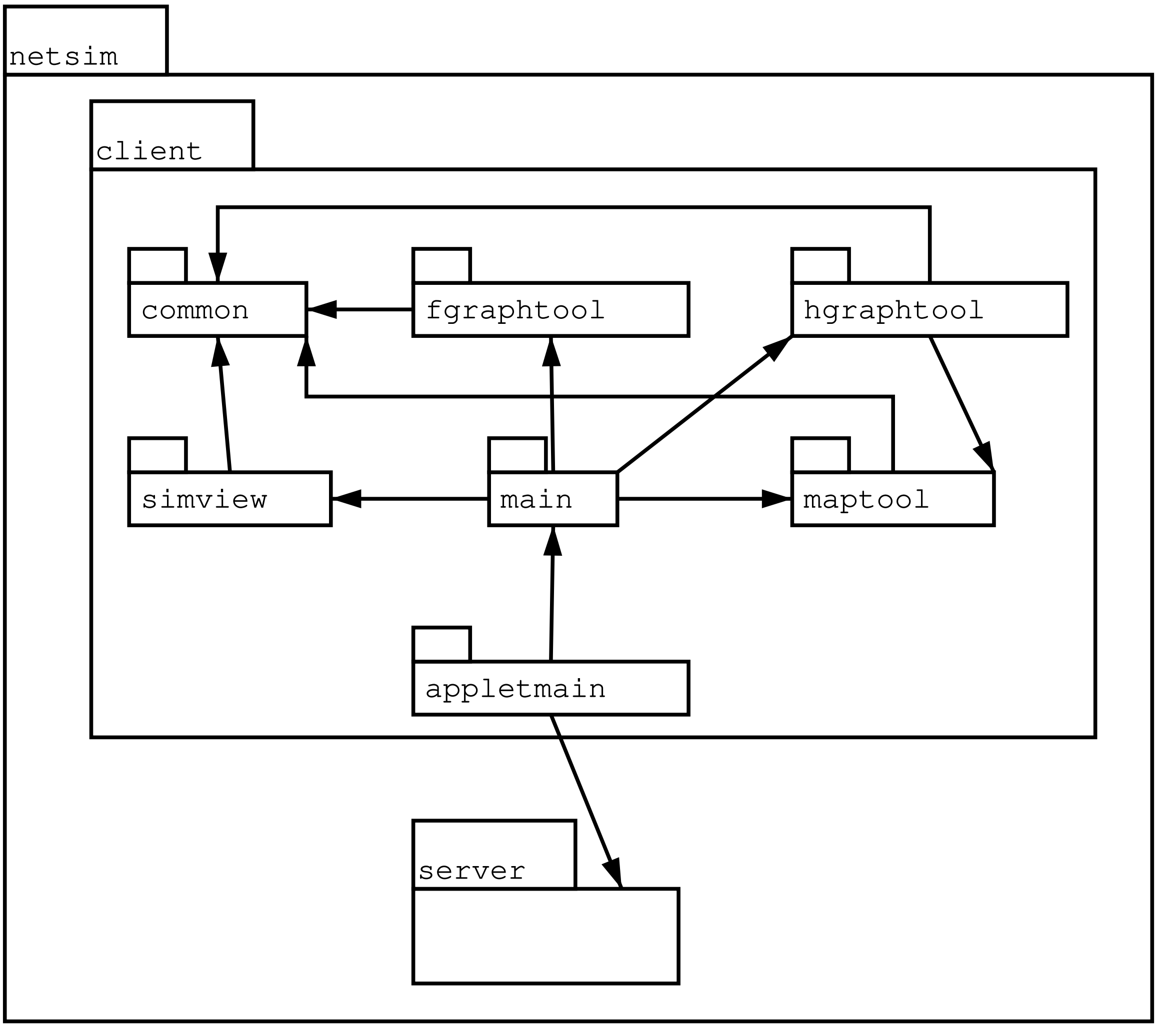

Figure 3.1: Package organization view

Chapter 4

Issues on web based service

Here we introduce a new concept of *resource unaware computing* after making a working model based on the latest advancement in web technologies. We discuss about Java applets and servlets which we have used to achieve part of our goals.

4.1 Java applet technology

A java applet is a *client-side* java program whose code is automatically dowloaded from the web server when the applet enabled page is opened from a browser. The user can interact with the applet just like an application. There are severe security restrictions for an applet to have access to any resource on the client's system but there are ways to grant permission to an applet to write to the client's filesystem. A brief discussion on how to create a policy to grant such permission to an applet is done in the appendix.

