MADPastry: A DHT Substrate for Practicably Sized MANETs

Thomas Zahn Jochen Schiller
Institute of Computer Science
Freie Universität Berlin, Germany
{zahn, schiller}@inf.fu-berlin.de

Abstract–**As mobile ad hoc networks (MANETs) become ever more popular, it also becomes more and more interesting to build distributed network applications that one is accustomed to from the Internet on top of MANETs. In the Internet, Distributed Hash Tables (DHTs) have recently proven themselves an efficient building block for such distributed applications. However, DHTs are ill-suited for direct deployment in MANETs as they are largely oblivious of the physical routing.**

Therefore, we propose MADPastry, a DHT substrate explicitly designed for the use in MANETs. MADPastry considers physical locality and integrates the functionality of a DHT and an ad hoc routing protocol at the network layer to provide an efficient indirect routing primitive in MANETs.

To answer the fundamental question whether the extra overhead of maintaining a DHT in MANETs is really worth the effort or, instead, one would be better off broadcasting the actual lookups in the first place, we compare MADPastry's performance against an unstructured Gnutella-style broadcast agent and a DHT substrate without locality awareness.

I. INTRODUCTION

When building distributed network applications, it is crucial to have an efficient mechanism by which to locate a node currently responsible for a certain object or service since there exists no authoritative central server that could map objects to specifics nodes. For that purpose, Distributed Hash Tables (DHTs) [12, 15, 18, 21] have recently been proposed. At the core of each DHT lies the ability to route a data packet to a node currently responsible for a certain key – usually some hash identifier – within a bounded number of hops. This routing process is also referred to as *Indirect Routing*. Unlike regular network-level routing, where a packet is routed from a source to a specific destination node, indirect routing delivers a packet from the source to a previously unknown destination solely based on the given key. Aside from efficiently locating nodes currently responsible for given keys, DHTs are also self-organizing and fault-tolerant and, thus, provide an elegant, scalable, and robust means for building distributed network applications. Lately, DHTs have been successfully used for building distributed data storage systems [3, 14], distributed email systems [8], and distributed event notification systems [16] to name but a few.

It is important to bear in mind, though, that DHTs have been designed for the Internet. They are application-level overlay networks that run on top of and are largely oblivious of the underlying physical network. Therefore, one overlay routing hop usually consists of multiple physical routing hops. Since reliable network routing is practically taken for granted in the Internet today, DHTs do not primarily concern themselves with the physical aspects of an overlay routing step and rather focus on the optimization of the application-level routing.

On the other hand, another field that has attracted great amounts of research are mobile ad hoc networks (MANETs). MANETs consist of wireless mobile devices that dynamically form a network between them. With wireless mobile devices being ever more present and powerful, it is becoming more and more interesting to build the distributed network applications that one is accustomed to from the Internet on top of MANETs. However, while numerous direct (i.e. from a source to a specific given target node) ad hoc routing protocols have been proposed over the last years, be they reactive [6, 10] or proactive [2, 9] approaches, *indirect routing* remains largely unsolved in MANETs.

It is clear to see that MANETs and P2P networks share a good number of key characteristics, such as the lack of a central infrastructure, a highly dynamic network topology, and the need for self-organization. Hence, when designing distributed network applications for MANETs, it would be intuitive to consider the building blocks that have proven themselves appropriate in P2P systems. However, conventional DHTs are ill-suited for a simple deployment on top of MANETs for the following three reasons:

1. First of all, it is important to realize that overlay traffic as such does not exist physically. What does exist, though, is the physical traffic *incurred* by the overlay network. Furthermore, as previously mentioned, DHTs were designed as application-level overlay networks for the (wired) Internet. By abstracting away the underlying physical network, standard DHTs generally do not consider the physical topology in the construction of their overlay topology. In other words, by no means do two overlay neighbor nodes also have to be physical neighbors. This usually leads to the situation that overlay hops can incur unnecessarily long physical routes. Fig. 1 shows an example where four overlay hops actually traverse the physical network twice. Although a number of approaches have been proposed recently [13, 19, 20] to alleviate this problem, standard DHTs are not primarily concerned with physical locality. While this might be tolerable on the wired Internet with its high bandwidth, it is obviously not feasible for MANETs. Here, the delivery probability of a packet quickly decreases with each physical hop due to factors such as low bandwidth, low computation power (of a node), packet

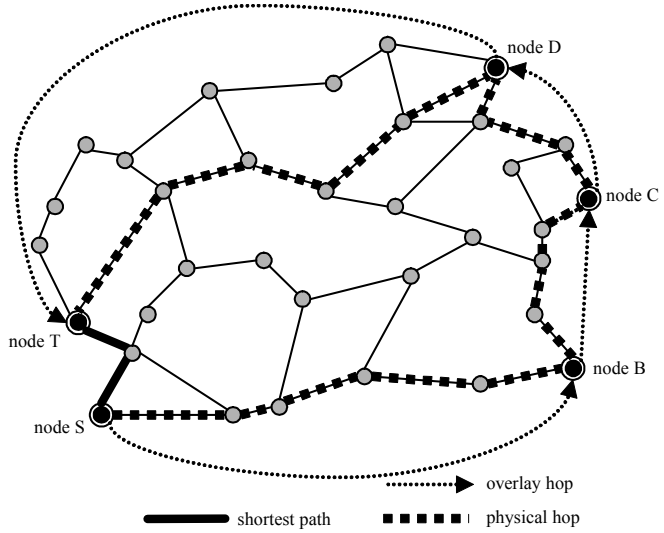

Fig. 1. Overlay vs. physical routing

collisions, or transmission errors.

2. As nodes move around incessantly, routes in MANETs are usually quite volatile and break quickly. For this reason, ad hoc routing protocols have to (re-) establish routes frequently. Due to the lack of a central infrastructure, the majority of ad hoc routing protocols, both proactive [2, 9] and reactive [6, 10], have to at one point or another resort to flooding the network – or regions thereof. This, of course, renders the overlay routing superfluous. There is no point in maintaining an application-level DHT when the physical route to carry out an overlay hop has to be (frequently re-) established through broadcasting. In that case, one would have been better off broadcasting the key lookup itself in the first place. In fact, it is easy to imagine a situation where a key lookup requires two overlay hops, both of which have to have their physical routes discovered through broadcasting. In that case, the key lookup would cause the network to be flooded twice, which is clearly suboptimal.

