7 Application II: A DHT-based Unicast for MANETs

This far, MADPastry has always been used to provide indirect, key-based routing in MANETs. Another common task in MANETs is the direct routing of a packet from a source to a given destination node. This conventional, point-to-point communication between a given source node and a given destination node is also referred to as *unicast*. A very large body of work exists on unicast protocols for MANETs. Please see Section 2.2 again for an overview.

However, this chapter will demonstrate that MADPastry, apart from providing indirect routing, can also be used to perform direct routing – i.e. unicasting – in MANETs. In fact, a MADPastry-based unicast can achieve better packet delivery rates and produce lower network traffic than conventional ad hoc routing protocols. This might be especially useful for MANETs that are already running a DHT application in the first place. In this case, nodes would no longer have to maintain a separate ad hoc routing protocol in addition to their DHT, but instead they could let MADPastry handle their point-to-point routing as well.

7.1 Address Servers

Conventional ad hoc routing protocols route a data packet from a source node to a destination node based on the destination's node address. MADPastry, on the other hand, routes data packets based on an overlay key. Therefore, when a node A wants to send a data packet to a specific node B using MADPastry, node A obviously needs to resolve node B's current MADPastry (i.e. overlay) ID.

The basic idea, here, is that a packet whose key is the same as some node Z's overlay ID will eventually be delivered by MADPastry's key-based routing to that node Z as Z is – trivially – responsible for the packet's key (its own overlay ID). Therefore, nodes need to discover the overlay IDs of the respective destination nodes to which they want to send packets.

To provide nodes with such functionality, a simple and straight-forward address resolution scheme will be used. With MADPastry's unicast scheme, each node has exactly one, unique but dynamic address server. Whenever a node assigns itself a new overlay ID (e.g. by joining the network, by moving to a different MADPastry cluster), it will publish its new, current overlay ID its address server. For this purpose, the node hashes its node address (e.g. its MAC or IP address), thereby acquiring an overlay key – its *address server key* (ASK). Next, the node simply routes a packet containing its current overlay ID toward its ASK. The node currently responsible for that ASK stores the originator's current overlay ID and becomes its temporary address server.

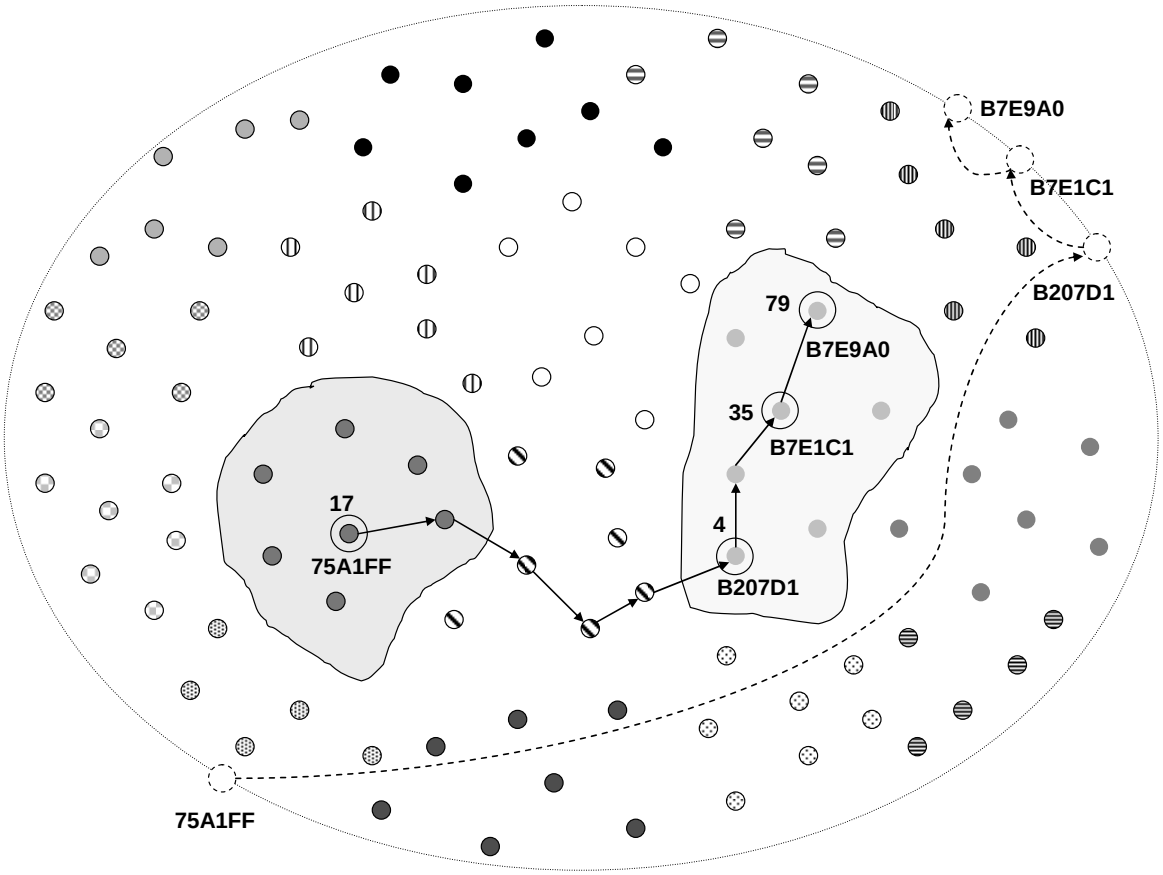

Figure 7.1 Address publication with MADPastry's unicast scheme.

