Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Lehrstuhl für Rechnernetze und Telematik
Seminar: Ad Hoc Networks

WS 2009/2010

Seminar

MADPastry: A DHT Substrate for Practicably Sized MANETs

Thomas Zahn, Jochen Schiller

Seminar report written by: Elmar Haussmann

February 7, 2010

Supervised by Prof. Dr. Christian Schindelhauer
and Arne Vater

Abstract

Because of the increasing interest in mobile ad hoc networks (MANETs) and their inherent similarities to peer-to-peer networks a large number of routing protocols based on peer-to-peer networks have recently been presented to provide distributed network applications with efficient and indirect routing for MANETs. Thomas Zahn and Jochen Schiller suggested MADPastry, an integration of a DHT substrate based on the peer-to-peer network Pastry that explicitly considers locality, and the ad hoc routing protocol AODV at the network layer. In this paper a technical description of MADPastry is given followed by an overview and discussion of related work.

Contents

List of Figures

List of Abbreviations

AODV	Ad-Hoc On Demand Distance Vector Routing
DHT	Distributed Hash Table
DSDV	Destination-Sequenced Distance-Vector Routing
DSR	Dynamic Source Routing
e.g.	Latin: exempli gratia, English: for example
i.e.	Latin: id est, English: that is
MANET	Mobile Ad Hoc Network
OLSR	Optimized Link State Routing
RERR	An Error message in the Ad-Hoc On Demand Distance Vector protocol
RREP	A Route Reply message in the Ad-Hoc On Demand Distance Vector protocol
RREQ	A Route Request message in the Ad-Hoc On Demand Distance Vector protocol

1 Introduction

With technological advancements in performance, energy consumption and wireless communication of mobile or embedded devices they become a more and more widely used appliance for data exchange between a possibly very large number of devices. The number of possible applications supported by those devices is vast, however the networks formed share common characteristics: they are dynamic, the individual nodes are usually bandwidth as well as energy constrained and they are infra-structureless [5]. These networks are commonly referred to as mobile ad hoc networks (MANETs) where communication between mobile nodes is established on an ad hoc basis and a self-configuring network of data-relaying nodes is formed. One of the major challenges and a popular research topic in these networks is the efficient routing between nodes [8]. On the other hand, in an, at first unrelated topic, peer-to-peer networks and their applications recently gained, for a number of discussible reasons, great popularity in Internet applications. Peer-to-peer networks describe networks where equal peers take the role of both client and server responsibilities as opposed to the traditional client/server model [12] and provide interesting features for MANETs such as lookup of services or resources in a distributed fashion without central control. A number of similarites exists between peer-to-peer networks and MANETs [10] and to provide an efficient routing strategy as well as distributed network applications in MANETs Thomas Zahn and Jochen Schiller developed MADPastry [22] at the Technical University of Berlin. MADPastry combines a DHT substrate based on the peer-to-peer network Pastry [18] with the ad hoc routing protocol AODV [14] to exploit these similarities. In this paper a technical overview of MADPastry is given in 2 followed by a short overview of related work in 3 and a critical appraisal and conclusion in 4. An introduction to Pastry and AODV is provided in the Appendix.

2 MADPastry

With increasing interest in peer-to-peer applications as well as MANETs and the main similarities of both being infra-structureless, dynamic and requiring self-configurartion [22] it is a logical next step to provide a layer for peer-to-peer applications in MANETs. Typical DHT based peer-to-peer applications are however designed for the Internet where bandwidth is no longer scarce and latency usually low as opposed to MANETs where also energy is limited. According to [22] the main problems of applying DHT based peer-to-peer protocols in MANETs are *a*) the lack of explicit consideration of locality, *b*) the fact that, caused by node mobility, routes break frequently and underlying routing

protocols often revert to broadcasting which renders application level routing superflous and *c*) the maintenance overhead incurred by maintaining DHT routing structures. Given these constraints Thomas Zahn and Jochen Schiller designed MADPastry by combining Pastry and AODV at the network layer. This section first provides information about the architecture and routing algorithm followed by a summary of the simulation results of MADPastry presented in [22].

2.1 Integrating Pastry and AODV

To create a scalable, lightweight and indirect routing primitive at the network layer MADPastry both adapts Pastry and the AODV routing procedure. MADPastry utilizes data structures in form of Pastry routing tables as well as a standard AODV routing table. Because only to Pastry considerable structural changes are made these are considered first and afterwards the changes for AODV are introduced as part of the overall routing procedure.

