4 The MADPastry Architecture

This chapter introduces the Mobile Ad Hoc Pastry (MADPastry) [69] architecture. MADPastry is 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.

4.1 Motivation and Architectural Overview

With the ever increasing proliferation of mobile, wireless devices, it is becoming more and more interesting to build efficient distributed network application for MANETs. In the Internet, Distributed Hash Tables have been successfully used as general-purpose building blocks for such applications. However, conventional DHTs are not well-suited for a deployment on top of such MANETs (see Sections 1.1 and 3.1). Other related approaches were designed for a specific application such as file-sharing or name resolution, or function efficiently merely in small and relatively stable networks, or require specialized hardware such as a GPS receiver (see Section 3 in general).

Therefore, MADPastry has been explicitly designed as a general-purpose DHT for mobile ad hoc networks without requiring any location information. MADPastry combines PASTRY-based [50] overlay routing and AODV-based [41] reactive ad hoc routing at the network layer to provide efficient indirect, key-based routing in MANETs. Figure 4.1 illustrates MADPastry's role in the network stack. An application sends a packet based on a key (in the example `72FE71BA`) and passes it down the network stack to the network layer (note that for simplicity and clarity the layers between the application and the routing protocol were omitted). MADPastry as the routing protocol examines the packet's key and determines the next overlay hop destination and sends the packet off toward that destination by passing the packet further down the stack.

MADPastry explicitly considers the following design criteria:

Locality awareness. MADPastry uses *Random Landmarking* (see Section 2.3) to map the physical topology to its overlay topology. MADPastry constructs clusters of physically close nodes that share a common overlay ID prefix. Therefore, physically close nodes in MADPastry are also quite likely to be close to each other in the overlay ID space.

Consideration of physical routes in the overlay routing process. As discussed earlier, an overlay hop always requires a physical route from the source node to the destination node of the overlay hop in order to be executed. If

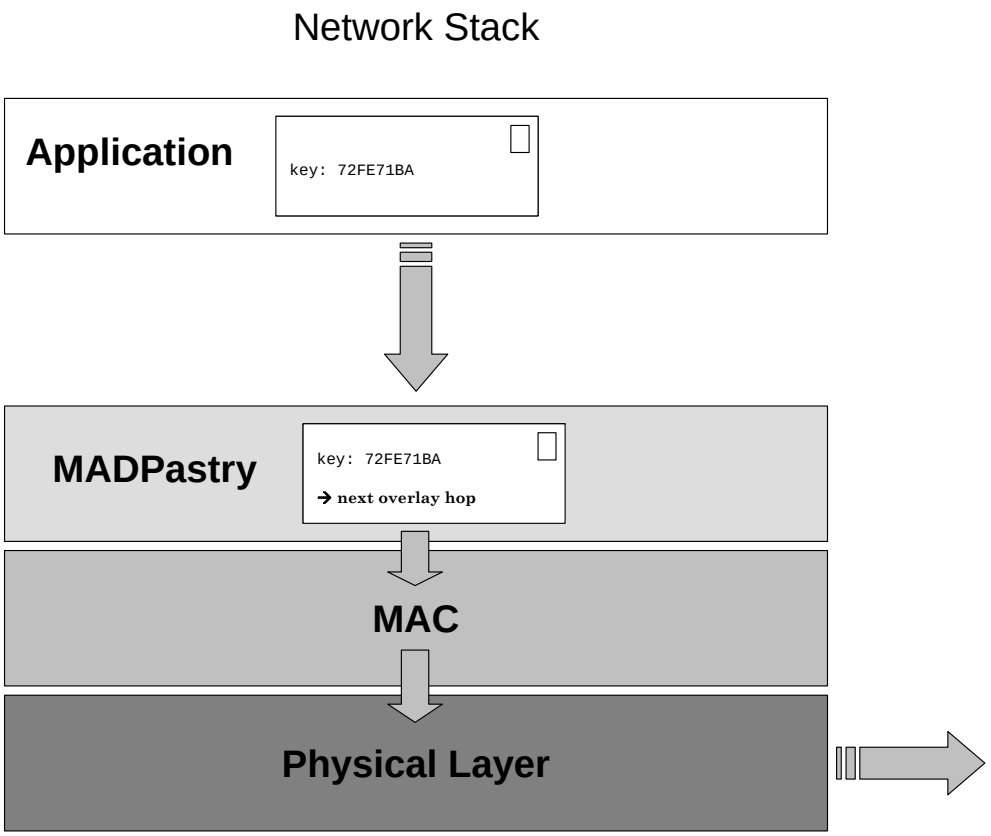

Figure 4.1 MADPastry in the network stack.