3. In order to guarantee routing convergence and consistency, DHTs have to periodically maintain their routing tables. Depending on the size and structure of a DHT's routing table and the lookup traffic pattern, the maintenance traffic can constitute a significant portion of the overall traffic. Given the limited bandwidth in MANETs, conventional DHT maintenance can be prohibitively heavy-weight and overwhelm the network.

In this paper, we present MADPastry (Mobile Ad Hoc Pastry), a DHT substrate explicitly designed for the use in MANETs. MADPastry combines Pastry [15] and AODV [10] at the network routing level to provide an indirect routing primitive for MANETs. Our experimental results show that MADPastry achieves better packet delivery ratios at significantly lower overhead than a reference broadcast system. Thus, it is ideally suited as a building block for distributed network applications in practicably sized MANETs.

The remainder of this paper is organized as follows. Section II presents MADPastry's system design in detail. In Section III, we evaluate and analyze MADPastry's performance with experimental results. Section IV discusses related work. Finally, Section V concludes this paper and provides a brief outlook on our future work.

II. MADPASTRY – SYSTEM DESIGN

Due to node mobility and the lack of a central infrastructure, conventional routing protocols in MANETs have to resort to flooding packets during their route discovery process at one time or another. However, these route discovery / maintenance broadcasts create an immense overhead and, thus, constitute a key scalability bottleneck. For this reason, MADPastry was explicitly designed to avoid broadcasts whenever and wherever possible. MADPastry integrates the reactive ad hoc routing protocol AODV [10] and the application layer DHT Pastry [15] to provide light-weight and scalable *indirect routing* functionality at the network layer.

A. Clusters

In standard DHTs, two overlay neighbors can be located arbitrarily far from each other in terms of the underlying physical network. As discussed previously, this can lead to a large overlay stretch (i.e. the ratio between the physical route length traveled during an overlay key lookup compared to the direct physical path from the source to the eventual target node) as subsequent overlay hops can literally crisscross the physical network. Therefore, we believe that it is essential for any DHT substrate in the context of a MANET to consider physical locality ([7]).

MADPastry utilizes the concept of *Random Landmarking* [20] to create physical clusters where nodes share a common overlay id prefix. Thus, two nodes that are physically close to each other are also likely to be "close" to each other in the overlay. Since there are generally no stationary nodes available in MANETs, MADPastry works without any fixed landmark nodes. Instead, it uses a set of *landmark keys*. A landmark key is simply an overlay id. Rather than having dedicated landmark nodes, in MADPastry those nodes become temporary landmark nodes that are currently responsible for one of the landmark keys (i.e. whose own overlay identifiers are currently closest to one of the landmark keys). Therefore, when one of the current landmark nodes fails or resigns, another node (that whose overlay id is *now* closest to the landmark key) will automatically assume its role.

Landmark keys should be chosen so that they divide the overlay id space into equal-sized segments. For example, in a hexadecimal-based id space, an appropriate set of landmark keys could be: 0800…000, 1800…000, 2800…000, . . . , E800…000, F800…000.

To form clusters of common overlay id prefixes, nodes associate themselves with the temporary landmark node that is currently closest to them (e.g. as determined by the hop count) by adopting its overlay id prefix. For that purpose, temporary landmark nodes send out beacons periodically. These beacons are broadcast and whenever a node overhears a landmark beacon, it stores the current landmark node's id and the distance to it as given by the hop count of the beacon. Nodes periodically examine their landmark list to determine whether they have moved closer to a new landmark, i.e. whether they

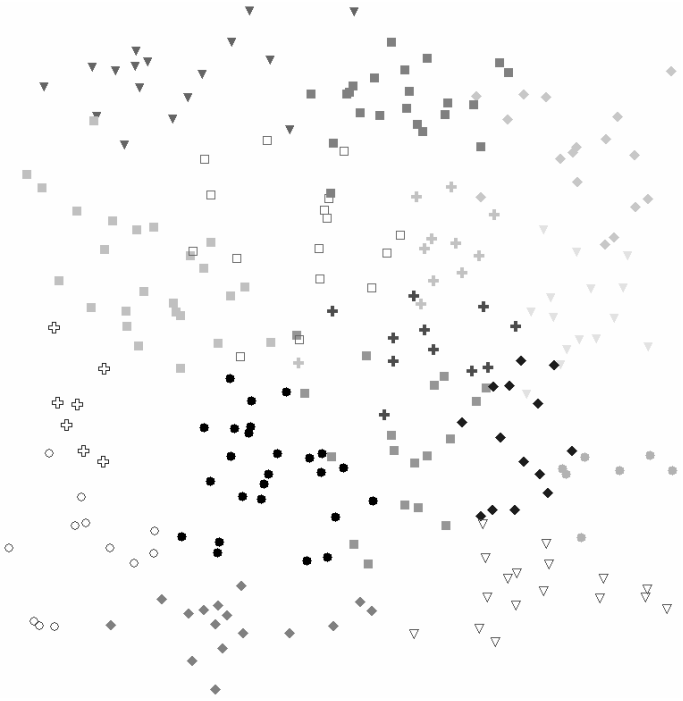

Fig. 2. Spatial distribution of id prefixes.

have moved – with high probability - into a new overlay cluster. If so, a node will assign itself a new random overlay id with its new cluster's overlay id prefix, resign from the overlay network with its old id, and rejoin the overlay network with its new id.