Figure 7.1 shows an example of this process. Node 17 has just assigned itself a new overlay ID `75A1FF`. To publish its new ID, node 17 now hashes its node ID into the overlay ID space to obtain its ASK: h(17)→`B7A9CC`. Using MADPastry, node 17 next sends a packet containing its new overlay ID towards its ASK. With the first overlay hop, MADPastry delivers the packet to node 4 (overlay ID `B207D1`) who, then, forwards the packet to node 35 (overlay ID `B7E1C1`). Finally, the packet arrives at node 79 whose overlay ID `B7E9A0` is closest to node 17's ASK, and, thus, node 79 becomes node 17's address server and stores its current overlay ID.

7.2 MADPastry's Unicast

Address resolution works analogously. When a node A now wants to send a data packet to some node B whose current overlay ID is still unknown to node A, it hashes B's node address to obtain node B's ASK. Using that ASK, node A then sends a request to node B's current address server to acquire node B's current overlay ID. Once node A has thus learned about node B's overlay ID, it can now use that overlay ID to send data packets destined for node B using MADPastry.

Figure 7.2 illustrates MADPastry's address resolution scheme. Adhering to the example from the previous section, node 51 wants to communicate with node 17 but does not know node 17's current overlay ID. Hence, node 51 needs to contact node 17's current address server. For this purpose, it hashes node 17's node

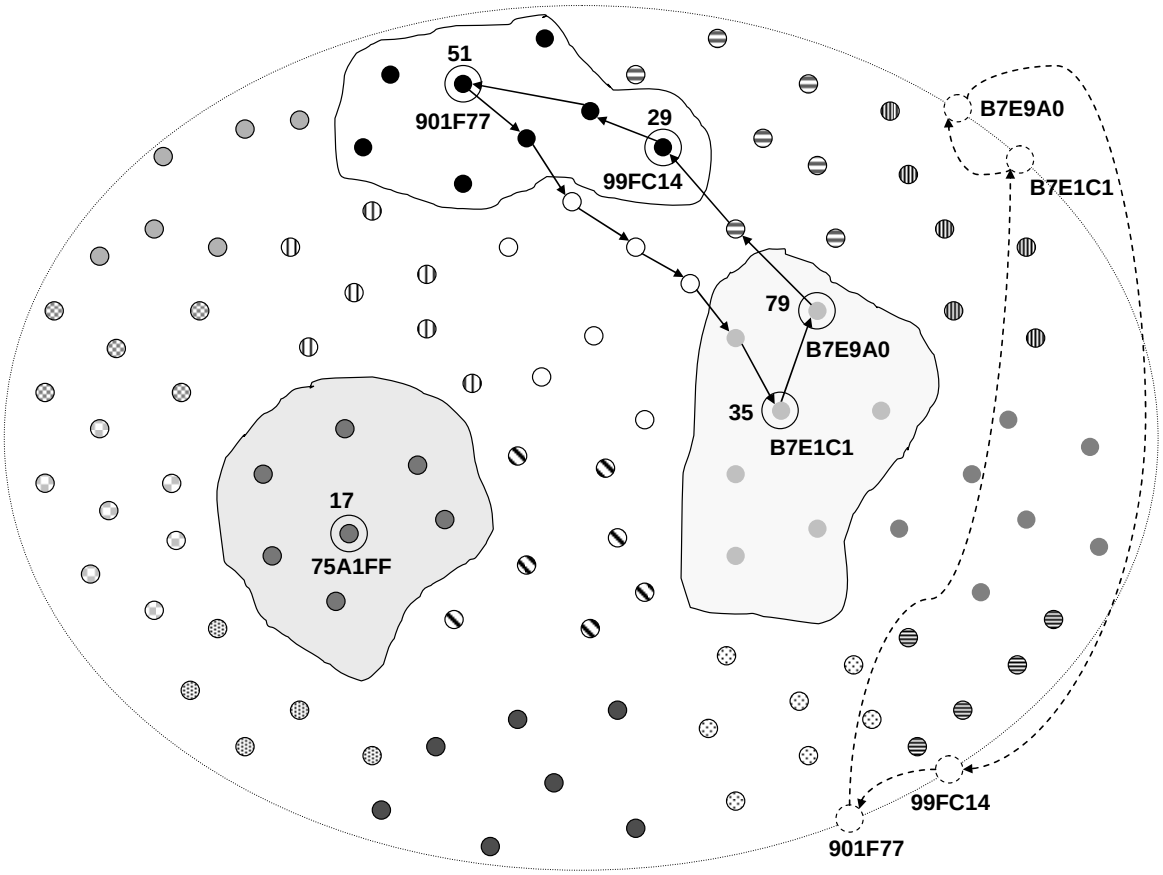

Figure 7.2 Address resolution.

address to obtain its ASK: h(17)→B7A9CC. Next, node 51 sends a request towards key B7A9CC using MADPastry, which is routed through node 35 (overlay ID B7E1C1) to node 79 – the current address server for node 17. Node 79, then, sends a response containing node 17's current overlay ID back towards the requester's overlay ID, which, in our example, is routed through node 29 (overlay ID 99FC14) back to node 51. Please note that, since MADPastry routes based on an overlay key, the response does not necessarily take the exact reverse path of the request – as is depicted in Figure 7.2.