2.1.1 Changes to Pastry

To provide an explicit consideration of locality in MADPastry so called *clusters* of nodes with same prefix are introduced. Because in an overlay neighboring nodes can be arbitrarily far apart from each other in the physical underlay[1], a common ID prefix is assigned to nodes that are likely to be physically close [2]. For this MADPastry uses so called *landmark-keys*, a set of keys that evenly divides the key space of the DHT. A node with ID closest to one of those predefined landmark keys is considered a *landmark-node* and periodically sends out beacons to inform neighboring nodes of its presence and distance. Nodes receiving the beacon join the cluster of the landmark node they are currently closest to by assigning themselves a new node ID with corresponding cluster prefix.[3] To minimize the generated traffic beacons are only broadcast within the own cluster as well as to nodes bordering the cluster, which are likely candidates for new members. The standard Pastry routing table consists of $log_{2^b}N$ rows with $2^b - 1$ entries in each row for a network of N nodes and a IDs interpreted to base $B = 2^b$. To reduce maintenance overhead MADPastry straps down the routing table to $log_{2^b}K$ rows with K being the number of clusters selected. Assuming hexadecimal IDs and e.g. 16 clusters, the routing

[1] This is also referred to as *overlay stretch*

[2] Nodes with the same prefix are then likely to be physically close as well as close in terms of the overlay

[3] Of course nodes need to leave their previous cluster and possibly *handover* objects they were responsible for as well as acquire responsibility for new objects

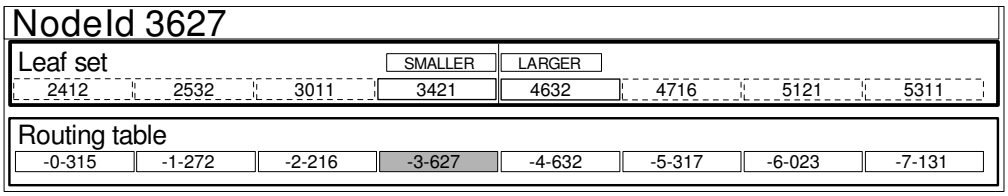

Figure 1: MADPastry routing table excerpt after [18]. $K = 8$ and $B = 8$ thus one row with a node for each cluster exists. One column (grey) is occupied by the node itself. Only the left and right neighbor of the leaf set are pro-actively maintained.

table would then consist of only one row where for each cluster an entry providing a direct link is present. To further reduce maintenance overhead the total correctness of the leaf set is abandoned[4] by only periodically pinging and querying the closest left and right neighbor instead of all members. Figure 1 shows an example of the modified Pastry routing structures and in Figure 2 the forming of clusters is represented by same colored nodes.
In summary the introduction of clusters introduces explicit representation of locality and the reduction of the Pastry routing table and leaf set and maintenance allow for reduced communication overhead by sacrificing scalability for large networks. This is because according to the authors the maintenance penalty incurred by a complete Pastry table outweighs its benefits for typical MANETs which are assumed to consist of up to 1000 nodes [22].

2.2 Message Routing in MADPastry

The routing procedure of MADPastry follows the standard Pastry and AODV routing process with some slight adjustments. Instead of strictly separating the application and network layer both are integrated for an increase of efficiency.

2.2.1 Routing Algorithm

Whenever a node receives a MADPastry message one of the following cases applies:

[4]It is important to realize that for correct routing it is sufficient for each node to know its correct left and right neighbor. This however sacrifices possible performance advantages (namely the expected number of messages in $O(logN)$ for routing with Pastry)

a) the node is the target of an overlay hop (i.e. the node's overlay ID equals the next overlay hop's ID according to the MADPastry packet[5]): proceed like standard Pastry routing i.e. consult Pastry routing table or leaf set for next overlay hop or consider message delivered

b) the node is an intermediate node on an overlay hop (i.e. the node's overlay ID is not equal to the next overlay hop's ID according to the MADPastry packet): consult the AODV routing table for the next physical hop towards the overlay hop target. If however the node's own overlay ID is closer to the target ID than the overlay ID of the next hop consider the overlay hop done (i.e. continue with standard Pastry routing)

An important change to the AODV routing strategy is that instead of using network wide broadcasts to discover missing routes a cluster wide broadcast can be used to directly deliver the MADPastry message to the overlay destination. This is possible if the node is already within the confines of the cluster the overlay destination belongs to. If however the node is not within the target's cluster a standard AODV expanding ring route discovery is applied. This strategy tries to avoid broadcasts for route discovery when an immediate broadcast with the message is more efficient.
Figure 2 shows an example routing procedure. The first overlay hop to the target cluster is handled by the MADPastry routing table, afterwards intra-cluster routing via the leaf sets takes place.

2.2.2 Overhearing of Packets

To further reduce generated traffic and to provide accurate and up-to-date information MADPastry leverages the information of overheard packets. Whenever a node overhears a packet it uses the information obtained by updating the AODV or Pastry routing tables with newer entries if possible[6]. The idea of relying on overheard packets has the drawback that accuracy and fill degree of the data structures depend on the traffic present in the network. In networks with low traffic one is, according to the authors, however better of broadcasting the message directly instead of maintaining an application level routing structure [22]. Thus MADPastry is mainly designed for networks with high traffic rates.