Since broadcast messages impose a serious burden on wireless networks, temporary landmark nodes do not broadcast their beacons throughout the entire network. Instead, landmark beacons are only propagated within the landmark's own cluster, i.e. beacons are only forwarded by nodes belonging to that cluster. Nodes outside the landmark's cluster will store the beacon information and then drop the packet. The reasoning behind this is that nodes will not be interested in beacons originating halfway across the network since they would not – and in fact should not – join that cluster anyway. In fact, a beacon is only of value to its own cluster members and to nodes bordering the cluster (note that bordering nodes will receive the beacon but not forward it) as those are the regions where cluster crossovers (should) occur.

MADPastry's Random Landmarking has the following effects. First of all, it leads to physically close nodes forming overlay regions, or clusters, with common id prefixes. This is demonstrated by Fig. 2 which shows the spatial distribution of overlay id prefixes in a 250 node MADPastry network. Equal symbols of equal colors represent equal overlay id prefixes. Furthermore, since the last overlay routing step in DHT systems is the numerically closest, with MADPastry the last overlay routing step also tends to be physically close, whereas with Pastry the opposite is often the case [1, 15].

B. Routing Tables

MADPastry maintains three different routing tables: a standard AODV routing table for physical routes from a node to specific target nodes, as well as a stripped down Pastry routing table and a standard leaf set for indirect routing.

Pastry Routing Table. The standard Pastry routing table consists of $\lceil \log_{2^b} N \rceil$ rows with $(2^b\text{-}1)$ entries each. The conventional Pastry protocol stipulates that each node would periodically choose one random entry from each routing table row for maintenance. It would then contact each selected node and receive its corresponding routing table row. Then, it would ping the entry candidate pair (i.e. the local entry and the remote entry) to determine the most appropriate entry (e.g. the closer one, the one with the lower latency, etc.) for each slot in the row. Obviously, the traffic induced by this maintenance process constitutes a large portion of the overall traffic and can easily overwhelm a wireless network.

To avoid this prohibitive maintenance overhead, a MADPastry node only stores a degenerate Pastry routing table. A MADPastry routing table only needs to contain $\lceil \log_{2^b} K \rceil$ rows, with K being the number of landmark keys. In other words, it only needs to have as many rows as are necessary to fit a "pointer" entry into each overlay cluster. For example, with b=4 (hexadecimal overlay identifiers) and K=16, a MADPastry node only needs to have the first row of a standard Pastry routing table. Each slot would then contain an arbitrary reference node in the corresponding overlay cluster.

At this point it is important to realize that with these strapped down routing tables we are deliberately sacrificing the O(log N) bound on the number of overlay hops during a key lookup for the sake of a drastically reduced maintenance overhead. In standard Pastry that bound stems from the idea that in each overlay routing step the current (intermediate) node determines the matching prefix length between the key and its own overlay id. It would then consult the corresponding row in its routing table to find the next hop that would, ideally, increase the prefix match by one ([15]). Clearly, this process is interrupted in MADPastry after the first (few) overlay hop(s).

However, we believe that the benefits of abandoning complete Pastry routing tables far outweigh its penalties in practicably sized MANETs. First of all, we consider network sizes in the order of up to 1,000 nodes far more realistic than 100,000 nodes in "pure" MANETs ([5]) without any wired infrastructural gateway nodes (in such wireless-cum-wired topologies, e.g., one could again have the wired gateway nodes maintain complete Pastry routing tables). Let's consider a large MANET of 1,000 nodes and let us assume 16 landmark keys and the Pastry id base b=4 (hexadecimal overlay identifiers). In that case, a MADPastry cluster would consist of slightly more than 60 nodes on average. Here, the first overlay hop would be decided by the first (and only) routing table row and would deliver a request to its target cluster. Once there, leaf set based intra-cluster routing would deliver the request to its eventual target node (see B for details on MADPastry's routing). Given a standard leaf set size L=16, intra-cluster routing would require about 8 hops in the worst case (62.5 / L/2). However, since nodes in a MADPastry cluster are very likely to be physically close to each other, there is a high chance that a) the eventual target node will overhear the request sooner, or b) the

current node has overheard a packet from the eventual target in the past and thus knows about it (and a route to it) already. Therefore, intra-cluster routing can be expected to be performed efficiently with only a few overlay hops.

Pastry Leaf Set. The standard Pastry leaf set contains L entries: the $L/2$ numerically closest (in terms of their overlay id) smaller nodes and the $L/2$ numerically closest larger nodes. Of course, the leaf set also needs to be maintained. For that purpose, a Pastry node periodically pings its leafs to determine whether a are still alive. The leafs respond with their respective leaf sets so the source node could learn about new close members of the overlay that it did not known about yet.

Again for the sake of a reduced traffic overhead, we sacrifice the 100% accuracy of the leaf sets. It is important here to bear in mind that for a correct routing process it is actually not necessary that nodes always have 100% accurate leaf sets. To guarantee routing convergence, it is only essential for a node to always know its correct "left" and "right" overlay neighbor, i.e. the node that has the numerically closest smaller overlay id and the node that has the numerically closest larger overlay id. Otherwise, the routing process might not always end up at the right node. Therefore, a MADPastry node proactively only pings its "left" leaf and its "right" leaf who will respond with the id of the node that they think is the originator's left or right leaf (i.e. ideally themselves). Furthermore, each node periodically sends out a beacon with its current overlay id that is propagated throughout its cluster. Since nodes in a MADPastry cluster share a common overlay id prefix, the majority of a node's leafs will likely be from its own cluster. Thus, given MADPastry's leaf set maintenance scheme, one can expect the leaf set of a MADPastry node to have the correct "left" and "right" leaf and to include a close approximation of the accurate $L/2$ entries in each half.

AODV Routing Table. To carry out a concrete overlay hop, a MADPastry node also maintains a standard AODV routing table. It includes for specific physical destinations the next (physical) hop address as well as for each such route a sequence number.

C. Routing

MADPastry provides an *indirect routing* primitive in MANETs. I. e., MADPastry routes packets based on an overlay id but the final (physical) target node is usually unknown. It does so by integrating overlay and physical routing. Therefore, when a MADPastry node receives a request packet, it can principally be due to the following two situations:

1) The node could be the target (i.e. the physical destination) of an *overlay* hop. In this case, the node needs to determine the next overlay hop. For this purpose, it will consult its Pastry routing table to find a node that would increase the matching key prefix by one or its leaf set to find a node that is numerically closer to the key than the current node is. This corresponds to standard Pastry routing.