4.2 Java servlet technology

A java servlet is a *server-side* java program which can process standard HTTP requests e.g, GET, POST from a HTML form. It is basically a substitute for the now obsolete CGI scripts. When a client clicks on the submit button on a html form, the content of the form can be sent to a servlet by the IP address of the machine where it resides and the port where a servlet listener program (a server program supplied with the servlet development kit) listens for connections. Now the servlet being on the server side, it has the capability to run processes in the server if a client requests so.

Our motivation to use servlets is that suppose the transaction server program is not running on the web server. So how will a client submit a network file ? As a solution we then present a servlet enabled form to the client with a button to activate the transaction server. Figure 4.1 shows the working of a servlet. Of course the servlet listener program should be listening at a specific port on the web server. Now when the client tries to activate the transaction server the servlet at the other end tries to run the transaction server process. If it is already running then a simple exception is generated and the client is informed that the transaction server is already running. So we make sure that only on transaction server process runs on the web server.

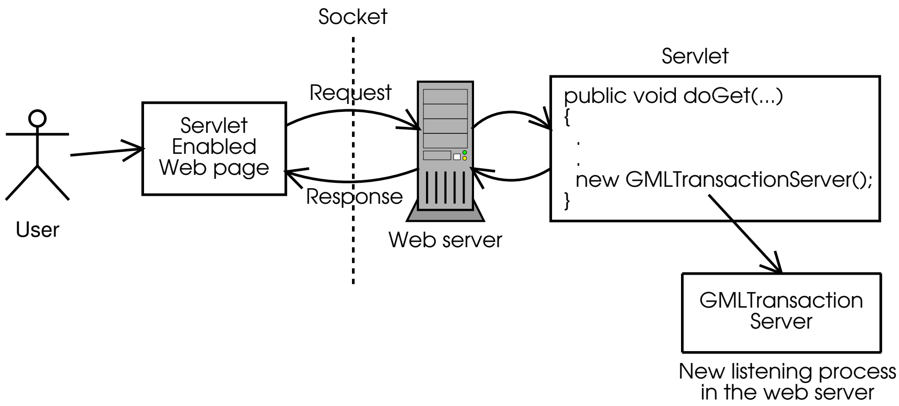

Figure 4.1: Role of a servlet in starting a process at the web server end

Our configuration

In our experiments, we have set the servlet listener port to 8888, GML transaction server port to 8151. The servlet listener must be listening in the web server machine. When the submit button is clicked in the servlet page the servlet residing at the web server end is contacted by the servlet listener. Then the servlet invokes its doGet() method which in turn attempts to run the GML transaction server at port 8151 and becomes successful if it is not running already. Then it sends an appropriate reply back to the client's browser.

4.3 The client-server model

An applet can always connect to the server it is downloaded from, whereas for other servers a successful socket connection will depend on the existence of a firewall. So we can have a small server program running on the webserver where the applet-enabled page resides. We call the server a *GML Transaction Server* which has the responsibility to accept a GML format file from the applet, run the simulator process on it, and then send the result (dump) file back to the client. For this purpose we must have the simulator program on the same machine as the transaction server.

Since the simulator is a native binary program, we run it from java by the *Runtime* class as follows:

```
Runtime r = Runtime.getRuntime();
Process proc = r.exec(new String[]{“simulator”, <gml file>”,
                                   “<dump file>”, “<params>”});
proc.waitFor();    // waits for the process to finish
```

4.3.1 Concurrency of transaction processes

Using the servlet mechanism to activate a transaction server process at the server end by any client we make sure that only one such process runs at any instant. Now the obvious question that arises is how to handle multiple transaction requests at the same time or to process one before finishing the previous one ? An efficient solution is obtained by incorporating multithreading technique into the transaction server. Whenever a new request comes, the main process creates a new thread of execution and hands over to it all the responsibility to handle the request. We achieve this in the following way:

```
while(true)
{
  try {
  Socket client = _server.accept();  // blocking call
  new TransactionThread(client);
  } catch(Exception e) {}
}
```

4.3.2 Security issues

When a client sends a network file to the transaction server a question about file name space arises naturally. If two clients send files of same name at the same time then how does the transaction server store them ? Here we use the utility to generate temporary file names provided by any programming language. We create a temporary file for storing the input GML file and another temporaray file for the DUMP file to be generated by the simulation. Then we send the dump file to the client with the name equal to the input file name with a `.dump` suffix added to it.