the physical route for an overlay hop is unknown, it has to be discovered previous to the execution of the overlay hop. However, route discovery is one of the most expensive (in terms of generated network traffic) tasks in MANETs. To avoid physical route discovery whenever possible, MADPastry might deviate from optimal overlay routing if the physical route of an overlay hop is unknown. Instead, MADPastry might choose a less optimal (in the sense of the overlay ID space) next overlay hop whose physical route is known, thereby favoring low physical traffic over optimal overlay routing.

Extensive exploitation of packet information. As described in Section 2.1.2, the maintenance of the overlay routing tables can be quite expensive with DHTs. To reduce the routing table maintenance overhead, MADPastry nodes exploit any packet information they receive. For this purpose, MADPastry augments its packet headers with both overlay and ad hoc routing information about the current node. This way, any node overhearing a packet can update its routing tables on-the-fly without having to engage in explicit routing table maintenance.

Standard DHT interface and functionality. By providing indirect, key-based routing and a conventional DHT interface, MADPastry constitutes a general-purpose DHT substrate for MANETs. Thus, distributed applications (e.g. [10, 49, 34, 51, 74, 2, 6, 8, 43, 61]) that have been build using DHTs in the Internet can be ported for a deployment in MANETs in a straight-forward manner.

The question arises why the combination of Pastry and AODV is chosen over other possible DHT / ad hoc routing combinations. Generally speaking, the design criteria and techniques described above are practically independent of any

concrete DHT. Nonetheless, Pastry is chosen for the following reasons. First of all, Pastry has been used for as a building block for a large number of various DHT-based applications and has been examined and analyzed in innumerable research papers. All of this makes it one of the most extensively studied DHTs proposed. Furthermore, with its quite flexible routing table structure – i.e. there are usually many possible candidates for a particular routing table slot – it is generally believed to be much better suited both for locality awareness and for the exploitation of overheard packets than, for example, Chord [56] with its strict definition of finger entries is. As for the ad hoc routing component, a reactive protocol is chosen over any proactive routing protocol in order to avoid the constant – and often considerable – overhead of a proactive routing table maintenance. In fact, the few proactive elements of MADPastry (such as the simplified leaf set maintenance – as described in Section 4.4) are already taken care of by the DHT component in any case. AODV is chosen as the reactive ad hoc routing component for the following reasons. Much like Pastry in the domain of Distributed Hash Tables, AODV is without doubt one of the most popular and most extensively investigated reactive ad hoc routing protocols. It is, furthermore, considered for standardization by the IETF [23]. Additionally, as opposed to DSR [24], we believe AODV is more appropriate for the relatively large MANETs that we consider with physical routes likely consisting of numerous hops since AODV's next hop routing keeps the header size constant – whereas DSR's source routing increases the header size with each additional hop.

4.2 Clusters

Nodes in a MADPastry network possess *two* IDs. First of all, each node has a unique and fixed node ID – for example its MAC or IP address. Additionally, each MADPastry node also has a unique but dynamic overlay ID that is assigned from a Pastry-style circular overlay ID space. While the node ID remains fixed during the lifetime of a node, the node's overlay ID can change depending on the node's position in the physical network.

As already mentioned, MADPastry utilizes the concept of *Random Landmarking* (see Section 2.3) 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*. Landmark keys are simply overlay IDs that 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. 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 IDs 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.