2) The node could be an intermediate node on the physical path of an overlay hop that is being carried out. Now, the node would behave like a regular AODV node. It would consult its AODV routing table to determine the next physical hop on the route toward the destination of this overlay hop and then forward the packet on.

To minimize the routing traffic, any such intermediate node on the physical path of an overlay hop inspects the destination of the overlay hop. If the intermediate node's own overlay id already happens to be numerically closer to the packet's key than that of the overlay hop's actual destination, it will "intercept" the packet. In other words, it will consider the current overlay hop completed and select from its Pastry routing table or leaf set the next overlay hop.

An interesting question arises when the physical route to carry out an overlay hop is unknown. Again, this can happen in two situations:

1) A node selects the next overlay destination from its Pastry routing table or leaf set, but there is no (valid) route information in its AODV routing table for that destination.

2) An intermediate node on the physical path of a current overlay hop might not have a (valid) next hop entry in its AODV routing table to forward the packet.

To avoid network-wide broadcasts whenever possible, MADPastry tries to leverage its cluster locality in such cases. If the node that has no (valid) information on how to continue the path of an overlay hop is already in the target cluster (i.e. shares a common prefix with the packet's destination), it will not issue an AODV-style route discovery for the destination. Instead, it will broadcast the overlay packet itself within the confines of its cluster. Due to the physical locality in MADPastry clusters, that broadcast is very likely to stay in a limited region of the network. Otherwise, if the node is not in the target cluster, it will queue the packet and start a regular AODV expanding ring broadcast to discover a route to the packet's destination.

At this point, it is worth mentioning that MADPastry provides indirect routing in MANETs. However, it is not a stand-alone network application as such. That means that it is up to the actual application running on top of MADPastry to determine the action a node should take when it receives a packet. MADPastry merely delivers a packet to the node currently responsible for the packet's key.

D. Routing Table Maintenance

As described in Section II.B, the only proactive routing table maintenance that a MADPastry node performs is the periodic pinging of its "left" and "right" leaf. This is necessary to guarantee overlay routing convergence.

All other routing entries are gained by overhearing data packets. For that reason, a MADPastry packet always contains the following information:

- the AODV sequence number of the packet's source (i.e. the destination node of the previous overlay hop)
- the AODV sequence number of the packet's previous physical hop (i.e. the immediate predecessor on the current physical path)
- the overlay id of the packet's
- the overlay id of the packet's previous physical hop

Whenever a MADPastry node now receives or overhears a packet, it extracts the AODV sequence numbers to update its

AODV routing table to contain a fresh route to the packet's source and, trivially, to the previous physical hop. MADPastry uses the heuristic that existing routes to those two nodes are overwritten in the favor of the fresh route. Analogously, it exploits their overlay identifiers included in the packet to insert the nodes into the corresponding slots in the Pastry routing table and leaf set. Again, any existing entries are overwritten in the favor of fresh physical routes.

It is clear to see that the fill degree and accuracy of the Pastry routing tables and leaf sets largely depend on the number of packets that a MADPastry node receives or overhears. When network traffic is low and nodes receive only few packets, their routing tables and leaf sets might be scarcely filled so that the routing performance is likely to suffer. We believe, however, that when there are relatively few lookups – i.e. the network traffic is low – there really is no point in maintaining much routing structure in the first place. One would be better off broadcasting the occasional lookups instead. As the lookup frequency increases, so does the network traffic and thus the fill degree and accuracy of the MADPastry routing tables and leaf sets. Therefore, MADPastry is especially geared toward MANETs with high lookup rates – as otherwise we believe DHT substrates are of little practical use to begin with. Our experimental results support these assumptions.

III. Experimental results

To evaluate the performance of MADPastry, we implemented MADPastry as a routing agent in ns2. All simulations the we carried out modeled wireless networks over the course of one (simulated) hour. Nodes are always moving around according to the random way point model with 0s pause time and at a steady speed. For data transmission, nodes are using the 802.11 communication standard with a transmission range of 250m.

The following metrics are analyzed:

Success Rate – the percentage of random lookups that are eventually delivered to the correct responsible node.

Packet Overhead – the number of packets that all routing agents forward during a simulation. This count is increased whenever a node forwards a packet to the next physical hop. In the case of MADPastry, this figure comprises *all* router-level packets that are created by a MADPastry node: lookups, leaf pings/pongs, join requests, join replies, leave messages and node beacons. In the case of the Gnutella-style broadcast router, this figure only consists of lookups as there simply are no maintenance messages.

Overall Traffic – the total network traffic in Kbytes that is created during the simulated hour. Whenever a node forwards a packet, this figure is increased by the packet size. Again, this figure includes *all* router-level packet types for MADPastry. Here, it is important to mention that MADPastry packets on average are about 4 times larger (excluding the IP header) than the corresponding broadcast agent's packets as they carry additional information such as the last hop's overlay id and so forth.

A. Reference Applications

The ns2 routing agent implements the MADPastry protocol as described in section II. Nodes send out cluster beacons every 30s and ping their left and right leafs every 60s. 16 landmark keys are used in the simulations. Additionally, to further increase the success rate, a lookup initiator always also issues a secondary lookup. That backup lookup is sent to the second best candidate for the first overlay hop. If both lookups arrive at the eventual target, the second one is dropped.

As discussed before, the fundamental question to be answered when deploying a DHT substrate in MANETs is whether the extra overhead of maintaining the DHT structure is really worth the effort. Or, is the benefit gained from using a DHT so miniscule that we would have, indeed, been better off just broadcasting the lookups in the first place. Therefore, as a reference application to compare MADPastry's results against, we implemented a Gnutella-style broadcast routing agent. The broadcast agent maintains no overhead structure and, thus, has no extra maintenance overhead. It broadcasts a packet to all its one-hop neighbors who, then, forward the packet to all their one-hop neighbors and so forth. Nodes keep track of the packet sequence number so that already forwarded packets will not be sent a second time.