Upon reception of the response from the address server, node 51 can now begin the actual unicast. Using the overlay ID 75A1FF (that of the destination node 17) provided by node 79, node 51 simply sends its data packet towards that ID. Figure 7.3 demonstrates how MADPastry routes the data packet to node 92 (overlay ID 7FF05C) during the first overlay hop, and then on to final destination node 17 who, trivially, is responsible for its own overlay ID 75A1FF.

7.3 Shortest Routing Paths vs. MADPastry's Routing Paths

The question that arises is why it would be better to route a packet to a known destination (i.e. the destination's node ID is known) indirectly over numerous routes (one for each overlay hop) instead of sending it directly over a single route.

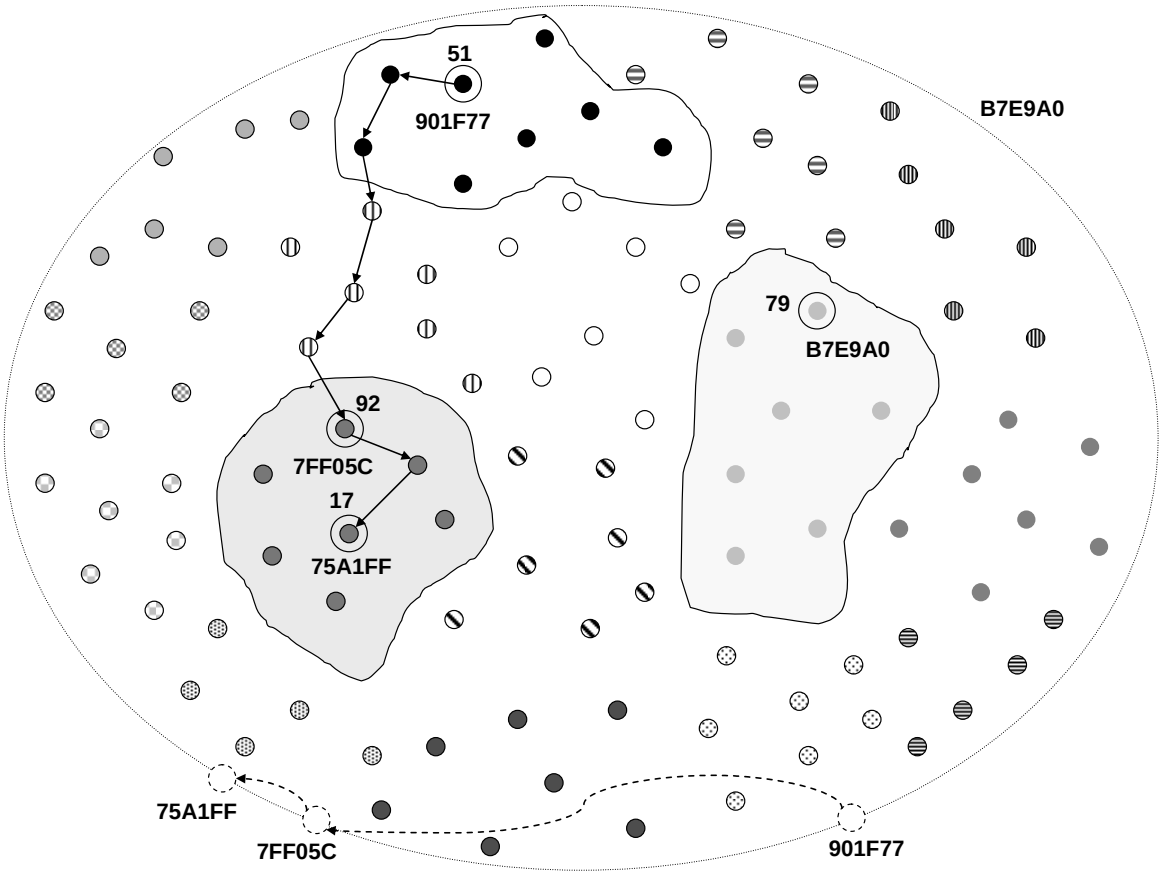

Figure 7.3 MADPastry's unicast.

For this, a closer look needs to be taken at the route characteristics of MADPastry and conventional ad hoc routing protocols.

Due to their route discovery schemes, data packets routed using conventional ad hoc routing protocols such as AODV [41], DSR [24], or OLSR [7] generally travel on a direct, straight path (quite likely also the shortest path) from the source node to the destination node – although this might no longer hold true as soon as the nodes comprising an established route continue to move about. On the other hand, packets that are unicast using MADPastry are likely *not* to travel on a direct, straight path. Instead, packets routed with MADPastry will probably travel multiple overlay hops – each of which can be thought of as a direct physical path from the current node to the destination node of the next overlay hop.

However, we do not consider this a drawback. On the contrary, it is important to bear in mind that MADPastry overlay hops are likely to be relatively short in terms of physical hops – especially once the packet has reached the destination cluster. Furthermore, a MADPastry node always replaces an existing Pastry routing table entry whenever it overhears a packet from a node that also fits into the corresponding routing table slot. Thus, the entries in its routing table are likely to be up-to-date and valid. Furthermore, when a reactive routing protocol such as AODV wants to deliver a packet to a destination node and it does not know the path to that destination, it needs to engage in a costly network-wide route discovery process. MADPastry, on the other hand, usually will not have to discover a route because it can use at each overlay routing step *any* existing route

that would bring the packet numerically closer to its key. On the other hand, one could think of MADPastry's address resolution scheme described in the previous subsection as its route discovery. But again, this address resolution also uses MADPastry's indirect routing, which, too, is likely to consist of short and recently updated routes.