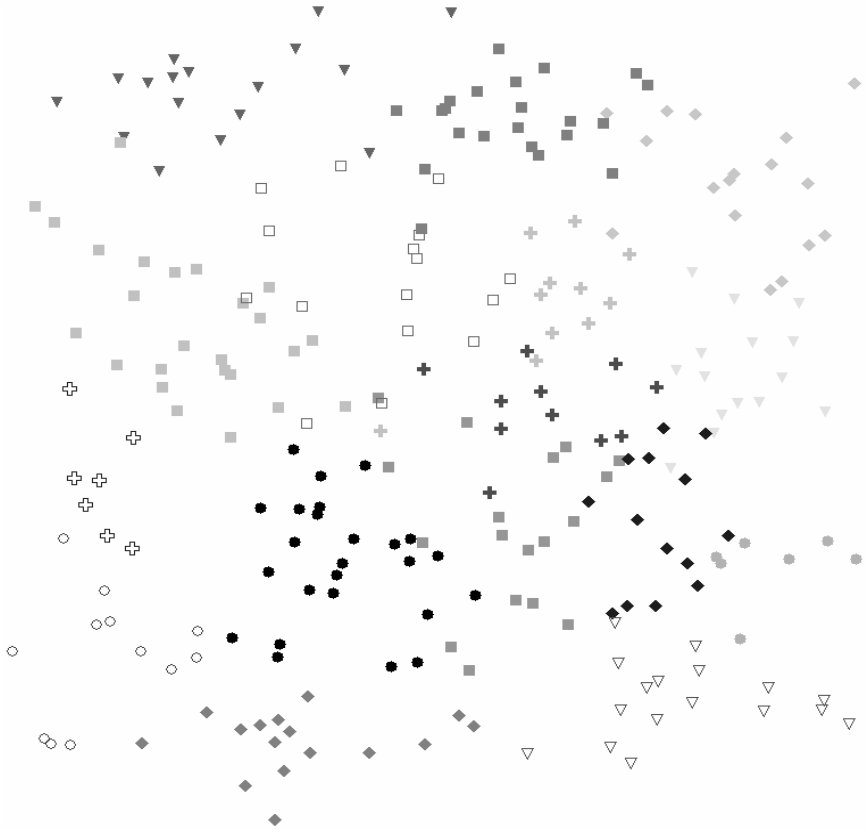

Figure 4.2 Spatial distribution of overlay prefixes in a 250-node MADPastry network.

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 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 under its new ID.

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 Figure 4.2 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 [5, 50].

Broadcast messages impose a serious burden on wireless networks. Therefore, 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.

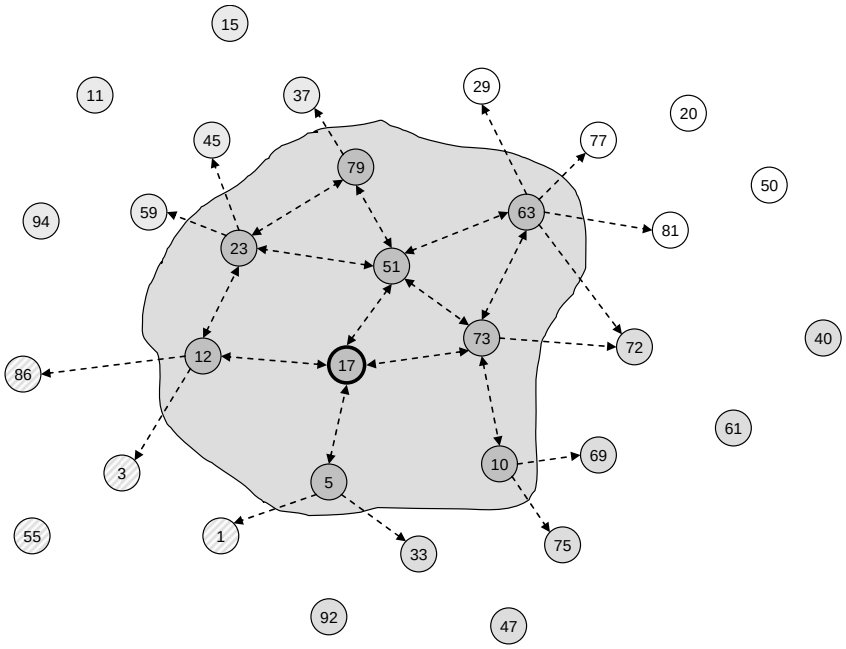

Figure 4.3 Beacon cluster broadcast.