To verify whether MADPastry's extra overhead stemming from cluster roaming (leaving, rejoining, coping with invalid overlay identifiers, etc.) is justified, we also implemented a routing agent that integrates regular Pastry and AODV as a second reference application. It works very similar to MADPastry except that it does not employ *Random Landmarking*. Thus, there are no physical clusters of nodes sharing a common overlay id prefix and no overlay id reassignment – i.e. leaving and rejoining the network – either. Since Pastry's standard routing table and leaf set maintenance is prohibitive in MANETs, the integrated Pastry routing agent, too, fills its routing table by forwarding and overhearing live packets and also only pings its left and right leaf proactively. Furthermore, beacons as well as lookups for which no physical route is known are broadcast throughout the entire network – as there are no clusters. Also, the integrated Pastry routing agent does not issue any secondary lookups (as the MADPastry routing agent does) since its overhead is already drastically higher than MADPastry's – as the simulation results will show.

B. Basic Results

We compare the performances of MADPastry, the Pastry routing agent without clusters and the Gnutella-style broadcast agent in networks of 100 and 250 nodes. In all simulations, we use square planes with a node density of 100 nodes/km². In a first set of simulations, nodes are moving around at a regular speed of 1.4 m/s, which corresponds to a fast walking speed. As a traffic source, we implemented an application sitting on top of either MADPastry, the Pastry router without clusters or the broadcast agent that issues lookups for random keys periodically. For this first set of simulations, each node sends out a random key lookup every 10s.

Fig. 3 shows the success rate of the three routing agents for the random lookups. As can be seen, MADPastry achieves

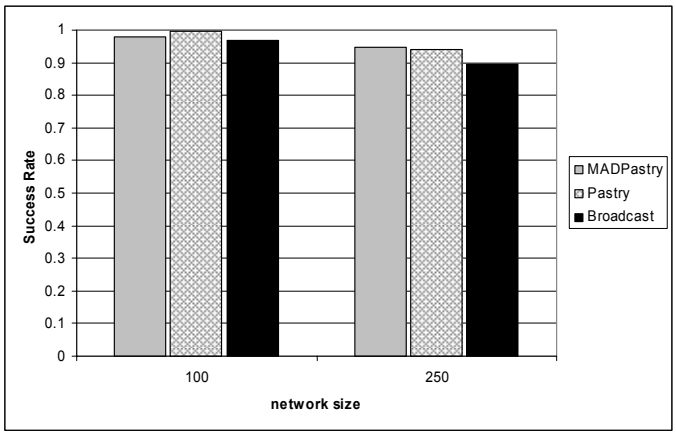

Fig. 3. Lookup success rate - 1.4m/s.

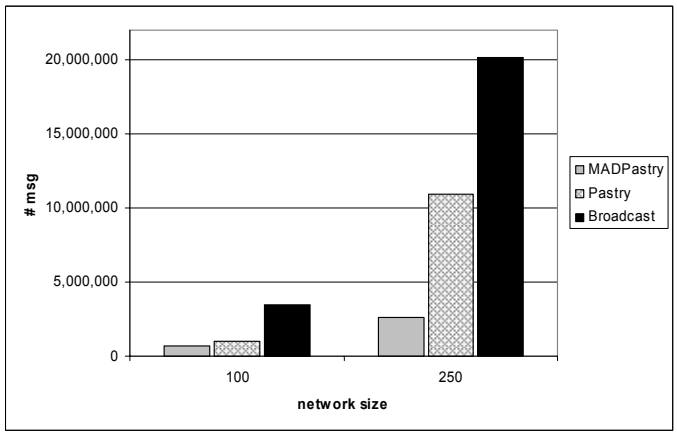

Fig. 4. Total number of messages - 1.4m/s.

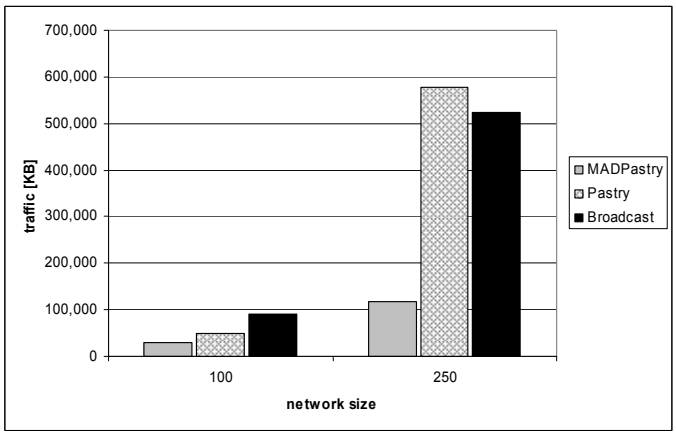

Fig. 5. Overall traffic in Kbytes - 1.4m/s.

better success rates in both 100 and 250 node networks compared to the broadcast agent. Furthermore, MADPastry retains success rates of well above 90% for both network sizes, whereas the broadcast agent's rate drops below 90% in a 250 node network. The success rate of the Pastry router without clusters is practically the same as MADPastry's (approx. 1-2% higher).

Fig. 4 shows the number of messages that the routing agents send and forward during the simulated hour in order to achieve their respective success rates. It becomes clear that MADPastry produces drastically less network traffic than the Gnutella-style broadcast agent does. In a 100 node network, the broadcast router needs about 5 times and in a 250 node network even about 7 times the number of messages that MADPastry needs. The Pastry router without clusters incurs roughly 1/3 of the message traffic of the broadcast agent in a 100 node network and roughly 1/2 in a 250 node network, which is well above MADPastry's message traffic.

