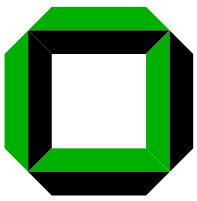


Universität Karlsruhe (TH)

Fakultät für Informatik
Institut für Betriebs- und Dialogsysteme (IBDS)
Lehrstuhl Systemarchitektur

Mobility Enhancements to an Approach for Structured Overlay Routing in Wireless Mobile Ad Hoc Networks

Diploma Thesis

Marcel Dischinger

24. November 2005

Advisors: Prof. Dr. Frank Bellosa
Dipl.-Inform. Curt Cramer

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Literaturhilfsmittel verwendet zu haben.

Karlsruhe, den 24. November 2005

_____

Marcel Dischinger

Contents

Chapter 1

Introduction

Devices like laptops, mobile phones and personal digital assistants (PDAs) nowadays form an integral part of our daily life. Companies like AOL or Intel propagate the idea of mobile offices where one is not bound to the local desk but can work almost everywhere. Recent development to combine PDAs and mobile phones to so-called smart-phones look very promising. In fact, progress in miniaturization and low-power architectures make small and long-running devices affordable to almost everyone. And the popularity will even grow within the next years. Recently, even Daimler-Chrysler announced to equip their cars with radio interfaces so that they can communicate with each other.

Wireless and so mobile ad hoc networks (MANETs) gain a growing popularity in the research community and probably also in commercial applications in the near future. All kind of household equipment or small multimedia devices could benefit from such a network which does not need an existing infrastructure or complex user configuration because the network is established fully self-organized and decentralized.

As possible applications for this environment, peer-to-peer (P2P) systems seem to fit very well. P2P networks are a building block for distributed applications like file sharing, instant messaging or distributed computing. So far, there are mainly approaches to run P2P systems developed for the wired Internet on top of an ad hoc routing protocol. But this does not seem to perform well.

In this thesis we present a novel cross-layer approach that combines both, an efficient and scalable ad hoc routing protocol and a P2P system. This combination can greatly help in reducing the message overhead and leave more of the limited MANET bandwidth to the applications using the network.

1.1 Mobile Ad Hoc Network - MANET

In contrast to wireless local area networks (WLANs), MANETs do not need a fixed infrastructure by access points which act as routers and gateways to other networks. At the moment access points cannot be found everywhere and also for some applications (like mesh or sensor networks) central components for communication are not appropriate. Especially in areas with spare or without any infrastructure – like rural areas, areas after a natural disaster or battle grounds – there is the need for networks which do not rely on infrastructure but establish their own one. And with the rapid increase in popularity of small handheld devices like PDAs or smart-phones there is also the

need for easy network connections from device to device or between a few devices in a certain area without implying any external infrastructure (which might not be available anyway).

In a MANET the network is not managed by an access point but by the nodes themselves. Every participating node acts as a router and forwards data for other nodes. MANETs impose several restrictions which are not known in wired networks.

First, the bandwidth is very limited, so a focus must be to limit the routing protocol overhead. Second, due to the mobility the network is highly dynamic, nodes appear, move and disappear, links break with a high frequency. Because of this, standard internet routing protocols like RIP or OSPF which do a good work in wired networks perform poorly in MANET environments. New routing protocols have to be developed to meet the new requirements of mobility and limited bandwidth.

We can distinguish mainly three different types of ad hoc routing protocols: *Proactive* and *reactive protocols* and *coordinate-based routing* (sometimes also called *landmark routing*).

Proactive routing protocols like DSDV [22] and WRP [19] are similar to those used in the wired internet. They actively maintain routes between all pairs of nodes and so produce a significant protocol overhead. Because of this they scale bad with the number of nodes, especially in very dynamic networks.

Reactive routing protocols like DSR [18] and AODV [23] maintain routes on demand – when they are actually used. By this, they scale a lot better than proactive protocols, but still have issues, as in particular DSR and AODV employ flooding for route discovery. In Section 2.3 and 2.4 we take a closer look on these two popular ad hoc routing protocols.

Coordinate-based protocols like $L+$ [5] and *Safari* [11] rely on the physical location of a node in the topology. Note, that this is different from approaches that use the global position – e.g. given by the global positioning system (GPS) – for addressing. They integrate this location into the addressing scheme to improve routing and to result in short routing paths. Although this approach seems to scale pretty well, it suffers additional effort to translated fixed addresses into location-dependent ones and this translation also causes additional routing delay.

Especially for the usage of a P2P system in ad hoc networks, just running one on top of an existing routing protocol is not satisfying. As the bandwidth is very limited in MANETs, duplication of the routing functionality in the routing protocol and the P2P system is not acceptable.