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. Thus, 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. Non-landmark nodes also periodically advertise their existence within their own cluster using the same cluster broadcasts, but those beacons are not used to re-evaluate cluster memberships.

Figure 4.3 demonstrates this cluster broadcast. The temporary landmark node 17 issues a beacon that is broadcast inside its own cluster. Equal colors/shades represent nodes who share an overlay prefix – i.e. who make up a cluster. Note that, inside the cluster, the propagation of the beacon is symbolized with double-arrows. This is because each node that receives a beacon will forward the beacon on to its one-hop neighbors which, due to the nature of wireless communication, will of course also include the previous node from which the beacon was received (under the assumption of bidirectional links). However, nodes will not forward the same beacon twice. As can be seen, border nodes of neighboring clusters also receive the beacon but, since their overlay ID prefixes differ from the beacon prefix, will not forward the beacon.

4.3 Routing Tables

MADPastry nodes maintain 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 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 (see Section 2.1.2). This serves two purposes: First of all, nodes can thus learn about new nodes and fill empty routing table slots. Secondly, for routing table optimization, nodes can ping the 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 given row. Obviously, the traffic induced by this maintenance process constitutes a large portion of the overall traffic and can easily overwhelm a wireless network.

In standard Pastry, on average $\lceil \log_{2^b} N \rceil$ rows in a routing table are populated [50]. To avoid the expensive routing table maintenance overhead, MADPastry nodes store only sparsely filled Pastry routing table. In fact, a MADPastry routing table only needs to contain $\lceil \log_{2^b} K \rceil$ populated rows, with K being the number of landmark keys. In other words, it only needs to fill as many rows as are necessary to have a "pointer" entry to each overlay cluster. For example, with b=4 (hexadecimal overlay identifiers) and K=16, it would suffice for a MADPastry node to merely store 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 stripped 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 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 (see Section 2.1.2). Clearly, this process will be 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-node "pure" MANETs ([19]) without any wired infrastructural gateway nodes (in wireless-cum-wired topologies, for example, one could again have the wired gateway nodes maintain complete Pastry routing tables). 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 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. 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

Local Overlay ID: 3CE54CA7

Pastry Routing Table

row	0	1	2	3	4	5	6	7
0	**0**3761261 **nodeID 12**	**1**BE4873B **nodeID 78**	**2**BBAEF29 **nodeID 117**		**4**55D125F **nodeID 54**	**5**AC101E6 **nodeID 67**	**6**FF47C7A **nodeID 151**	**7**11C4B01 **nodeID 109**

Pastry Routing Table cont'd

row	8	9	A	B	C	D	E	F
0	**8**6596535 **nodeID 27**	**9**354C24B **nodeID 49**	**A**7AA51C6 **nodeID 243**	**B**7CF5174 **nodeID 126**	**C**A52CE41 **nodeID 17**	**D**301A17E **nodeID 61**	**E**55C0772 **nodeID 81**	**F**105B6FA **nodeID 97**

Leaf Set

smaller	larger
379E2070 **nodeID 47**	3CEF7003 **nodeID 57**
390B56E1 **nodeID 72**	3D42FE1C **nodeID 192**
3B76A92E **nodeID 63**	3DF4102F **nodeID 136**
3C017EEA **nodeID 31**	3F02CD52 **nodeID 44**

AODV Routing Table

Dest	Next Hop	Other
12	47	...
17	57	...
27	136	...
31	31	...
44	44	...
47	47	...
49	72	...
54	57	...

Dest	Next Hop	Other
57	57	...
61	192	...
63	63	...
67	136	...
72	72	...
78	44	...
81	72	...
97	136	...

Dest	Next Hop	Other
109	192	...
117	72	...
126	47	...
136	136	...
151	57	...
192	192	...
243	192	...

Figure 4.4 MADPastry routing tables.

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 they 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 overlay ID ring would break and 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 in its own cluster. Thus, these cluster beacons help nodes keep their leaf sets quite accurate. Therefore, 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 (see Section 2.2.1).