[5] The appendix A.3 shows the contents of a MADPastry packet
[6] In case AODV the route must be fresh enough in terms of the sequence number

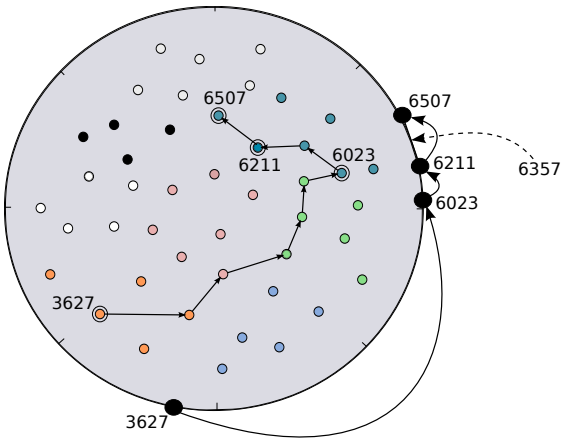

Figure 2: Message routing in MADPastry with $K = 8$ and $B = 8$ after [23]. Node 3627 is sending a message to 6357, which gets routed to the numerically closest node: 6507. The outer circle represents the overlay hops while inside the physical hops performed are shown.

2.3 Simulation and Results

To evaluate their design the authors of MADPastry run simulations of a number of different scenarios[7] over the duration of one simulated hour using a random waypoint model. The main question to be answered is whether MADPastry's overhead caused by the maintenance of a DHT routing structure is worth the effort compared to directly broadcasting the lookup [22]. MADPastry is thus compared to a Gnutella-style broadcasting agent, and an implementation of Pastry and AODV similar to MADPastry but without explicit consideration of locality using clusters. Overall MADPastry provides success rates comparable to the broadcast agent generating only a fraction[8] of the traffic which is attributed to the confining of broadcasts to clusters. Because of the smaller traffic incurred, MADPastry works better than its competitors in networks with a high traffic volume. However in scenarios consisting of fast moving nodes routes break too frequently for MADPastry to keep up and in networks with low traffic MADPastry's concept of overhearing packets can not excel. The standard implementation of Pastry and AODV performs similar to, or worse than, MADPastry in all scenarios confirming

[7] Variable network sizes of 100 and 250 nodes, as well as different node speeds between 0.1m/s and 5 m/s and different key lookup rates were evaluated

[8] Depending on the scenario 50% or less

the idea of explicitly considering physical locality. In a last set of scenarios the effect of nodes joining and leaving clusters with resulting object handovers is evaluated. MADPastry again outperforms its competitors showing first congestion signs, considerably later[9], with 1,000,000 objects in a network of 250 nodes.

3 Related Work

A large number of suggestions have recently been made on how to provide distributed applications in MANETs at the same time providing indirect or direct routing. Most of them use DHTs as a basis to provide a distributed application platform and some of them also consider physical locality. In this section we will only consider those that are to some extent related to MADPastry. In [13] an approach very similar to MADPastry is presented which is based on *marker keys* and *prefix clustering*, utilizing Bamboo, a modification of Pastry and AODV or OLSR [4] as the network layer routing protocol. PeerNet [9] differs from MADPastry in using CAN instead of Pastry but also explicitly considers physical locality using *zones* and *virtual residences*. On top mobility profiles and models can be applied to predict node movement.
However not only indirect routing is of concern which is why some protocols also provide direct point-to-point routing. In [23] Zahn and Schiller extend MADPastry to also provide unicast point-to-point communication by storing the mapping between node IDs and IP addresses using *address server keys* distributed in the network. Indirect Tree-based Routing [2] is also based on DHTs and provides both direct and indirect routing by using location dependent identifiers and applying a tree based routing algorithm [3]. A comparison to MADPastry in simulations[10] even shows a slightly better performance. Most of the protocols are complex which is why some approaches focus on formulating a simple and comprehensible routing procedure. One interesting approach taken is Virtual Ring Routing (VRR) [1] which is also inspired by DHTs and Pastry and forms a virtual ring between all nodes where nodes only maintain routes to neighboring nodes. VRR avoids broadcasting completely and does not rely on an underlaying routing protocol. Extensive simulations, also including real-life scenarios using motes, show that VRR performs equal or better than standard ad hoc routing protocols (DSDV, AODV, DSR). An evaluation of a simple Chord implementation in MANETs is discussed in [6] and compares AODV, DSR and OLSR as its network layer routing protocols. Results show that not the overhead of maintaining a DHT based routing structure but the pessimistic

[9] A lookup rate of 1 lookup per node per second was used which already congested the network for the broadcast agent and standard Pastry implementation in a previous simulation
[10] A network of 50 nodes was used

fail-over strategy of Chord causes a performance worse than flooding techniques.
Early approaches like DPSR[10] and its successor Ekta[15] again are based on Pastry but in contrast to MADPastry use DSR instead of AODV and do not explicitly consider physical locality. Similar to MADPastry, overhearing of packets to capture up-to-date information plays a major role. CrossROAD [7] is also based on Pastry and extends OLSR by hashing IP addresses in the routing table. A node maintains routes to all other nodes in the network similar to a fully connected mesh topology.
Given all of these approaches with similar capabilities the question arises which approach provides the most generic and efficient solution. However it seems that currently no such comprehensive comparison or evaluation exists. The number of different approches however clearly indicates the attention and importance the topic of distributed applications and routing for MANETs has.