Therefore a new architecture called SONAR (*Structured Overlay Networks for Ad Hoc Routing*) combines both worlds and acts as a cross layer approach. The used P2P system structures the network and can do the routing without any underlying routing protocol. At the same time SONAR is capable to act as a full P2P system to offer standard P2P functionality to applications. Proactive mechanisms are accepted to keep up the P2P structure, but the overhead for maintenance stays low as I show in the following. The P2P structure is also used to route towards the destination and so explicit route discoveries are avoided and the routing paths stay reasonably short.

The *Iterative Successor Pointer Rewiring Protocol* (ISPRP) [7] is the basis of SONAR. It is used to initialize the P2P structure message-efficient and is described in detail in Section 2.1. So far, ISPRP does not support mobile and dynamic environments. The goal of this thesis is to enhance ISPRP with mechanism to meet the new requirements introduced by mobility and limited bandwidth of MANETs. This means to find and evaluate mainly efficient structure maintenance and local link repair mechanisms. The implementation was evaluated in the network simulator GloMoSim [13].

1.2 Document Outline

The next chapters are organized as follows: First, we present related work. Then we analyze the task this thesis deals with and point out the problems and challenges. In chapter 4 we focus on the design of our solution and work. Chapter 5 gives a brief insight of the implementation in the network simulator GloMoSim. In chapter 6 we discuss the results of the simulations while the last chapter concludes with an outlook on future work.

Chapter 2

Background and related work

In this chapter we present the background and related work of this thesis. First, we give an introduction to ISPRP which is the basis for this work. As we mentioned in chapter 1, ISPRP is not ready for usage in mobile and dynamic environments. Section 2.2 gives an overview of mobility models used for simulation. After that we describe the two very popular ad hoc routing protocols DSR and AODV and outline the concept of coordinate-based routing. Furthermore, the P2P system Chord is presented, as ISPRP borrows its structure from it. Finally we give a brief overview of approaches for P2P in MANET environments.

2.1 Iterative Successor Pointer Rewiring Protocol

While most developed P2P systems assume node connectivity to be provided by an underlying network-layer routing protocol, this is not appropriate in many scenarios. Especially in networks with nodes with limited capabilities a cross-layer approach can greatly help in reducing the message overhead. We focus here on MANETs in particular.

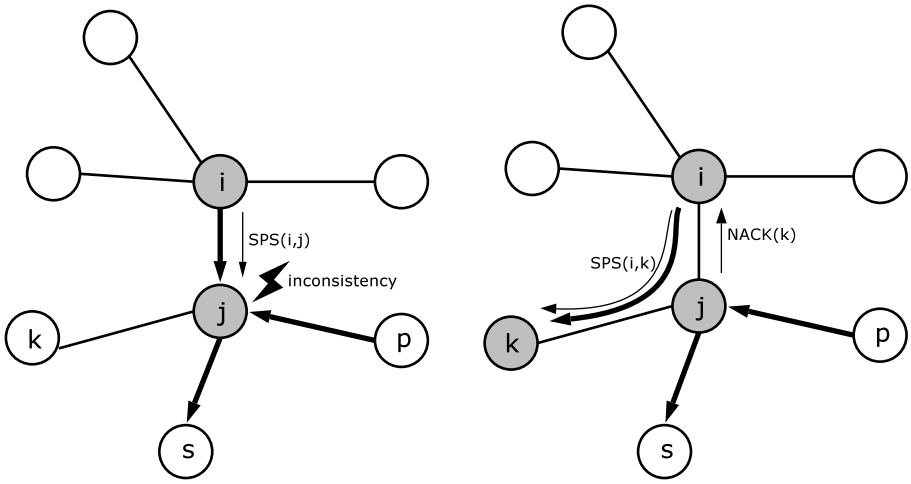
2.1.1 P2P Structure Initialization

Without an underlying routing layer nodes only have link-layer connectivity and the problem arises how to initialize the P2P network.

Some use a bootstrap process where a joining node needs to know a node from the network and connects to the rest by this one. This is not appropriate for a fully decentralized and self-organized network (and in this case without an underlying routing protocol that would additionally provide the path to the bootstrap node).

The second approach is to employ flooding. A joining node floods the network to introduce itself. After the whole network learned about the new node, it is integrated into the P2P structure. It is obvious that this approach is very bandwidth-intense and sub-optimal in MANET environments.

ISPRP (Iterative Successor Pointer Rewiring Protocol) [7] initializes a P2P routing network efficiently among a freshly deployed set of nodes having but link-layer connectivity. It is fully self-organized and issues only a small per-node amount of messages by keeping interactions between nodes as local as possible. It requires no general flooding to establish the P2P structure – for ISPRP, a ring as in Chord is used.