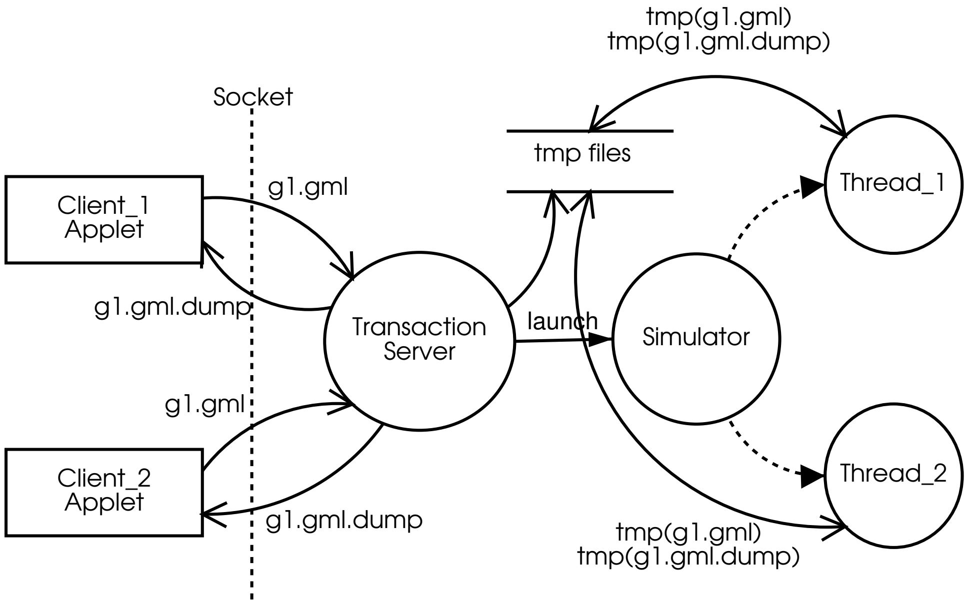

Figure 4.2: The client-server model

4.4 The protocol used

The client (applet) communicates with the server (transaction server) according to a definite ASCII string based protocol. An overview of the protocol is shown in figure 4.3. After making connection, the client sends the GML filename followed by the file contents and then a special string “END” to signify end of file. Then it sends the global parameters of the simulation as strings. If the server receives everything correctly it sends the string “SUCCESS” as acknowledgement. It then launches the simulator in a separate blocking process and waits for its completion. After that it sends the dump file name to the client followed by the dump file contents followed by "END". The client saves the dump file after successfully reading the contents from the socket.

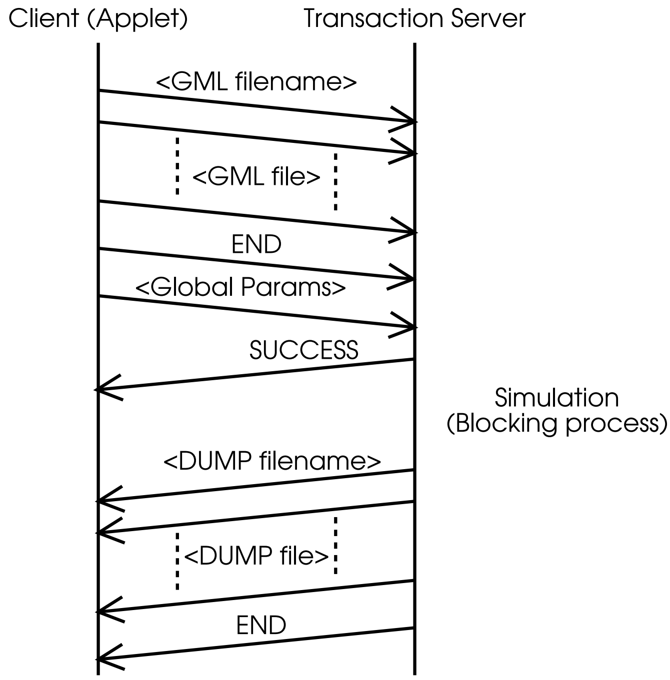

Figure 4.3: The protocol used in the client-server model

Chapter 5

Interaction with the developed tools

5.1 Flat-Graph-Tool

The Flat-Graph-Tool is used to create flat network topologies. Because hierarchy is not supported here the topology should be small to be manageable. Some basic utilities provided by the tool are:

- Load a graph from a custom format
- Save a graph to both custom and GML format
- Zooming in/out for navigating through a large graph
- Efficient editing of node/link parameters by using popup menus on right click
- Automatic generation of route table

5.2 Hierarchical-Graph-Tool

The Hierarchical-Graph-Tool is developed to extend the functionality of the Flat-Graph-Tool. It enables to generate an arbitrarily large yet easy to manage hierarchical network topology. But whatever be the hierarchy, only the underlying flat topology is saved in GML format to be taken as input by the simulator. The extra functionalities supported by this tool are:

- Load a hierarchical graph from object format
- Save a hierarchical graph and map together in object format. This is the only way of storing the hierarchy information of the graph, because whenever saving is done in custom or GML format the hierarchy information is lost.
- Load a map from object format
- Expand/Collapse supernodes to reveal/conceal everything beneath it in the hierarchy (see figure 5.3)

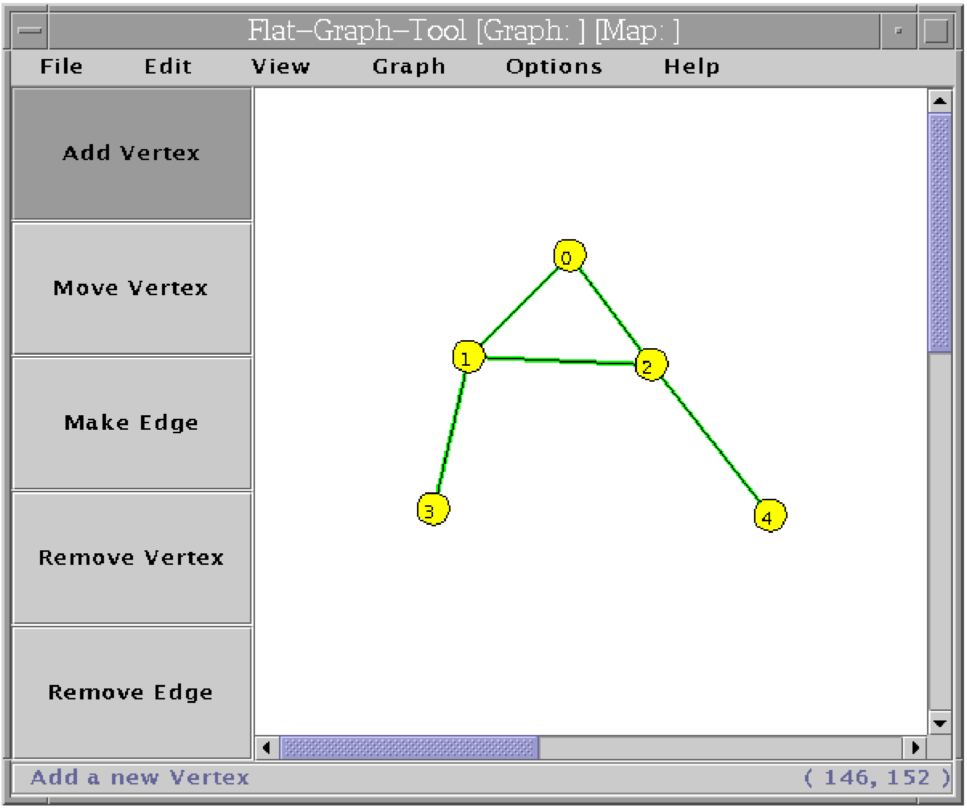

Figure 5.1: Flat-Graph-Tool

5.3 Map-Tool

The Map-Tool is used to interactively create a hierarchical geography by specifying the boundary points of regions. A tree view is incorporated to visualize the hierarchy information of the regions. Clicking on a tree node will select the corresponding region alongwith all its sub-regions. The basic operations supported in this tool include:

- Load and Save a geography (map) in object format
- Zooming in/out for navigating through a large and detailed geography
- Dragging along the boundary points of a region

5.4 Simulation-Viewer

The Simulation-Viewer is basically equivalent to the Hierarchical-Graph-Tool except the editing functionality. Once a network topology and a geography is carefully created and saved we need a read-only interface to handle them. Here we added the visualization features like the utilities to view bar/pie charts based on a dump file got after a simulation. The node and link parameters we visualize here are:

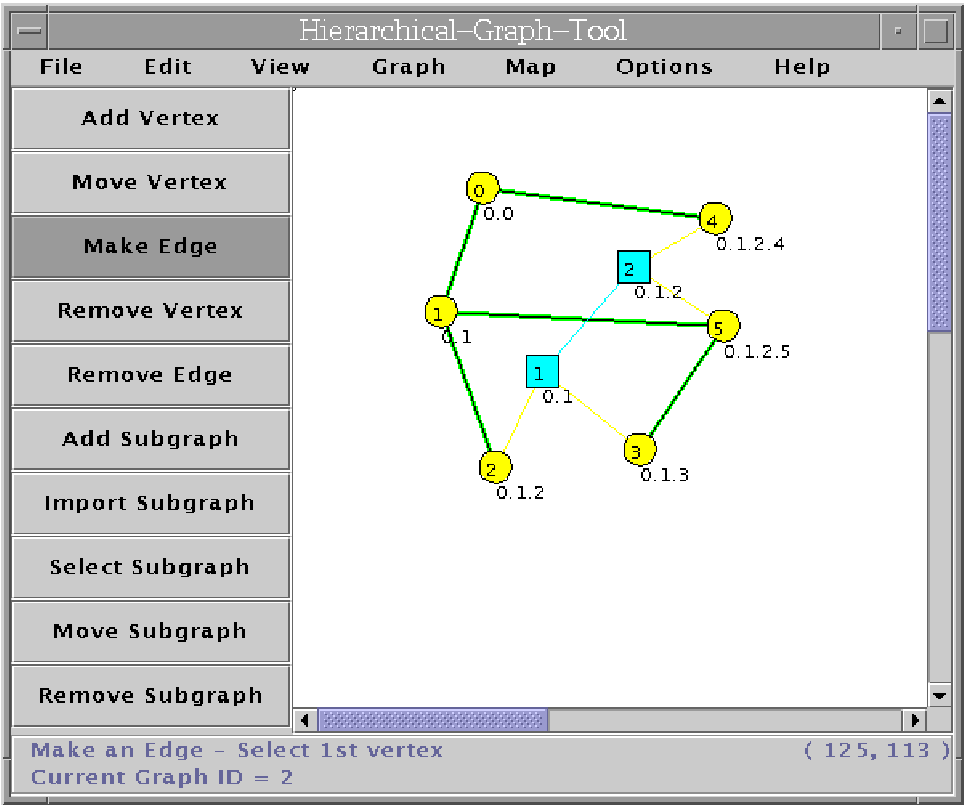

Figure 5.2: Hierarchical-Graph-Tool

Node parameters

- Average latency
- Number of hits
- Number of documents retreived
- Number of request events
- Total bytes downloaded

Link parameters

- Total number of packets sent
- Total number of packets dropped
- Number of packets sent from lower ID node to higher ID node
- Number of packets sent from higher ID node to lower ID node

A query utility is added in this tool to perform queries like "what set of nodes are in a given region ?" This will be useful if one is interested about regional characteristics, instead of focussing the attention to individual nodes.

We have options for choosing from bar/pie chart as the visualization method. which shows a relative comparison of all the nodes/links for any of the parameters listed above. There is a table viewing utility which shows all the node/link parameters for all nodes/links in a tabular format.

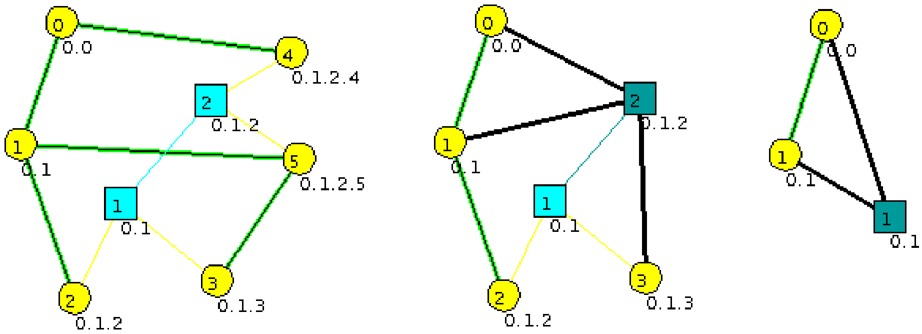

Figure 5.3: Collapsing/expanding a supernode

5.5 Main Application

All the component applications developed so far requires launching from command prompt. To combine all of them under a single interface a main GUI is developed. This also gave us an added advantage - an very small applet program could be written to launch the main GUI within it to be put in a web page. Also we could incorporate the setting of global simulation parameters as given below:

- Packet size : The size of packets in KBytes (default 4096 KByte)
- Page count : The maximum aggregate request rate for the page(s) from/across the network (default 20)
- Window size : The window size to be used in the sliding window protocol (default 64, normally 2^n)
- Event generation time : The time in seconds during which events will be generated (default 5)
- Halt time : The maximum time of simulation in seconds (default 10 for small simulations)