4 Critical Appraisal and Conclusion

As the results presented in [22] show, MADPastry certainly provides an efficient protocol for indirect routing in MANETs and can serve as the basis for distributed applications. There are however some drawbacks or questionable decisions that need to be considered. On the one hand the choice for Pastry is not clearly articulated. For Ekta [15] this is due to the locality awareness provided by Pastry so similar motives can be assumed. However Pastry is already a complex protocol which makes theoretical proofs difficult [11] and explicit locality awareness could also be incorporated in simpler peer-to-peer protocols such as Chord. Virtual Ring Routing (VRR) [1] is a good example of a simpler protocol that even avoids broadcasts completely and also provides the benefits of DHTs. On the other hand MADPastry performs good in the scenarios provided by the authors but the Pastry routing table has been strapped down, sacrificing scalability. The authors consider MANETs with up to 1000 nodes realistic but simulations have only been performed with 250 node networks and predicting the size of future MANETs is risky. In a related issue the simulation carried out to evaluate the effect of *handovers* resulting from clusters is not fair because a lookup rate was used that already caused congestion in the compared protocols. A closer evaluation might reveal scalability problems.
As presented in the previous section a large number of similar protocols have been suggested, most of them based on DHTs and all of them outperforming the common re-active or pro-active ad hoc routing protocols in provided simulation results. A realistic and unbiased comparison between the suggested protocols would provide valuable information

[11] Nonetheless an attempt of a theoretical proof is provided for Ekta [15]

about their applicability and true performance. Especially a comparison between simpler protocols such as VRR and more complex versions such as MADPastry or Ekta would benefit future development and enhancements. On top of that the number of implemented distributed applications will dictate how and where further research is needed, and realistic applications that are also implemented in MANETs are still rare. It remains however clear that to provide distributed applications in MANETs DHTs are capable of providing the required basis and MADPastry is a promising candidate showing the potential benefits and performance of exploiting the similarities in requirements and behavior of peer-to-peer networks and MANETs.

A DHTs,Pastry and AODV

Due to their great importance in recently developed peer-to-peer protocols and their direct relevance to MADPastry the appendix provides a short intuitive introduction to Distributed Hash Tables followed by an overview of Pastry and Ad-hoc On Demand Distance Vector Routing (AODV), the peer-to-peer protocol respectiveley ad hoc routing protocol MADPastry is based on.

A.1 Distributed Hash Tables

Distributed Hash Tables (DHTs) have first been proposed in [11] to provide an intelligent way of caching web sites to relieve hot spots on the Internet. In the context of peer-to-peer networks DHTs are used to map objects to corresponding nodes in a distributed fashion. To illustrate the motivation in peer-to-peer networks consider using standard hashing where buckets represent the nodes participating and each item or object is hashed using a hash function to one of the buckets. A node (represented by one or more buckets) is then responsible for the set of objects (items) mapped to it. This approach provides good balance in that each node will be responsible for about the same number of objects. However given the dynamics of peer-to-peer networks where nodes join and leave frequently a different hashing function would need to be chosen based on the number of nodes currently present, resulting in a complete and expensive rehashing. Objects would need to be exchanged between the nodes because the responsibility has changed due to the new hash function. This is of course a major drawback which is one of the reasons DHTs were proposed. Thus the three main goals and characteristics of a DHT as given in [11] can be categorized as follows (*Views* represent a subset of the buckets):

- **Monotony**: Adding or removing buckets (nodes) will not result in unnecessary movement of items (objects). When adding a bucket items are only moved to the new bucket to achieve even distribution, no transfers between two old buckets should be necessary.
- **Spread**: The total number of copies of each item (object) should be bounded.
- **Load**: The total number of items any bucket obtains (in all views) should not be substantially higher than the average.
- **Balance**: Considering a subset of the buckets (view V), a number of items and a hash function each bucket is assigned a fraction $O(1/|V|)$ of the items with high probability.

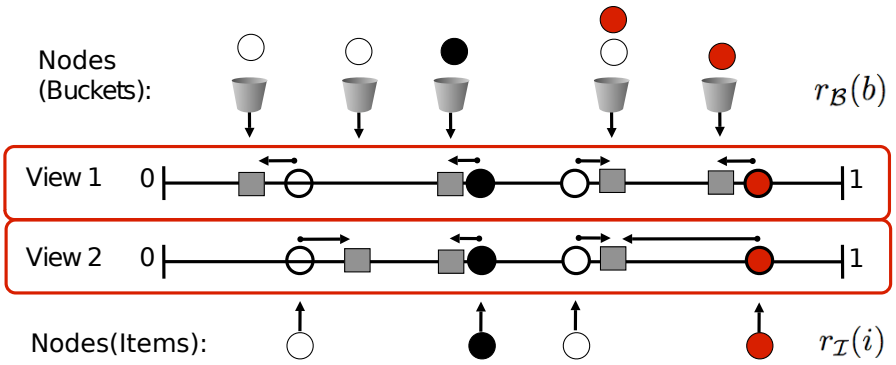

Figure 3: Distributed Hash Table after [20]