However, it is important to bear in mind that MADPastry / Pastry packet headers are longer than those of the broadcast router due to the extra information included in them (see above). To make sure we are not comparing apples and oranges, Fig. 5 shows the traffic in forwarded Kbytes instead. Even with that metric, MADPastry still produces several times less traffic than the broadcast router does. An interesting observation can be made here for the Pastry router without clusters. While still below the broadcast agent's overhead in a 100 node network, its overall traffic becomes larger than the broadcaster's in a 250 node network. This can easily be explained by the fact that Pastry's overlay routing usually requires several overlay hops per lookup. Since there are no clusters, successive overlay hops can crisscross the physical network. Thus, when the Pastry router has to resort to broadcasting a lookup (because the physical route to carry out the next overlay hop is unknown), the lookup could already have crossed the network several times. Obviously, one would have been better off if one had broadcast the lookup right away – which is exactly what the broadcast agent does. Furthermore, even if the lookup could be delivered without being broadcast (i.e. the routes for all overlay hops involved were known), the accumulated physical path lengths of the overlay hops might only be slightly more light-weight than a broadcast. Additionally, the required periodic beacon broadcasts are added on top. Since both physical and overlay paths are much shorter in a 100 node network, this effect is less pronounced there.

This clearly demonstrates how important it is for a DHT substrate in MANETs to consider physical locality – as MADPastry does.

C. Node Velocity

So far, we have only considered networks with node velocity of 1.4 m/s. In the next set of simulations, we will examine 250 node networks with various node velocities: 0.1 m/s, 0.6 m/s (slow walking speed), 1.4m/s (fast walking speed), 2.5 m/s and 5.0 m/s. We continue to use a request frequency of one random lookup every 10s per node.

Fig. 6 shows the success rates of the three routing agents in reference to the different node velocities. One can see that both MADPastry and Pastry without clusters achieve better success rates than the Gnutella-style router does for speeds up to a fast walking speed (1.4 m/s). At a speed of 2.5 m/s, the success rates of MADPastry and Pastry without clusters start falling below the broadcast router's. With fast speeds, routes break so frequently that Pastry without clusters can no longer keep its

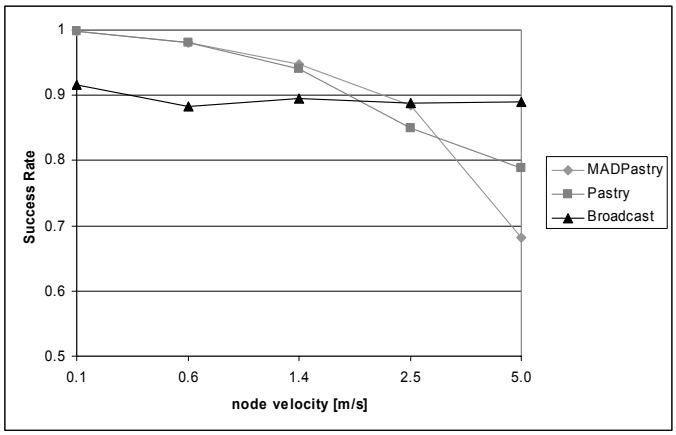

Fig. 6. Success rates vs. node velocity.

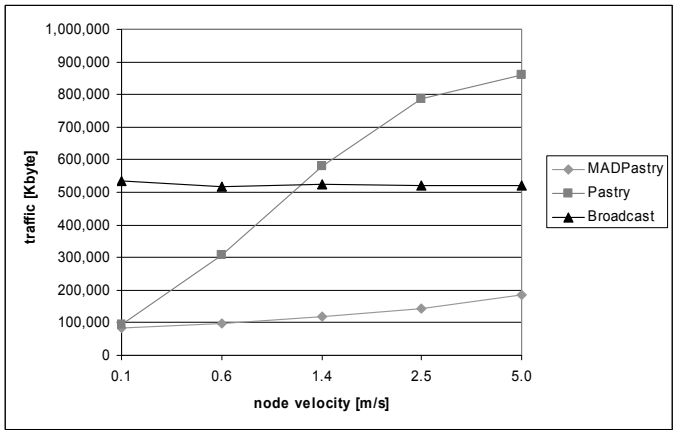

Fig. 7. Overall traffic vs. node velocity.

routing table and leaf set sufficiently valid – hence its success rate drops below the broadcast agent's success rate. With MADPastry this problem is further aggravated by the fact that nodes move from cluster to cluster so rapidly that a) they spend a significant amount of their time leaving and rejoining the network, and thus b) their overlay routing tables frequently contain stale entries.

Fig. 7 shows the overall traffic produced by the routing agents during an average simulation run. As can be expected, the overhead of the broadcast agent is practically independent of the node velocity. Since broadcasts in MADPastry are restricted to their respective cluster, MADPastry's overall traffic stays significantly below that of the other two routing agents. For Pastry without clusters, the overall traffic quickly surpasses even that of the broadcast agent as route failures occur more and more frequently and the effects described in Section III.B become ever more pronounced.

D. Request Rates

The scenarios considered thus far all assumed a node lookup rate of one lookup per 10s. Next, we take a look at the impact of the lookup rate on the overall performance. We will evaluate lookup intervals of 1s, 10s, and 60s in a 250 node network with a node velocity of 1.4 m/s.

Fig. 8 shows the success rates of the three routing agents in reference to the lookup rate. As already seen in Section III.B, MADPastry and Pastry without clusters achieve comparable success rates well above 90% for a per-node lookup interval of 10s. The Gnutella-style router's success rate here drops below 90%.

A very interesting observation can be made in networks with high lookup rates of 1 lookup per second per node. At such high lookup rates, both the broadcast agent and Pastry without clusters can no longer keep up with MADPastry. Their (frequent) network-wide broadcasts of the lookup requests clearly overwhelm the wireless physical network, resulting in so many packet collisions that the majority of lookups fail to be delivered. Thus, their success rates drop to 20%. On the other hand, MADPastry's physically shorter overlay hops (compared to Pastry without clusters) and its local cluster broadcasts allow it to still maintain a success rate of 92% in the presence of such high lookup rates. Again, MADPastry's overall traffic remains significantly below that of both the broadcaster and Pastry without clusters, as Fig. 9 shows.