Figure 4.4 shows the routing tables of an example node with overlay ID `3CE54CA7`. In this example, hexadecimal overlay IDs are assumed along with 16 landmark keys. Note that the node has one entry for each of the 15 clusters other than its own. The entries in the Pastry routing table and leaf set store the overlay IDs and node IDs of the respective nodes. For the physical paths to the entry nodes, the AODV routing tables contains the required information.

4.4 Routing

MADPastry provides an *indirect routing* primitive for MANETs. I. e., MADPastry routes packets based on an overlay ID. The final (physical) target node – i.e. the node currently responsible for the packet's key – is usually unknown. Therefore, when a node (or the application running on the node, rather) wants to send a packet to the node currently responsible for the packet's key, it leaves the destination fields of the packet blank and passes it down the stack to the MADPastry routing agent. MADPastry will, then, consult its Pastry routing table and/or leaf set to determine the destination of the next overlay hop. Next, it will consult its AODV routing table for the next physical hop towards the destination node of the current, pending overlay hop. Once the next physical hop is determined, MADPastry passes the packet down to the MAC layer so that the packet can subsequently be transmitted. This process is illustrated in Figure 4.5.

MADPastry integrates overlay and physical routing. Therefore, when a MADPastry node receives a packet, it can now 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. Again, it will then consult its AODV routing table to determine the physical route to the destination node of the next overlay hop.

2) The node could be an intermediate node on the *physical path* of an overlay hop that is being carried out. Now, the node will behave like a regular AODV node. It

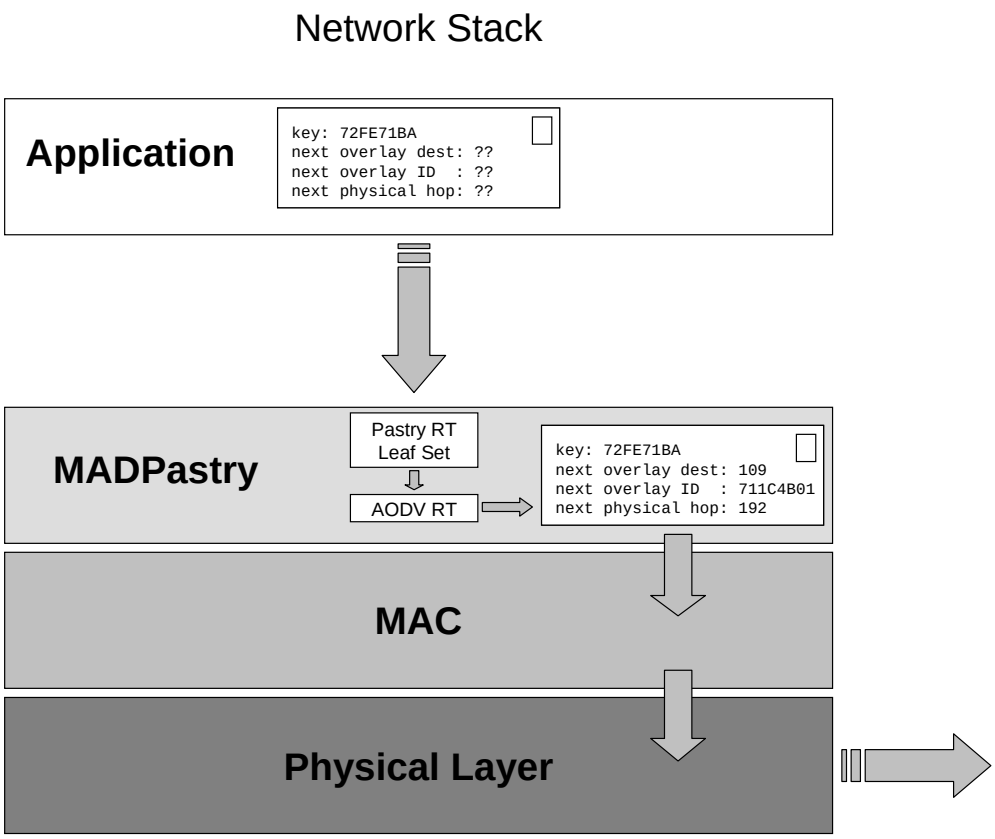

Figure 4.5 MADPastry routing - local view.

will consult its AODV routing table to determine the next physical hop on the path toward the destination of this overlay hop and then forward the packet on.