Therefore, we believe that it can, in fact, be advantageous to travel numerous routes that are relatively short and likely to be up-to-date instead of traveling a single long and direct path – especially as the network size and, thus, the average path length between two arbitrary nodes increases. This belief is corroborated by the simulation results in the next section.

One can further expect that, when two nodes are engaged in bi-directional communication using MADPastry, the request packet will likely be routed on a different path compared to the response packet, as the corresponding Pastry routing table entries at the source and destination node can differ substantially. This can help distribute the load of bidirectional communication more fairly. This behavior was already sketched out in Figure 7.2.

7.4 Experimental Results

MADPastry's unicast performance is evaluated in ns-2. To put MADPastry's results into perspective, its results are compared to those of both a popular reactive routing protocol – AODV [41] – and a popular proactive routing protocol – OLSR [7]. For AODV, the AODV-UU 0.9.1 [1] implementation is chosen due to its much better RFC compliance compared to the standard AODV implementation in ns-2. For OLSR[1], we implemented a routing agent for ns-2 in compliance with the RFC 3626 [7].

As traffic source, each node runs a simple application on top of the respective routing agents that periodically picks a random node and sends a request to that target node. Upon reception, the target node sends a reply back to the originator. In other words, each node periodically engages in a bidirectional communication with a random target node.

Similar to the previous chapters, the following metrics are analyzed:

Success Rate. This is the percentage of random requests that eventually deliver a response back to the originator node. In other words, the success rate is simply

[1] Please note that the results of OLSR presented in this section deviate slightly from the initial results presented in [71]. For the initial results in [71], a third-party implementation [58] of OLSR for ns-2 was used with manually optimized control message intervals. Since, by default, [58] uses IPv6 node addresses, the effects of having optimized message intervals were largely cancelled out by the longer node addresses so that the performance of OLSR closely resembled that of OLSR with IPv4 addresses and default settings as specified in [7]. Since the initially used OLSR implementation [58] suffered from *severe* performance problems (in terms of the duration of a simulation), we reimplemented OLSR for ns-2 in compliance with the RFC 3626 [7] to provide a more comprehensive analysis of OLSR in this section. This reimplementation runs *significantly* faster than [58] and also achieves noticeably better routing results.

defined as the ratio between the total number of successfully received responses and the total number of sent requests. Note that in the case of MADPastry this request/response communication can be preceded by an additional address resolution communication as described in Section 7.2 if the target node's current overlay ID is unknown

Packet Overhead. This is the total number of packets that are forwarded during the entire simulation. This count is increased whenever a node forwards a packet to the next physical hop – in other words, whenever the MAC layer of a node is being passed a packet down from an upper layer. Hence, it includes all application and router packets such as AODV route requests and replies, OLSR control packets, MADPastry maintenance packets, and application requests and replies, etc.

Overall Traffic. Since many different packet types (e.g. AODV route requests, MADPastry packets, application requests and replies, etc.) of various packet lengths are transmitted during a simulation run, not merely the total packet count is analyzed. Additionally, the total network traffic in Kbytes that is created during the simulated hour is also considered. Whenever a node forwards a packet, this figure is increased by the packet size. Again, this figure includes *all* routing and application level packet types (AODV, OLSR, MADPastry and application packets).

As in the chapters before, all simulations that were carried out modeled wireless networks of 250 nodes over the course of one (simulated) hour. Nodes are always moving around according to the random way point model with 0s pause time (constant movement). For data transmission, nodes are, again, using the 802.11 communication standard with a transmission range of 250m. The node density in the investigated networks is always 100 nodes/km². Again, for MADPastry, a 32-bit overlay ID space is assumed with hexadecimal overlay IDs. In other words, each overlay ID consists of 8 hexadecimals digits. Nodes advertise their current overlay IDs every 10s. Please refer to Appendix 9.2 for the list of MADPastry system parameters and the values used throughout this section. In the case of AODV, an active route timeout of 10s is used and local route repairs are enabled. All three routing agents use link layer feedback.

7.4.1 Network Size

Table 7.1 Simulation parameters and values – different network sizes.

Simulation Parameter	Value range
Simulation duration	3600s (simulated)
Network size	50, 100, 150, 200, 250
Network density	100 nodes/km²
Node mobility	1.4m/s (constant, 0s pause time)
Random request interval (per node)	10s