Because of these properties DHTs are often also referred to with *Consistent Hashing* (i.e. the choice of assigning a responsibility for each item is consistent when inserting a new bucket) [12].
An example is shown in Figure 3. Buckets (nodes) and items (objects) are hashed onto the reals $[0, 1]$ using two hash functions. $r_B(b)$ maps $k * \log C$ copies of each bucket b randomly into the interval (C: number of buckets) and $r_I(i)$ maps item i randomly into the same interval. An item is then assigned to a bucket it is closest to, i.e. the distance $|r_B(b) - r_I(i)|$ is minimal. The buckets are assigned the sum of all objects assigned to it in the individual views. It has been shown that above mentioned properties for Monotony, Balance, Load and Spread hold, e.g. in [11]. The details of implementation like the size of the interval that is mapped to or even the number of dimensions can be chosen differently but the underlaying idea remains. DHTs allow distributed applications to look up resources at the same time being scalable as well as fault-tolerant by design. Each node can independently map objects and nodes using the global hash functions and thus, given it knows its own neigbourhood in the hash space, knows who is responsible for which object or which other node might know who is responsible. Examples of peer-to-peer networks based on DHTs besides Pastry are e.g. CAN [16], Tapestry [24] or Chord [21].

A.2 Pastry

Pastry is a peer-to-peer network providing routing and object location services designed for distributed applications on the Internet. It was developed by Antony Rowstron and Peter Druschel in Cambridge at Microsoft [18]. The goal of Pastry is to provide a de-

centralized, fault-tolerant, scalable peer-to-peer-network that is adapted to the underlay network for optimization of latency [12]. It is important to realize that application level routing and message delivery takes place in an overlay, a virtual network constructed on top of the physically existing one, in which only peer-to-peer members participate [12]. This does however not replace the physical routing in the underlay required between hops in the overlay. In fact an overlay hop in Pastry will usually consist of several hops in the network below. A number of applications has been based on Pastry e.g. Scribe [19] for event-notification or Past [17], an archival storage facility. In this section we will concentrate on how message delivery in Pastry works followed by some properties that were proven theoretically and practically.

A.2.1 Routing in Pastry

Pastry, like most recently developed peer-to-peer networks, is based on Distributed Hash Tables (DHTs) where each node and object is assigned a unique 128-bit long ID. The key space for Pastry is one-dimensional, each ID is interpreted as a number to the base $B = 2^b, b \in \mathbb{N}$, and usually $b = 4$ is selected which results in hexadecimal interpretation of IDs. A node is then responsible for an object if its ID is numerically closest to an object's ID, which may be computed using a cryptographic hash-function of e.g. the file name. The basic idea is, that if a node knows what object ID it is looking for, it can send a message using Pastry to the node responsible for it. The node responsible will have to answer according to the request by e.g. serving a file or responding with the current address of the file. This is however not part of Pastry but part of the application implementation on top of Pastry. Pastry is only considered with delivering the message in the first place. Next an overview of the data structures maintained by each node is given.

Routing Data Structures Each node p maintains three sets of connections (note that each entry in a set associates a node ID with e.g. an IP address in the underlay):

- **Routing table** R: A table in which each row n (starting at 0) consists of entries with node IDs that share the first n digits with the ID of p followed by every possible next digit (remember that we interpret the IDs to base B and thus have $B = 2^b$ columns). Thus each node knows for each of its own prefix z and digit $j \in B$ another peer whose ID starts with z followed by j [12]. If no peer with the required prefix exists or is known the entry remains empty, and if several peers are possible candidates, the one with lowest latency (e.g. round trip time) is chosen.

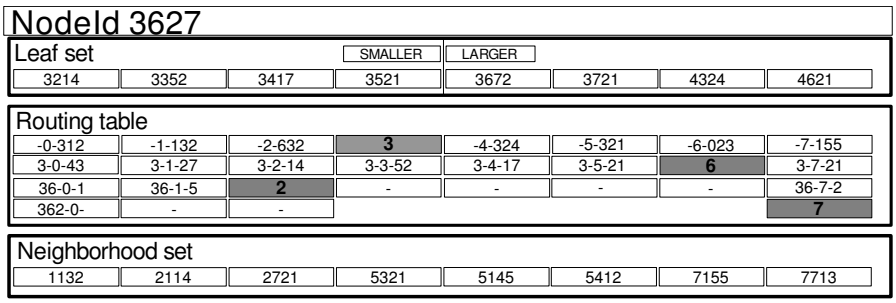

Figure 4: Routing table of a node with ID 3627 where $b = 3$ and $|L| = 8$ after [18]. Grey cells represent entries the node itself occupies.

- **Leaf set**L: The leaf set consists of nodes with the next $|L|/2$ numerically higher and lower IDs. Ultimately this forms a ring containing all nodes. Usually $|L| = 16$ is selected.
- **Neighbourhood set M**: The neighborhood set M consists of usually 2^b arbitrary nodes which are closest to p w.r.t. latency. This set is not used in the standard routing process but in problem scenarios.

Figure 4 shows an example of each of the three sets and Figure 5 provides a graphical representation.