This process continues at each intermediate node until the packet eventually arrives at the node that is currently responsible for the packet's key.

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.

One of MADPastry's central goals is to avoid network-wide route discoveries whenever possible. Therefore, whenever a MADPastry node selects the next overlay hop from its Pastry routing table and/or leaf set, it checks whether a valid physical route to the destination node of that next overlay hop is known. If not, the destination node will be removed from the Pastry routing table and/or leaf set and the selection process is repeated. This process continues until a) a next overlay hop is determined for which the physical route to its destination node is known, *or* b) the destination of the next overlay hop is the "left" or "right" leaf of the current node, in which case the destination node must not be removed from the leaf set as otherwise the overlay ID ring would break. Note that, again, MADPastry deliberately accepts a less optimal overlay routing progress for the sake of avoiding expensive network-wide route discoveries.

However, an interesting question arises what should happen in case an overlay hop has been chosen for which the physical route to its destination node is unknown after all. This can happen in two situations:

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

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

As said, MADPastry's tries to avoid network-wide broadcasts whenever possible. Therefore, 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 key), 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 broadcast a regular, network-wide AODV-style route request to discover a route to the packet's destination.

Due to the dynamic overlay IDs in MADPastry networks, another special routing situation could occur. Some node A might change its overlay ID because it has joined a new cluster. In this case, node A will send a sign-off message to its former "left" and "right" leafs to inform them to remove node A under its old overlay ID from their leaf sets. However, another node B might still have node A under its old overlay ID in its routing table before the entry expires and might route a packet to node A based on that old overlay ID. In such a case, node A will send the packet back to node B including A's new overlay ID plus an additional tag signaling B to remove A's old overlay ID from its routing table(s).

Figure 4.6 provides a network view of MADPastry's routing algorithm. The dotted outer circle represents the circular overlay ID space. Otherwise, it shows the spatial distribution of the nodes. Equal colors and shades, again, represent equal overlay ID prefixes. In this example, node 17 with overlay ID `75A1FFE2` wants to send a packet to the node currently responsible for the key `B7EA2709`. After consulting its Pastry routing table, in the first overlay hop, MADPastry will send the packet to node 4 with overlay ID `B207D11F`. For this purpose, MADPastry acquires the next physical hop from its AODV routing table: node 54. Upon the reception of the packet, node 54 checks its AODV routing table to determine yet the next hop towards node 4 and sends the packet to node 32. This regular AODV-style routing continues at each intermediate node until the packet reaches node 4 via nodes 39 and 90. Node 4 (or rather its MADPastry routing agent) realizes that it is the destination of the current overlay hop. Note that, after this first overlay hop, the packet has reached its target cluster. Now, node 4 will determine the next overlay hop using its Pastry routing table or leaf set. For the next overlay hop, node 4 will send the packet to node 35 with overlay ID `B7E1C101`. Consulting its AODV routing table, node 4 forwards the packet to node 47, who will forward it node 35. Node 35 then determines the next overlay hop – node 79 with overlay ID `B7E9A014` – and sends the packet to that