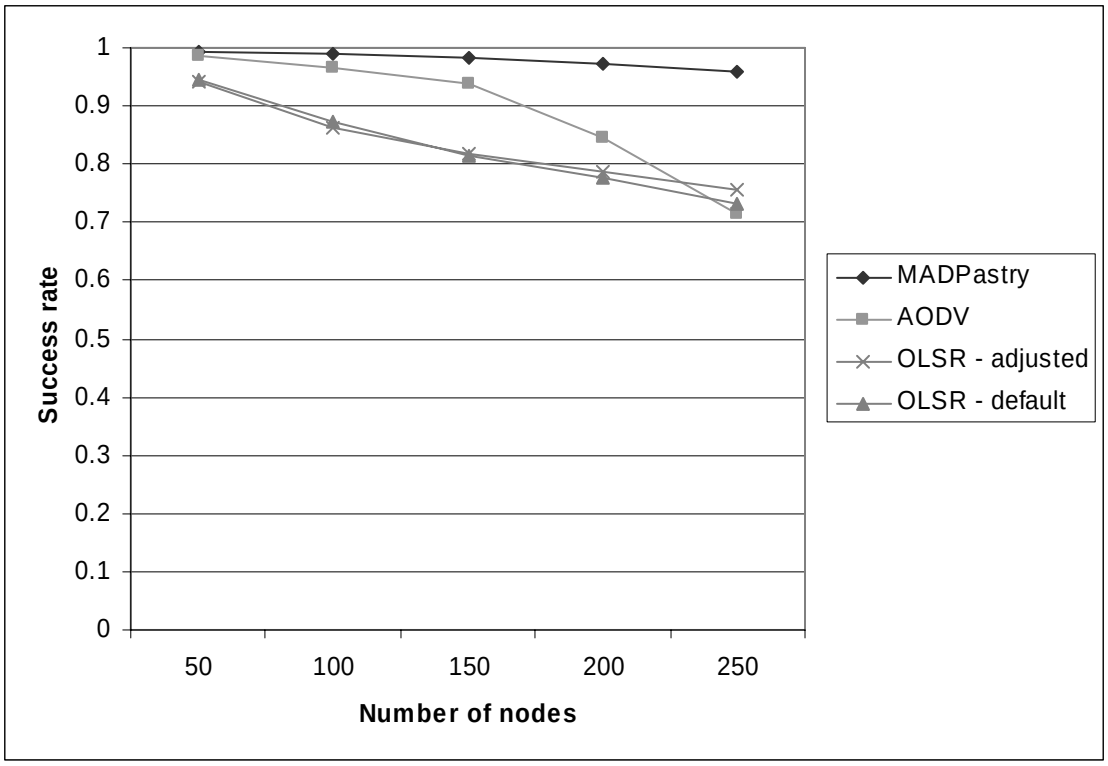

Figure 7.4 Success rate vs. network size.

In a first set of simulations, the success rate that MADPastry, AODV, and OLSR achieve in networks of varying sizes (50, 100, 150, 200, and 250 nodes) is examined. All nodes are constantly roaming around at a speed of 1.4 m/s (a quick walking pace). To measure the success rate, every node sends out a request to a random node every 10 seconds. Table 7.1 provides an overview of the simulation parameters and their respective values.

At this point, it is important to note that, being a proactive routing protocol, OLSR is very susceptible to "parameter tweaking". For this reason, the following results contain two different performance measurements for OLSR. First of all, "OLSR – default" shows the performance of OLSR with the default values for parameters such as control message intervals, the lifetime of routing table entries, etc. as stated in the RFC 3626 [7]. Additionally, "OLSR – adjusted" shows the results of OLSR with manually adjusted control message intervals. For the simulations presented in this section, a HELLO interval of 4s and a TC interval of 10s was manually chosen, which achieved the best results among all evaluated parameter combinations[1]. Note that it is well beyond the scope of this work to consider the question of how an OLSR network could dynamically adapt its system parameters to the present network environment. The purpose of "OLSR – adjusted" is merely to show what results OLSR could theoretically achieve if it had some means of dynamically optimizing its performance, as is, for example,

[1] Note that these chosen values do not necessarily have to be the exact optimal values for each simulated network. Due to the practically infinite amount of possible parameter value combinations (e.g. HELLO interval, TC interval, routing table hold times, etc.), it is practically impossible to always identify the optimal parameter values with certainty. However, we do believe that the chosen values closely approximate OLSR's best performance.