If there is only one lookup per minute, MADPastry's lookup rate falls below 90% (87%). This is due to fact that the nodes overhear much less packets to update their routing tables with. Therefore, nodes often do not detect other nodes' cluster changes, which can result in packets being routed to stale overlay addresses. However, we believe that a request rate of

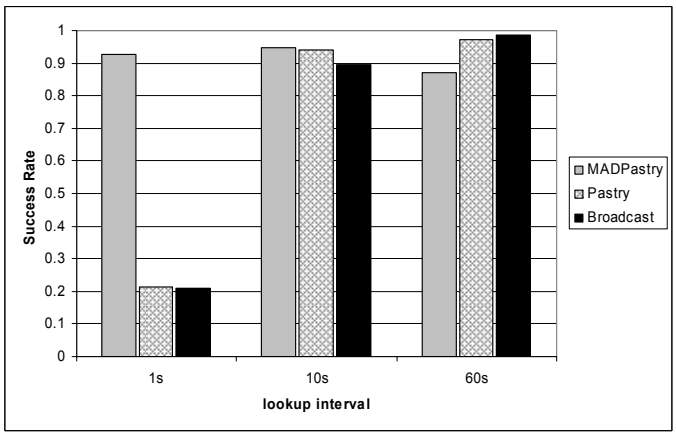

Fig. 8. Success rates vs. lookup intervals.

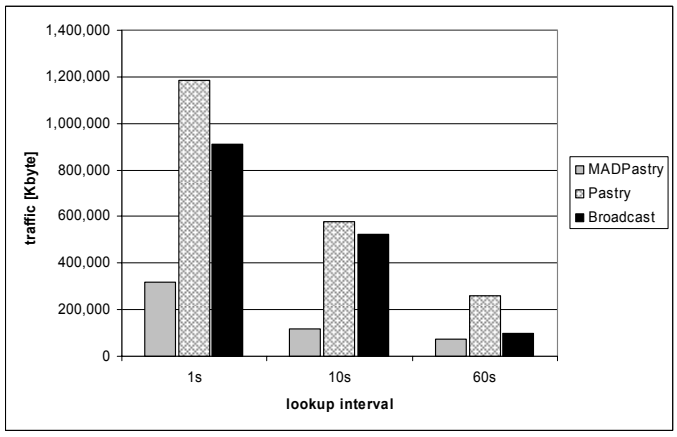

Fig. 9. Overall traffic vs. lookup intervals.

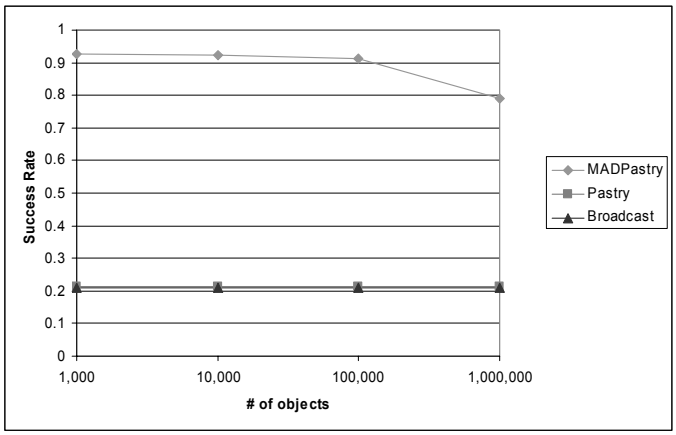

Fig. 10. Success rate vs. # of objects.

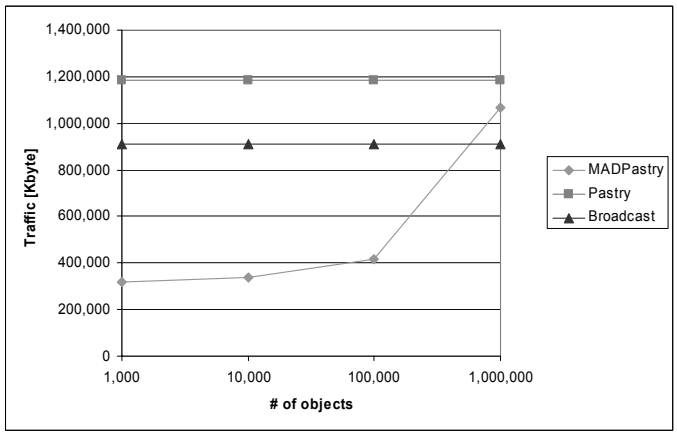

Fig. 11. Overall traffic vs. # of objects.

one lookup per minute is probably too low to justify the effort of maintaining a DHT in the first place.

E. Handovers

Our experimental results have shown that MADPastry produces drastically less overhead than the broadcast agent and Pastry without clusters do. It is important to realize, though, that our experimental results present the *gross* overhead savings of MADPastry. When MADPastry nodes change their cluster membership, they effectively change their overlay id. Therefore, when a MADPastry node changes its overlay id, it would have to pass the objects (or, more likely, references to them) that it was responsible for under its old overlay id to its old left and right leaf before leaving the network and acquire the new objects (or, more likely, references to them) that it is now responsible for from its new left and right leaf. However, the nature of that additional handover traffic clearly depends on the actual application running on top of MADPastry, as well as the amount and distribution of the objects in the network.

To evaluate the principal performance impact of handovers, we next extended the test application from the previous sections. Aside from periodically issuing random lookups, we also distribute objects randomly among the nodes. Since it can be prohibitive to transfer large objects in MANETs, the DHT actually stores references to the objects. A reference contains the object's id (i.e. hash key) and the physical address of the node where the object resides. When a MADPastry node changes its overlay id, it hands over and acquires the corresponding references.

Again, we employ a 250 node network and examine the effect that a total of 1,000, 10,000, 100,000, and 1,000,000 randomly distributed objects have on the overall performance. We consider a lookup rate of one lookup per second. Fig. 10 shows that for up to 100,000 objects, MADPastry can sustain success rates of above 90% as the additional handover packets help spread node information through the network, thereby mitigating the negative effects of an increased number of packet collisions. MADPastry's overall traffic (now also including handover packets) remains well below 50% of the broadcast agent's overhead (see Fig. 11). With 1,000,000 objects in the network, however, the handover packets start massively interfering with lookup packets, as they now dominate the overall traffic, so that the success rate starts falling.