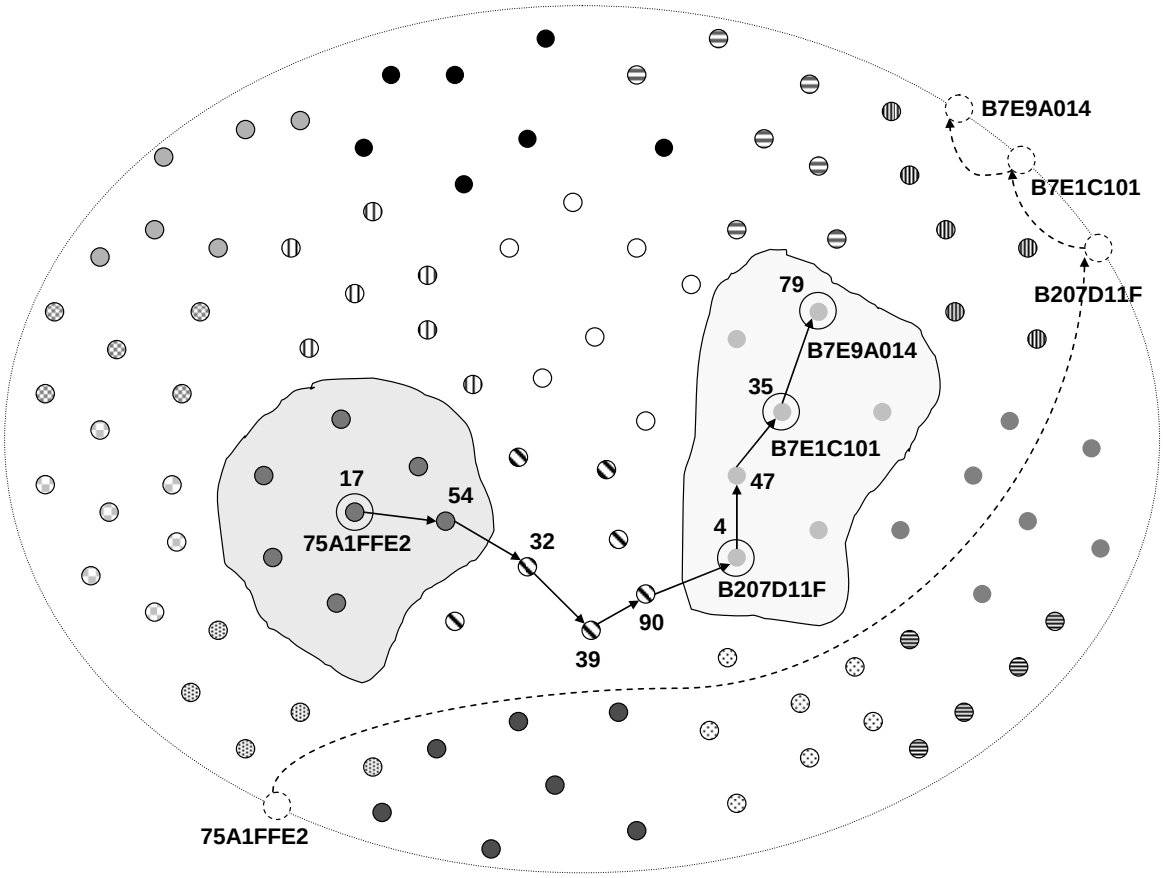

Figure 4.6 MADPastry routing - network view.

destination using its AODV routing table. Node 79 realizes that it is responsible for the packet's key and, thus, the packet has been delivered.

Again, it is important to bear in mind that MADPastry merely provides indirect routing for 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 a packet is delivered. MADPastry merely delivers a packet to the node currently responsible for the packet's key.

4.5 Routing Table Maintenance

As described in Section 4.3, Pastry routing table maintenance can generate large amounts of traffic. Therefore, the only proactive routing table maintenance that a MADPastry node performs is the periodic pinging of its "left" and "right" leaf as this is necessary to guarantee overlay routing convergence. For this purpose, whenever a node B receives such a LEAF PING from a node A, node B checks its own leaf set to see if it (node B) really is the numerical predecessor, or successor respectively (depending on whether node A stated that node B be its "left" or "right" leaf), of node A. Node B then replies with a LEAF PONG containing the node that B thinks is node A's "left" or "right" leaf – i.e. ideally node B itself.

All other routing entries, however, 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 source (i.e. the destination node of the previous overlay hop)

- 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 always overwritten in 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 favor of fresh physical routes.

Again, the motivation here is that MADPastry prefers slightly less optimal paths over shorter but invalid paths that would have to be repaired using network-wide route discovery or repair mechanism. Of course, with this heuristic, it could happen that a relatively near Pastry routing table entry is replaced by an exceptionally remote one. However, as nodes frequently overhear packets from their vicinity, such exceptionally suboptimal routing table entries can be expected to be soon replaced by better entries themselves.