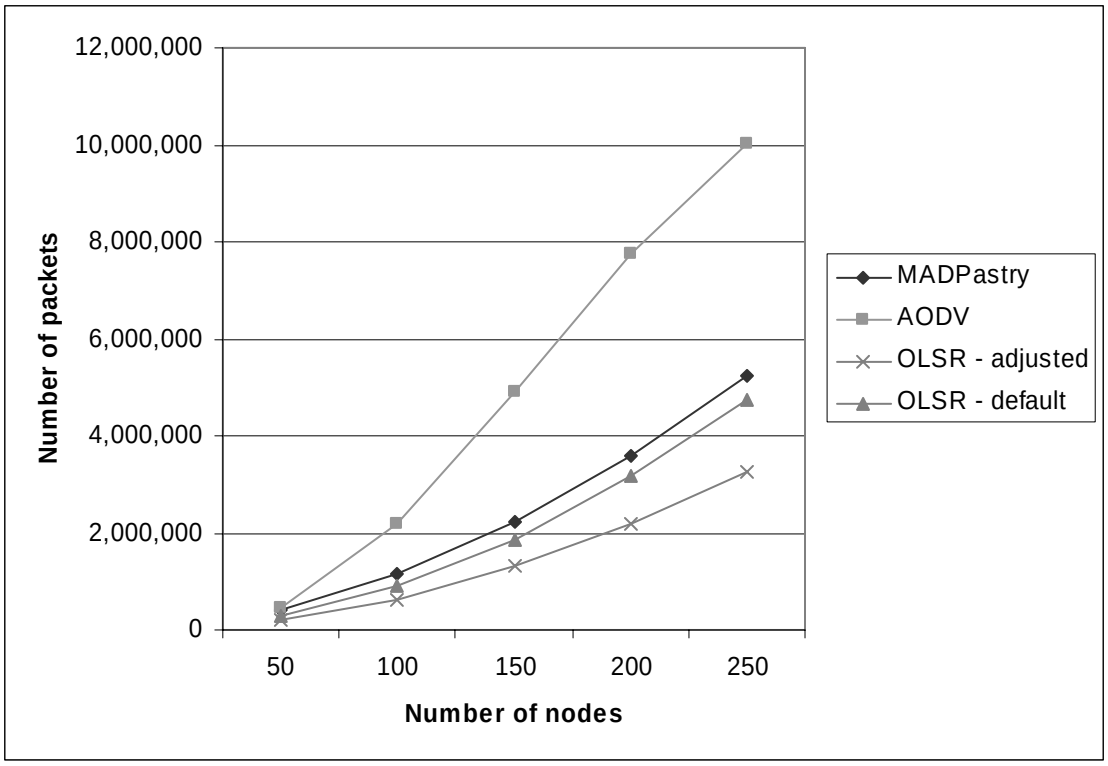

Figure 7.5 Total number of packets sent vs. network size.

proposed in [62]. Figure 7.4 shows the success rates of the respective unicast protocols versus the different network sizes. As can be seen, only MADPastry achieves success rates of above 95% for all network sizes. The success rates of both AODV and OLSR, however, drop quickly as the network size increases. As already discussed in Section 7.3, the reason for MADPastry's markedly better success rates is that MADPastry uses numerous short routes that are likely to have been recently updated. AODV and OLSR on the other hand, will try to route a packet on a direct route from the source to the target. As the network size increases, these long routes become more and more volatile and break frequently, which results in their lower success rates.

Figure 7.5 depicts the total amount of packets exchanged during one simulated hour. These numbers include any packets generated by the routing agents and the applications – i.e. data packets, routing packets, control packets, etc. One can see that MADPastry produces far less packets than AODV does. This is because MADPastry often uses short and recently updated routes, and, therefore, it rarely has to engage in AODV-style route discoveries. AODV on the other hand frequently needs to (re-)discover or repair its long and direct routes. These broadcast route discoveries also frequently collide with other packets (e.g. data packets), which further explains AODV's lower success rates as seen in Figure 7.4.

It is interesting to note that both instances of OLSR produce fewer packets than MADPastry does. This is because OLSR maintains its routes by periodically exchanging control messages. Therefore, when a route for a data packet is unknown, no route discovery is started and the data packet is simply dropped. In the networks considered, physical routes break frequently and the periodic

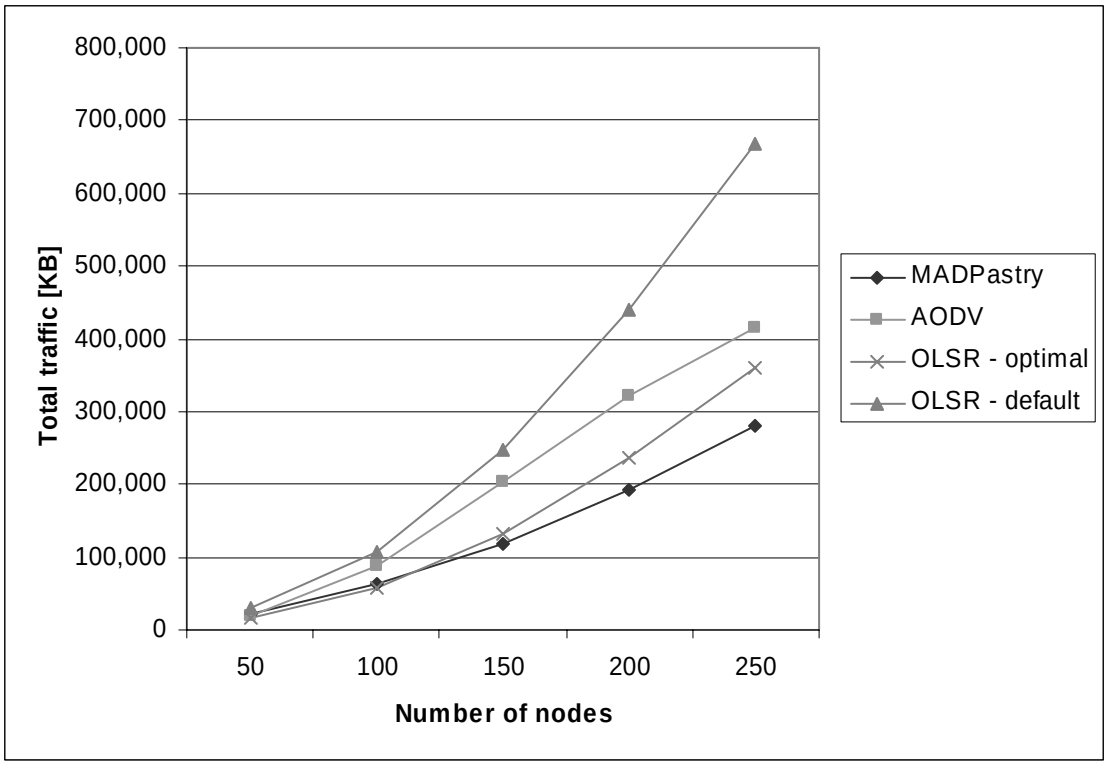

Figure 7.6 Total amount of generated network traffic vs. network size.

exchange of control messages cannot always maintain valid routes from each node to every other node. Thus, data packets are frequently dropped because no (valid) route is available, which explains OLSR's overall lower success rates.