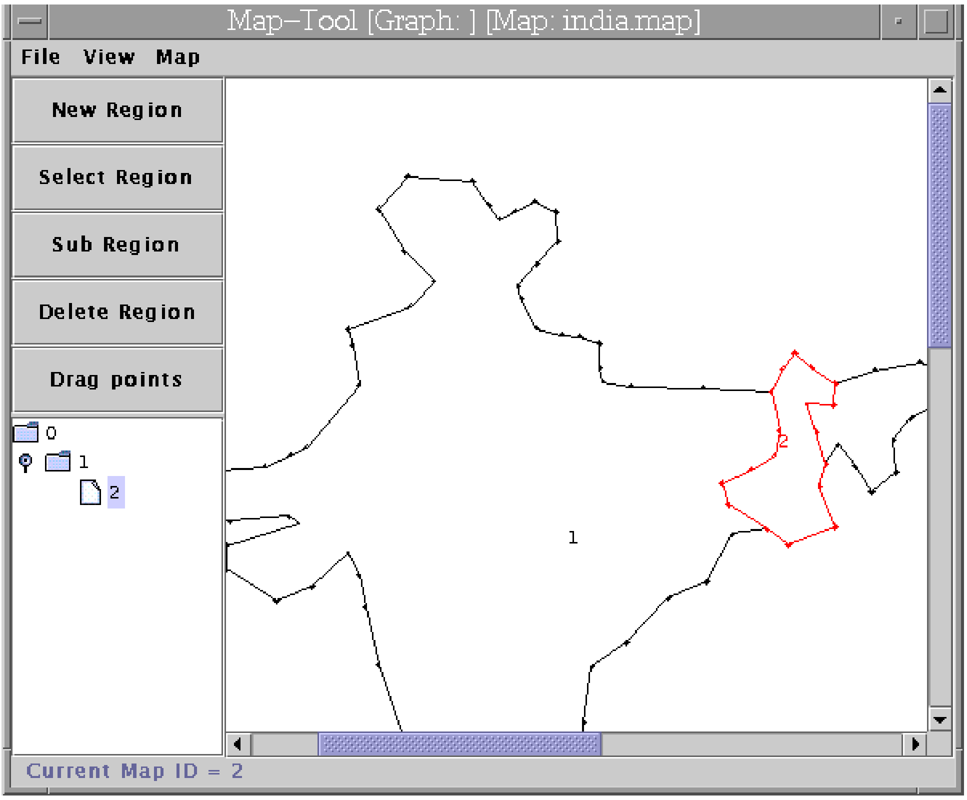

Figure 5.4: Map-Tool

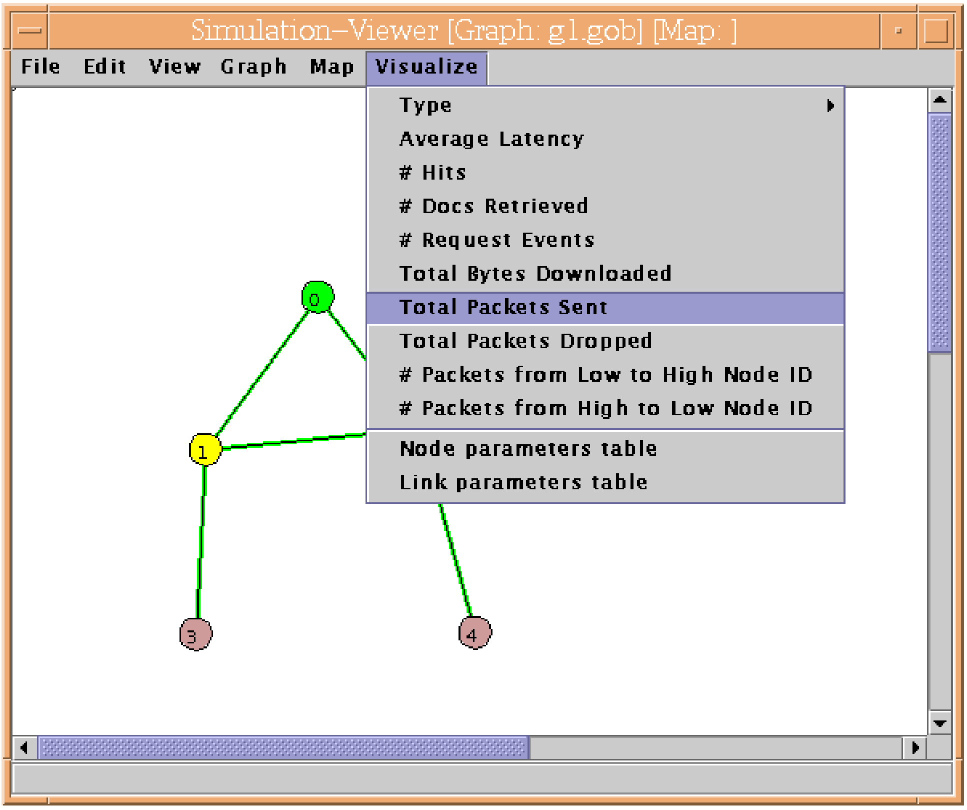

Figure 5.5: Simulation-Viewer

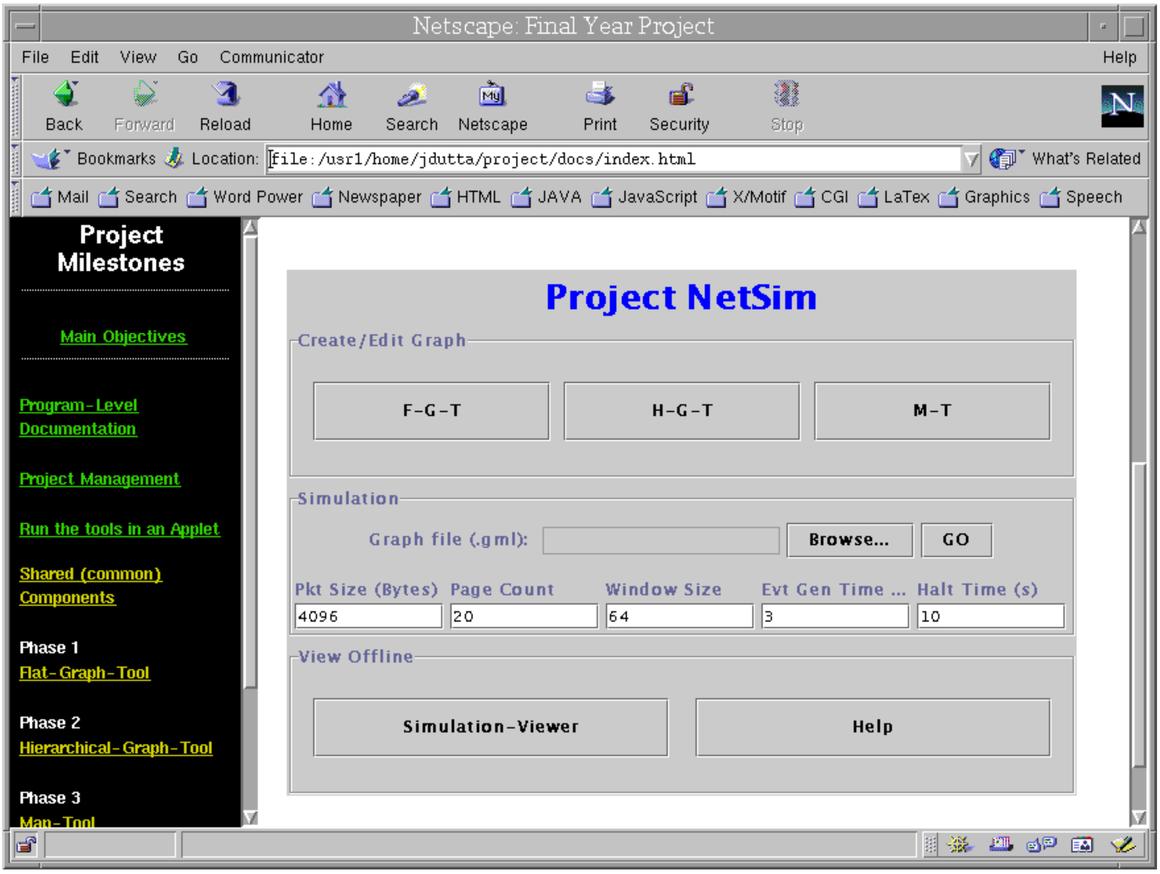

Figure 5.6: The main GUI in an applet

Chapter 6

Conclusion

6.1 Achievements of the project

Initially we started out with the motto to develop a robust network visualizer software. But as we progressed we could realize the overhead of using the software in the traditional way, e.g, installing, configuring and running in individual machines. We then started to form a basis of a general framework of a web based software service, which makes the user feel like using the actual software but not having to install it except a small plugin to display the interface in the browser. At the end, we have fully functional working model of a powerful web based visualizer software which communicates with a server to exchange files and information for the purpose of remote simulation of a network and interpreting its results.

6.2 Scalability and future scope

There is a lot of scalability issues for the visualizer software. It is developed in a highly modular fashion so addition of new features will become quick and easy. At present our software can only deal with offline simulation data got from the simulator in the form of files. Major scalability issues for the system include:

Online visualization

By choosing a suitable framework to communicate with the simulator we can extend the visualization capabilities to online interpretation of data from simulator i.e, visualizing dynamically changing performance parameters over time. We borrow the idea from a network simulator project *ns-2* [3] where a post-simulation file is used to show an animation of network simulation over the time elapsed in the simulation. We can follow this approach by dumping time information of the simulation to the dump file and then interpret it after the simulation. But a more realistic approach would be to create a data pipe with the simulator and get data in real time.