In Section 4.3, it was described that, in order to prevent expensive routing table maintenance, MADPastry – unlike standard Pastry – does not bother to keep on average $\lceil \log_{2^b} N \rceil$ rows in its Pastry routing table populated. For efficient key-based routing in MANETs, it would still suffice if a MADPastry routing table merely contained an entry to each cluster – hence, $\lceil \log_{2^b} K \rceil$ populated rows, with K being the number of landmark keys. In fact, it would actually be enough for each MADPastry node to simply know its left and right leaf in order for the key-based routing to still converge – but this would, of course, generate an enormous overlay stretch. Nonetheless, this does *not* imply that a MADPastry node also only stores $\lceil \log_{2^b} K \rceil$ rows in total in its Pastry routing table. In fact, a MADPastry node keeps a regular size Pastry routing table but does not actively maintain its content. Instead, a MADPastry node's Pastry routing table really behaves like a cache for the overlay IDs that are extracted from overheard packets.

It is clear to see that the fill degree and accuracy of the Pastry routing tables and leaf sets largely depends 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 will 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.

4.6 Bootstrapping

MADPastry provides two methods for bootstrapping the network. First of all, a new node can join an existing network in the "regular" fashion. For this, a new node just needs to know one other node that is already part of the network (for example by broadcasting a bootstrap node discovery within a certain hop radius). To join, a new node simply assigns itself an overlay ID (quite possibly one that shares a prefix with the bootstrap node's overlay ID) and asks its bootstrap node to route a join request to the node currently responsible for that new overlay ID. Upon reception of such a join request, a node will send a response containing its leaf set back to the new node. Note that, deviating from the original Pastry join procedure, the reply will not contain any routing table information to keep the network traffic low as the new node will quickly fill its routing table by overhearing packets. Once the new node has received the join reply, it will contact its "left" and "right" leafs (one of which will be the node from which it received the join reply) to inform them about its arrival. This way, the new node inserts itself into the overlay ID ring and has thus joined the network.

The regular join method works well for individually arriving nodes. On the other hand, should a large group of nodes decide to start a MADPastry network, this could lead to many concurrent node joins and the network might need a considerable amount of time, during which nodes detect inaccurate leaf set entries through the periodic cluster beacons and leaf pings that nodes carry out, until it stabilizes. Therefore, MADPastry also offers another bulk bootstrapping method. Here, each node assigns itself a random overlay ID and at a random point during a certain start-up interval (e.g. 30s) broadcasts its overlay ID throughout the network so that, after the start-up interval, each node should be aware of the overlay IDs of all other participating nodes. This way, each node will be able to determine whether it is a temporary landmark node. During a shorter intermediate interval (e.g. 10s), the temporary landmark nodes issue their regular landmark beacons so that each node can determine its closest landmark node and adapt its overlay ID accordingly. After this intermediate interval, the initial start-up interval is repeated so that the participating nodes can broadcast their new, cluster-based overlay IDs. At this point, it shall be remarked that, for the bulk bootstrapping, it is assumed that *one* of the group members initiates the bootstrapping by sending out a network-wide broadcast containing the duration of the start-up and intermediate interval. How this initiator node would be elected among the nodes in the group (e.g. smallest node ID, etc.) is unrelated to the concept of MADPastry and thus considered beyond the scope of this thesis.

4.7 Summary

MADPastry enables indirect, key-based routing for MANETs. For this purpose, it combines Pastry overlay routing and AODV ad hoc routing at the network layer. Thus, MADPastry provides a DHT substrate for such MANETs. MADPastry explicitly considers physical locality in its overlay topology by employing Random Landmarking to group physically close nodes in clusters with common overlay ID prefixes. Furthermore, physical routes are also explicitly considered during the overlay routing process so that network-wide route discoveries are kept to a minimum. MADPastry nodes exploit the information included in all packets they overhear to update their routing tables without generating additional maintenance traffic. By providing a standard DHT interface and functionality, MADPastry can be used as a general-purpose building block for a wide variety of distributed network applications in MANETs.