Routing Process We next consider the routing algorithm in detail. Pastry's routing process is based on the Plaxton routing mechanism. To deliver a message responsible for a data object we compute the object's ID and use it as *target ID*. The message is then *indirectly* routed to the node responsible for it i.e. the one whose overlay ID is numerically closest to the target ID without explicit knowledge of the final target like e.g. its IP address.
Assume a node p is executing the routing algorithm for a message M. If the target ID of M is within the bounds of the leaf set, the message can directly be delivered to the node with numerically closest ID, possibly itself: then p was the target. Otherwise the message is sent to a node p' from the routing table R whose matching prefix length to the target ID is at least one digit longer than the matching prefix length of p with the target ID. If no such node exists in the routing table (remember that the routing table is not required to be complete) the node sends the message to a node with same matching prefix length but closer to the the target ID. Such a node exists in any case in the leaf

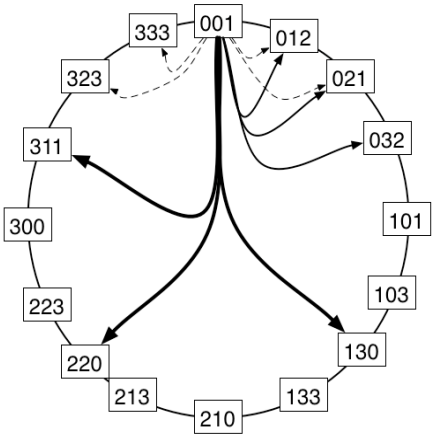

Figure 5: Graphical representation of a Pastry routing table from [20] where $b = 2$ and $|L| = 4$. Bold arrows represent entries from the routing table R, dashed arrows the leaf set entries in L.

set, except the improbable case that $|L|/2$ nodes fail at the same time. The same steps are executed at each node p' until the message arrives at the correct node [12].
From this routing algorithm directly follows that it always converges: in each step either the matching prefix length with the target ID is increased by one, or the message is forwarded to a node numerically closer [12].
Figure 6 shows an example routing process where node 3627 sends a message to 6257 which gets routed to the numerically closest node: 6367. With each overlay hop the matching prefix length is increased by one, using the nodes' routing table entries. Note that although the distance for a hop in the overlay is small, a large number of physical hops can be necessary.

A.2.2 Joining Nodes and Error Situations

Up to now we only considered a static Pastry network but of course nodes join and leave frequently. Pastry employs an algorithm where to join the network a node selects a close peer w.r.t. latency and initiates a search for its own ID. The node then copies routing entries from each of the nodes visited while the message traverses the network. This

[12] Note that Pastry is only concerned with indirectly delivering the message to the responsible node. How the message is processed and what subsequent actions are taken is part of the application implementation

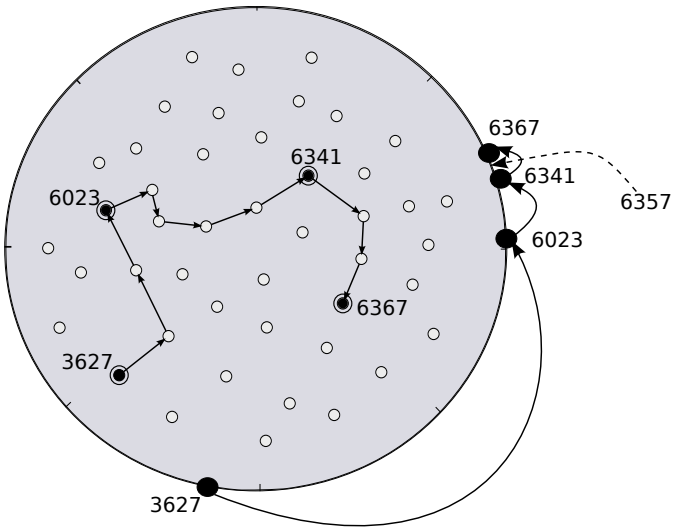

Figure 6: Routing with Pastry. Node 3627 is sending a message to 6257 which gets routed to the numerically closest node: 6367. The outer circle represents the overlay hops while inside the physical hops performed are shown.

process is described in detail in [18]. It also notifies nodes that require knowledge about the presence of the new node. Of course certain error conditions can occur like e.g. stale or empty routing table entries. It is important to realize that as long as the leaf sets are consistent, correct routing is guaranteed. If routing tables entries are missing, nodes periodically contact overlay peers to request entries. This benefits performance but is not required for correct routing.

A.2.3 Properties of Pastry, Locality

The routing procedure of Pastry is designed to be scalable which has also been theoretically proven. Given correct routing table entries it can be shown that Pastry routes a message in $O(log_{2^b} n)$ expected steps for a network of n nodes. The worst case can be shown to be $O(n/l)$ steps if only the leaf set is used for routing. Other important factors like the number of messages required for a join also scale well for large networks. The proofs are e.g. presented in [12] which is why we refrain from doing so here. These important properties have also been confirmed in practical experiments [18]. During construction of routing tables locality is always considered implicitly by choosing a low latency peer if possible. One of the interesting properties is that the last hops in the

routing process are the most expensive ones. This is mainly based on the fact that in each step, with increasing prefix match, the number of nodes the message could be routed to exponentially decreases, which also limits the chances of selecting a low latency next hop.