Collection of real network data

We can collect HTTP request data by actual probing of a network instead of statistically generating the request events. This can be done by setting up a proxy server program for a given network. All the machines in that network are configured to send their HTTP requests to the Internet through this small proxy server program. We can then collect the actual pattern of requests for a given network.

3D visualization

After all the scalability issues, the concept of 3D visualization comes as the most interesting one. Here we add a compulsory dimension to a network, that is its physical location. We can compare the parameters of nodes by the z-values above the x-y plane which represents the geography on which the network nodes are situated.

Sharing all types of data in XML format

We are storing a network in GML format for the simulator, custom format for special purpose, and object format for easy loading and storing hierarchy. The simulation results are in a GML-like format. We can make a standard by converging all the formats into the one and most powerful eXtended Markup Language. This way our interface can interact with simulators built in any other platform or language. Here we are forced to export topologies in GML format because the simulator being built upon LEDA supports the format. XML parsers are freely downloadable from the Internet so this is quite an important issue on platform independence and scalability.

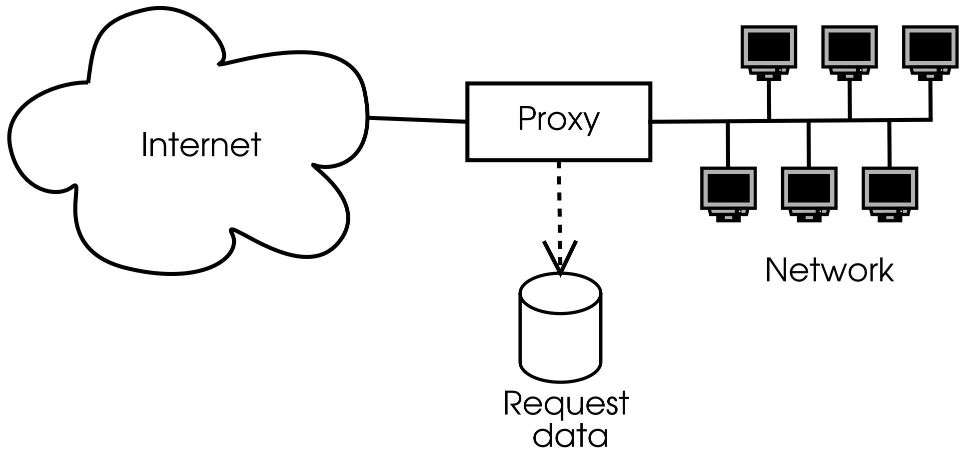

Figure 6.1: Network probe model for collection of real network data

Chapter 7

Appendix

7.1 Installation

The installation of the whole system is quite a big issue as lot of dependencies are there. We discuss them in two parts:

7.1.1 The visualizer software

The visualizer is basically a Java application program. At the top level there is a central interface which launches other applications. If it has to be run as an application then Java version at least 1.2 is required to be present in the system. Otherwise if it has to be accessed as an applet with full functionality there are two issues to be addressed - the client side requirements and the server side requirements.

For publishing the applet in a page at the server side, a page with a simple `<applet>...</applet>` tag is created first. Then using a `htmlconv` utility that simple applet tag is converted to a complex object tag that is compatible with both Netscape in Linux and IExplore in Windows. The html converter utility is freely available at `http://java.sun.com/products/plugin/1.3/docs/htmlconv.html`. On the other hand the client who wishes to access the interface through the applet enabled web page, must have relevant plugins installed for his browser. For Netscape browser the relevant informations can be obtained from `http://java.sun.com/products/plugin/`.

7.1.2 The simulator

The simulator is written in GNU C++ and uses the LEDA library. Therefore it requires that the LEDA library must be installed on the system. LEDA is freely downloadable from `http://www.mpi-sb.mpg.de/LEDA/leda.html`. Regarding memory requirements, a 128MB RAM Linux machine is ok for small simulations, but we should recommend at least 1GB for large and realistic ones for better results.

7.2 Granting access permissions to applets using `policytool`

There is a `policytool` utility with every java installation. This is used to create a file named `.java.policy` under the home directory. Give at least the File Permission. A sample `.java.policy` file is given below which grants all permission to any applet.

```
/* AUTOMATICALLY GENERATED ON Mon Apr 02 00:45:54 GMT+05:30 2001*/
/* DO NOT EDIT */
grant {
  permission java.security.AllPermission;
};
```

In normal cases all permission should not be given. Only file permission will enable the applet to write network files in the client's filesystem. Other permissions should be used with caution as more permissions increase the risk of harm by any malicious applet situated in a web page that a user unknowingly browses.

Bibliography

[1] http://moat.nlanr.net

[2] http://www.ntop.org

[3] http://www.isi.edu/nsnam/ns