Due to varying packet sizes, the number of packets sent has only limited significance. Figure 7.6, therefore, shows the overall amount of generated traffic. As before with the number of packets, MADPastry produces significantly less traffic than AODV does. Furthermore, Figure 7.6 shows another reason why OLSR's success rates are below MADPastry's. Since OLSR TC messages usually contain large amounts of node addresses, the messages themselves are usually quite large as well. Thus, although both instances of OLSR sent fewer packets than MADPastry does, they both generate markedly more traffic. "OLSR – default" even produces significantly more traffic than AODV does. Of course, this large amount of control traffic frequently collides with data packets, which further explains OLSR's lower success rates.

7.4.2 Node Velocity

So far, the networks considered have used a node velocity of 1.4 m/s. Therefore, we next evaluated the performance of the routing protocols for varying node velocities of 0.1 m/s, 1.4 m/s, 2.5 m/s, and 5.0 m/s. For this set of simulations, we consistently used a network size of 250 nodes. Table 7.2 provides an overview of the simulation parameters and their respective values.

Figure 7.7 shows the respective success rates. As can be expected, the success rates start decreasing as the node velocity increases. This is intuitive as with higher node velocities routes start breaking more frequently and MADPastry and AODV have to engage in route discovery more often. In the case of OLSR, its

Table 7.2 Simulation parameters and values – different node velocities.

Simulation Parameter	Value range
Simulation duration	3600s (simulated)
Network size	250
Network density	100 nodes/km²
Node mobility	0.1, 1.4, 2.5, 5.0 m/s (0s pause time)
Random request interval (per node)	10s

periodic control messages can no longer maintain valid routing tables in fast changing networks (i.e. high node velocities), which explains why OLSR's performance drops well below AODV's. MADPastry, on the other hand, still achieves significantly better success rates compared to the other routing protocols in all tested scenarios – although MADPastry's success rate drops to around 80% for node velocities of 5.0 m/s. This is because, at such a high velocity, MADPastry's clusters start becoming transient so that MADPastry frequently needs to resort to AODV-style route discoveries.

This becomes obvious in Figure 7.8. As can be seen, MADPastry's AODV-style route discoveries start having a more profound impact on the overall traffic the higher the node velocity is. The overall traffic produced by AODV, on the other hand, is largely unaffected by the node velocity since, with completely random request every 10s, AODV has to almost always engage in a route discovery before

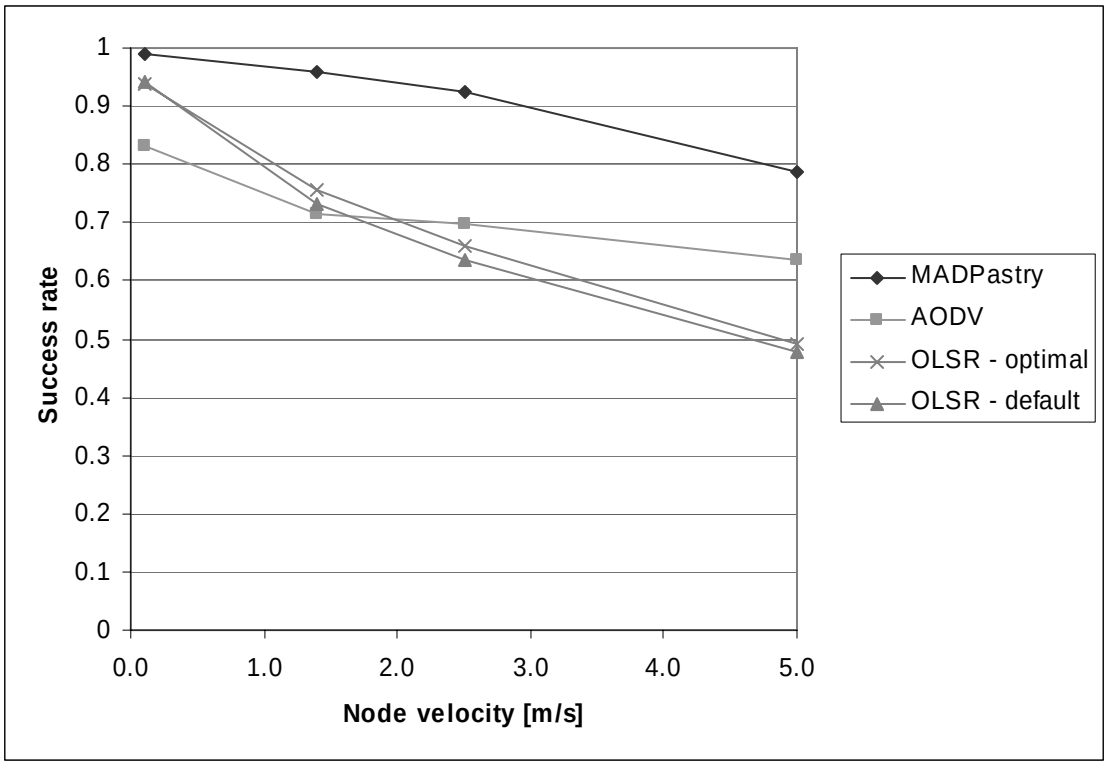

Figure 7.7 Success rate vs. node velocity.