A.3 MADPastry Packet

According to [22] a MADPastry packet contains the following information:

- The AODV sequence number of the packet's source
- The AODV sequence number of the packet's previous physical hop
- The overlay id of the packet's *source*[13]
- The overlay id of the packet's previous physical hop

Additionally the packet must of course contain the overlay ID of its final destination (e.g. the ID of an object generated by hashing) as well as the next overlay hop ID.
It is obvious that a MADPastry packet is substantially larger than a broadcast or traditional Pastry packet because of the additional information carried. According to [22] the packet is around four times larger than a Gnutella-agent broadcast packet.

A.4 Ad-Hoc On Demand Distance Vector Routing (AODV)

MADPastry relies on the Ad-Hoc On Demand Distance Vector Routing (AODV) protocol to forward messages in the underlay. This section provides a short description of the principles AODV applies, a detailed description in form of an RFC is available [14]. AODV is a lightweight, re-active and quickly adapting routing protocol for ad hoc networks that only maintains routing information for connections between nodes that are communicating. To overcome the classic problems of distance vector routing protocols like *routing loops* or *count-to-infinity* AODV uses sequence numbers. We first look at some principles followed by the process of routing setup and some information about routing table maintenance.

[13] The paper [22] does accidentally not mention "source" which is clearly an error. By context it can be assumed that the packet's "source" overlay ID is meant.

A.4.1 Routing Principle

To guarantee loop freedom AODV applies sequence numbers. Each node maintains a monotonically increasing sequence number which is used to identify the freshness of routes. The sequence number is increased whenever a node issues a Route Request (RREQ) and in some cases when it responds with a Route Reply (RREP). Whenever a node has the choice between two routes it chooses the one with the greater destination sequence number as it is probably the newest. Routes are only established when communication is required[14] and maintained as long as the link is still active, i.e. data is transmitted. The basic routing procedure consists of a Route Request message that is broadcast from the source through the network to the destination and a Route Reply message from the destination that is unicast back to the source. All messages defined by AODV are based on UDP to keep the overhead minimal. As a supporting data structure the AODV routing table consists of the destination's physical address, the next hop towards the destination, a sequence number for each route, and the length of each route.

A.4.2 Route Request (RREQ)

To establish communication to a node no route is known or valid to, a node broadcasts a Route Request (RREQ) message to its neighbors. The message contains the last known sequence number of the destination as well as its own and of course the destination address a route is requested for. Nodes receiving the RREQ increase the hop count and retransmit the message ignoring duplicate RREQs received. Each node receiving a RREQ stores a route to the originator of the RREQ via the intermediate node it received the message from. This is known as the *reverse-path*, a path towards the originator of the RREQ. Dissemination control in form of a Time-To-Live field allows strategies like expanding ring searches to reduce the traffic generated.
Figure 7 shows an example of a *reverse-path* path being set up. A RREQ is sent by the white node to discover a route to the black node. The RREQ is broadcast by every node receiving it and the *reverse-path* is set up by storing a route towards the source. Note that for simplicity only RREQs and route entries are shown that lie on the path to the destination, the other nodes will forward the RREQ and store routes as well but those will time out as no RREP is sent along that way.

[14] which is the definition of a re-active routing protocol

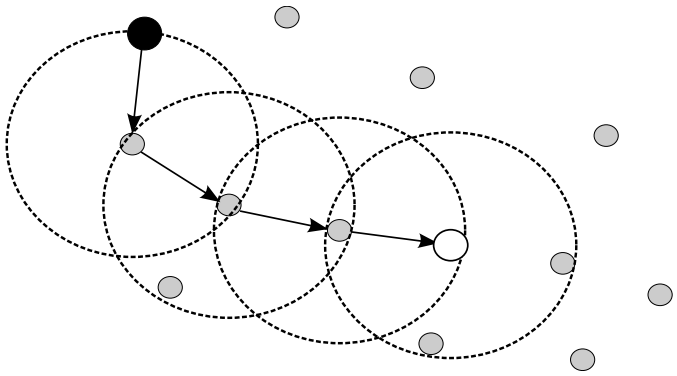

Figure 7: A RREQ is sent by the white node to discover a route to the black node. The route entries generated for the *reverse-path* are represented by the arrows.

A.4.3 Route Reply (RREP)

A node receiving a RREQ can reply using a Route Reply (RREP) message if it is an intermediate node and has a fresh enough route (according to the destination sequence number) to the destination, or if it is the destination itself. The RREP message is sent back along the already set up *reverse-path*, generating the *forward-path*. Each node forwarding the RREP stores a route towards the originator of the RREP message via the intermediate node it received the RREP from. If an intermediate node can answer to a RREQ with a RREP it can also send a RREP to the destination to ensure a bi-directional communication path is set up. Figure 8 continues the example by showing the *forward-path* being set up. The RREP is unicast back along the already set up *reverse-path*. The nodes receiving the RREP stores a route towards the destination through the node it received the RREP from.