IV. RELATED WORK

To the best of our knowledge, the first approach that proposes the integration of a conventional DHT with an ad hoc routing protocol to provide indirect routing in MANETs is Ekta [11]. Ekta, like MADPastry, is based on Pastry [15], but it uses DSR [6] for its route discoveries. The main difference to MADPastry is that Ekta does not explicitly consider physical proximity in its DHT routing table. Instead, it merely tries to optimize its DHT entries by overhearing packets and replacing physically remote entries by nearer ones. Ekta has no notion of overlay clusters of physically close nodes. Thus, the routes traveled during its overlay routing process may be expected to be less efficient than those in the cluster-based MADPastry. This should become even more pronounced as the network size increases.

In [4], cross-layering is used to combine Pastry with the proactive ad hoc routing protocol OLSR [2]. Again, physical proximity is not explicitly taken into consideration in the DHT so that routes may also be expected to be less efficient than in MADPastry. Furthermore, experimental results are only provided for an 8-node network, which we believe shows only little of the characteristics that we consider (such as multi-hop overlay routes and physical routes).

A different approach to providing indirect routing in MANETs is taken by the Safari Project [17]. The Safari architecture is based on a hierarchical ad-hoc routing protocol and ultimately aims at providing network services such as name resolution, storage, email, instant messaging, etc. in very large-scale mobile networks. Safari's routing is based on hierarchical cells of increasing diameter. Each node is assigned an overlay address based on its position in the hierarchy. Objects are then hashed into the Safari address space to map objects to nodes. Due to its many hierarchy levels, overlay ids change frequently in Safari, which leads to a significant maintenance overhead. We believe that this overhead will outweigh MADPastry's overhead in practically sized MANETs in the order of up to 1,000 nodes that we consider.

V. CONCLUSION

As MANETs become ever more popular, it also becomes interesting to build distributed network applications that one is accustomed to from the Internet on top of MANETs. We have presented MADPastry as an efficient building block for such applications in MANETs. MADPastry provides reliable indirect routing in MANETs by a) considering physical locality in the construction of its DHT and b) by integrating the functionality of a DHT and an ad hoc routing protocol at the network layer. Our simulation results have shown that MADPastry achieves comparable or better lookup success rates at *significantly* less overall traffic compared to a reference broadcast application and a reference DHT substrate without locality awareness for most practicable scenarios considered. We, therefore, conclude that it is essential for any DHT substrate in MANETs to explicitly consider physical locality. Then, it is well worth the effort to maintain a DHT in MANETS with node velocities up to at least fast walking speeds.

As future work, we plan to implement real network applications, such as an event notification system, on top of MADPastry. Furthermore, it would be interesting to investigate how other DHTs such as CAN or Chord would fare in MANETs when integrated with ad hoc routing protocols.

REFERENCES

[1] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. "Exploiting Network Proximity in Peer-to-Peer Overlay Networks". In *Proc. of FuDiCo*, June 2002.

[2] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, A. Qayyum, and L. Viennot. "Optimized Link State Routing Protocol for Ad Hoc Networks". In *Proc. of IEEE INMIC*, December 2001.

[3] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. "Wide-area cooperative storage with CFS". In *Proc. of ACM SOSP*, October 2001.

[4] F. Delmastro. "From Pastry to CrossROAD: CROSS-layer Ring Overlay for AD hoc networks". In *Proc. of PerCom*, March 2005.

[5] P. Gupta and P. R. Kumar. "The Capacity of Wireless Networks". In *IEEE Transactions on Information Theory, Vol. 46, No. 2*, March 2000

[6] D. B. Johnson and D. A. Maltz. "Dynamic Source Routing in Ad Hoc Wireless Networks". *Kluwer Academic*, 1996.

[7] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris. "Capacity of Ad Hoc Wireless Networks". In *Proc. of ACM SIGMOBILE*, July 2001.

[8] A. Mislove. "POST: A Decentralized Platform for Reliable Collaborative Applications". *Master of Science Thesis*, Rice University, December 2004.

[9] C. E. Perkins and P. Bhagwat. "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers". In *Proc. of ACM SIGCOMM* , August 1994.

[10] C. E. Perkins and E. M. Royer. "Ad hoc on-demand distance vector routing". In *Proc. of IEEE WMCSA*, February 1999.

[11] H. Pucha, S. M. Das, and Y. C. Hu. "Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks". In *Proc. of IEEE WMCSA*. December 2004.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. "A Scalable Content-Addressable Network". In *Proc. of ACM SIGCOMM*, August 2001.

[13] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. "Topologically-Aware Overlay Construction and Server Selection". In *Proc. of IEEE Infocom*, June 2002.

[14] A. Rowstron and P. Druschel. "PAST: A large-scale, persistent peer-to-peer storage utility". In *Proc. of HotOS VIII*, May 2001.

[15] A. Rowstron and P. Druschel. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". In *Proc. of Middleware*, November 2001.

[16] A. Rowstron, A-M. Kermarrec, M. Castro and P. Druschel. "SCRIBE: The design of a large-scale event notification infrastructure". In *Proc. of NGC2001*, November 2001

[17] Safari Project. http://safari.rice.edu/

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In *Proc. of ACM SIGCOMM*, August 2001.

[19] M. Waldvogel and R. Rinaldi. "Efficient Topology-Aware Overlay Network". In *Proc. of HotNets-I*, October 2002.

[20] R. Winter, T. Zahn, and J. Schiller. "Random Landmarking in Mobile, Topology-Aware Peer-to-Peer Networks". In *Proc. of FTDCS*, May 2004.

[21] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. "Tapestry: An Infrastructure for Fault-Resilient Wide-area Location and Routing". *UCB Tech. Report UCB/CSD-01-1141*, April 2001.