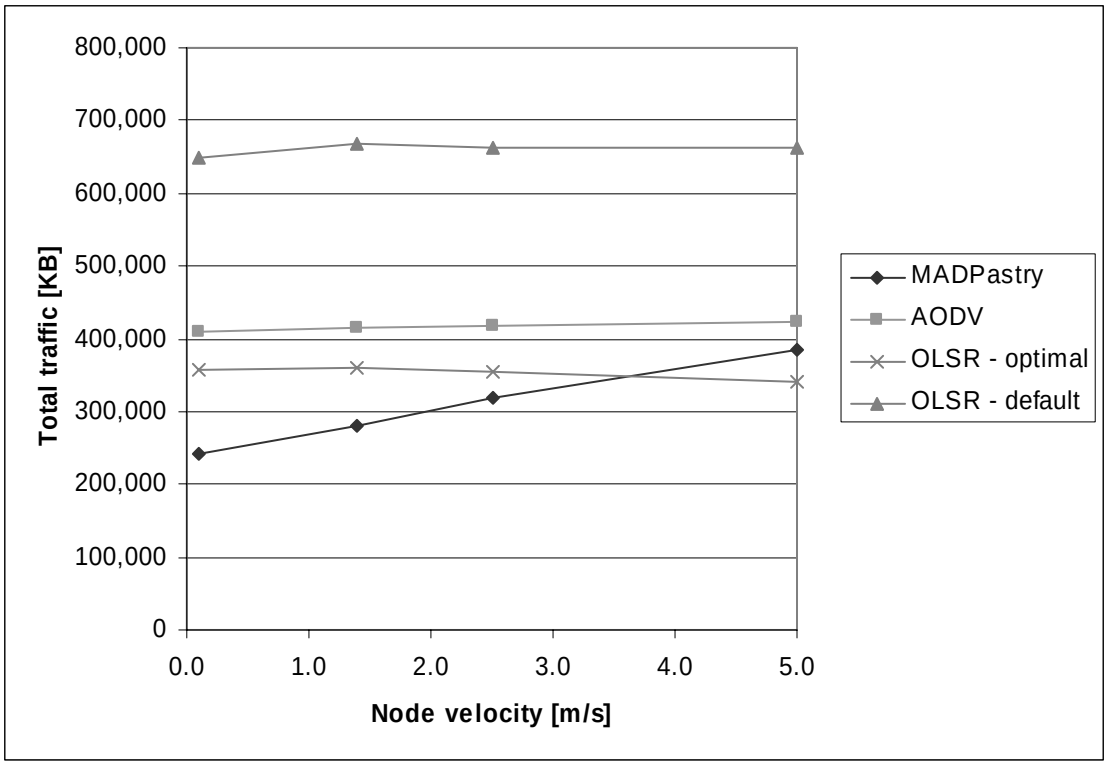

Figure 7.8 Total amount of generated network traffic vs. node velocity.

sending a data packet regardless of the chosen node velocity. In the case of OLSR, an interesting behavior can be observed. At the highest node velocity of 5.0m/s, OLSR produces less traffic than in the slower scenarios. To understand this effect, one needs to bear in mind that OLSR proactively exchanges periodic control messages. Thus, the route maintenance traffic induced by OLSR is completely independent from the node velocity. The reason why OLSR's overall traffic decreases with faster node velocities is simply that more and more often no valid route is known for a data packet. Thus, the data packet is dropped and the overall data traffic decreases, which results in the rapidly decreasing success rate.

Overall, MADPastry achieves significantly better success rates for all tested node velocities with markedly less generated traffic (with the exception of 5.0 m/s, where "OLSR – adjusted" produces somewhat less traffic at the expense of a success rate that is 30% lower than MADPastry's) than the conventional routing agents do. These simulations results support the initial assumption that, in large MANETs with constant node movement, it can, indeed, be favorable to route a packet to its destination indirectly over numerous relatively short and recently updated routes instead of using a single long and direct but possibly volatile route.

7.5 Related Work

A large body of work exists on unicast protocols for MANETs. Please refer to Section 2.2 for an overview of such ad hoc routing protocols. However, to the best

of our knowledge, none of the existing approaches use a DHT – which might be present in the MANET already to supply indirect routing – to provide unicasting in MANETs.

7.6 Summary

Far from proclaiming MADPastry's unicast scheme to be the one and only optimal ad hoc routing scheme for all conceivable MANETs, the purpose of this chapter was to demonstrate how MADPastry can be used to not only provide indirect, key-based overlay routing but also for providing DHT-based point-to-point unicast routing. The presented simulation results show that the unicast approach based on MADPastry generally outperforms both a popular reactive ad hoc routing protocol (AODV) as well as a popular proactive ad hoc routing protocol (OLSR) decidedly in the scenarios considered – both in terms of packet delivery rates and network traffic. This might be especially useful for MANETs that are already running a DHT application in the first place. In this case, nodes would no longer have to maintain a separate ad hoc routing protocol in addition to their DHT, but instead they could let MADPastry handle their point-to-point routing as well.

The simulation results presented in this chapter provide a first and encouraging look at the performance of DHT-based unicasting in MANETs. To further evaluate DHT-based unicasting, additional simulations would be needed. It would be particularly interesting to study the effects that varying network parameters such as node density, request rates, request distributions, etc. might have on the performance. Furthermore, it would be interesting to compare the performance of MADPastry's unicast to additional existing ad hoc routing protocols, especially hybrid (e.g. [21]) or hierarchical approaches (e.g. [39]).