A.4.4 Maintenance

Besides the already discussed RREQ and RREP message AODV also defines periodic Hello messages to inform neighbors of a node's presence as well as error notifications (RERR). Nodes maintain the state of a route keeping track of the nodes using it. If a route breaks affected nodes are notified using a RERR message which is then propagated to other nodes if necessary. This ensures fast recovery in error situations. Routes that are unused are deleted after a certain time to avoid keeping track of unused and probably stale information.

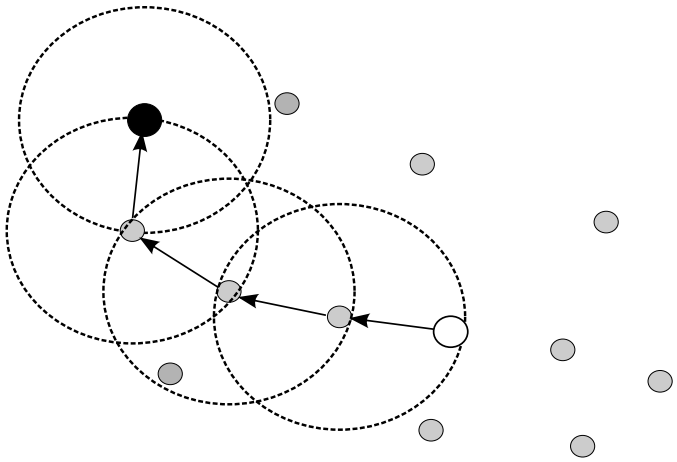

Figure 8: A RREQ arrived at the black node and it responds with a RREP along the route to the white node. The routes entries generated for the *forward-path* are represented by the arrows.

References

[1] Matthew Caesar, Miguel Castro, Edmund B. Nightingale, Greg O'Shea, and Antony Rowstron. Virtual ring routing: network routing inspired by dhts. *SIGCOMM Comput. Commun. Rev.*, 36(4):351–362, 2006.

[2] Marcello Caleffi. Mobile ad hoc networks: The dht paradigm. In *PERCOM '09: Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–2, Washington, DC, USA, 2009. IEEE Computer Society.

[3] Marcello Caleffi, Giancarlo Ferraiuolo, and Luigi Paura. Augmented tree-based routing protocol for scalable ad hoc networks. *CoRR*, abs/0711.3099, 2007.

[4] T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr). RFC 3626, 2003.

[5] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations, 1999.

[6] Curt Cramer and Thomas Fuhrmann. Performance evaluation of chord in mobile ad hoc networks. In *MobiShare '06: Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, pages 48–53, New York, NY, USA, 2006. ACM.

[7] F. Delmastro. From pastry to crossroad: Cross-layer ring overlay for ad hoc networks. In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pages 60–64, March 2005.

[8] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 114–128, New York, NY, USA, 2004. ACM.

[9] A. Gopalan and T. Znati. Peernet: a peer-to-peer framework for service and application deployment in manets. In *Wireless Pervasive Computing, 2006 1st International Symposium on*, pages 6 pp.–, Jan. 2006.

[10] Y. Charlie Hu, Saumitra M. Das, and Himabindu Pucha. Exploiting the synergy between peer-to-peer and mobile ad hoc networks. In *HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems*, pages 7–7, Berkeley, CA, USA, 2003. USENIX Association.

[11] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM.

[12] Peter Mahlmann and Christian Schindelhauer. *Peer-to-Peer Netzwerke*. Springer-Verlag, 2007.

[13] Grant P. Millar, Tipu A. Ramrekha, and Christos Politis. A peer-to-peer overlay approach for emergency mobile ad hoc network based multimedia communications. In *Mobimedia '09: Proceedings of the 5th International ICST Mobile Multimedia Communications Conference*, pages 1–5, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[14] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[15] Himabindu Pucha, Saumitra M. Das, and Y. Charlie Hu. Ekta: An efficient dht substrate for distributed applications in mobile ad hoc networks. In *WMCSA '04: Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications*, pages 163–173, Washington, DC, USA, 2004. IEEE Computer Society.

[16] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[17] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 188–201, New York, NY, USA, 2001. ACM.

[18] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[19] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*, pages 30–43, London, UK, 2001. Springer-Verlag.

[20] Christian Schindelhauer. Lecture algorithms and methods for distributed storage, Wintersemester 2008/2009.

[21] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[22] Thomas Zahn and Jochen Schiller. MADPastry: A DHT Substrate for Practicably Sized MANETs. In *Proc. of the 5th Workshop on Applications and Services in Wireless Networks (ASWN2005)*, 2005.

[23] Thomas Zahn and Jochen Schiller. DHT-based Unicast for Mobile Ad Hoc Networks. In *Proceedings of the 4th IEEE PerCom Workshops. 3rd IEEE International Workshop on Mobile Peer-to-Peer Computing (MP2P'06)*, Washington, DC, USA, 2006. IEEE Computer Society.

[24] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and. Technical report, Berkeley, CA, USA, 2001.