(a) A new joining node i chooses the best known successor j from its routing cache and sends an SPS message to it. But j already is the successor of p which results in an inconsistency.

(b) j calculates, that it still is the successor of p and suggests, that k would be a better successor for i.It includes a path to k in the NACK message. Node i then sends an SPS message to k and so on until it receives an ACK.

Figure 2.1: ISPRP algorithm

2.1.2 Algorithm

When a node i joins the network, it first has to get to know its physical neighbors. This can be done by sending a special HELLO-message as an one-hop broadcast. Nodes in range reply with a HELLO-message by themselves which may also include useful information of the nodes neighborhood (details about this issues can be found in Section 4.2.1). From this information the node chooses a suitable successor – the node closest to itself in the circular identifier space but with a higher identifier as can be seen in Figure 2.1(a).

A *Successor Pointer Solicitation* (SPS) message is sent informing the possible successor j about this choice. Node j then checks if this choice – corresponding to its knowledge of its own neighborhood and comparing to its already existing predecessor – is correct. In this example j already is successor of node p, an inconsistency occurs. Node j now calculates which one of both is its real predecessor – in this case still p. As can be seen in Figure 2.1(b), j sends back a NACK-message including the path to node k which is a more suitable successor for i according to j's routing cache. i then goes on to send an SPS-message to k and the whole process is repeated iteratively until the network is stabilized and the ring structure up.

The structure formed by ISPRP might not always be a ring. In fact, as a result of wrapping the address space around at its border nodes could form two independent rings. This happens if the network can be partitioned by breaking only a few links; this global inconsistency cannot be detected by ISPRP as described above. In each ring, there is exactly one node whose successor pointer crosses the wrap-around point of the address space. Thus, each of those nodes has to flood the network to detect such

an inconsistency. So paths between them are found and the rings are joined, leading to a globally consistent state. This has to be done periodically because we cannot decide locally whether the network has already converged. Note that this is significantly different from each node having to flood the network, as the total additional per-node message amount equals the number of independent rings formed.

ISPRP can be implemented in either a recursive or an iterative fashion. Reference [7] describes a recursive version whose early design is only suitable for reliable networks. SONAR is based on an iterative version. We give a brief overview of the original, recursive version to highlight its differences in the following.

A joining node i sends an SPS message to a potential successor k. If this successor knows a better successor j for node i, it will send back a Successor Rewiring Solicitation (SRS) message, thus node i can adjust its successor pointer, but will forward the SPS message to j. This is done recursively until the (locally) correct successor is reached.

In unreliable networks we have to rely on timeouts to detect packet loss. Thus SRS messages could be used to reset the timer of the original sender. However, in the recursive version of ISPRP a node assumes its successor pointer to be correct as long as it receives no SRS message. As the SPS message could get lost, this cannot be assumed in unreliable networks. To be able to use a timeout mechanism, the protocol has to be extended with ACK messages as in the iterative version.

Another problem is, that if an SRS message gets lost, node i will send a new SPS message thus resulting in multiple SPS messages from the same sender concurrently traveling through the network. Furthermore, the successor paths grows with every recursion and can get very long, and optimizing it is complicated. This also makes it hard to choose good timeout values, as the sender is not aware of the actual successor path length.

Before we go on describing the functionality of existing ad hoc routing protocols and how they handle node mobility, we give an introduction how mobility can be modeled for simulations.

2.2 Mobility Models

Simulating MANETs does not only afford simulation of the traffic but also of the mobility of the nodes. It is not easy to model mobility. People walking around or cars driving by include several restrictions in their movements like following a street or sitting on benches. Two simple although very popular approaches for mobility models are the *random waypoint model* [3] and the *random direction model* [28].

New approaches try to deal with real world movements by introducing frameworks to include several different mobility characteristics into one model for simulation. But this research is just at the beginning and these models have not been verified yet.

2.2.1 Random Waypoint Model

One of the most popular mobility model is the random waypoint model. Here, a node randomly chooses a destination point in the system area (the waypoint), moves with constant speed v (which is chosen between $[v_{min}, v_{max}]$) on a straight line to this point and then pauses for a certain time before it again chooses a new destination.

A node is completely described by its current space vector ($x(t), y(t)$), its current speed $v(t)$ and its current destination point ($x_d(t), y_d(t)$).

2.2.2 Random Direction Model

In the random direction model, a node randomly chooses a direction φ (taken from a uniform distribution on the interval $[0...2\pi[$). It moves with constant speed v (chosen between $[v_{min}, v_{max}]$) in this direction for a certain time before it again chooses a new direction.

A node is completely described by its current space vector ($x(t), y(t)$), its current speed $v(t)$ and its current direction $\varphi(t)$.

In contrast to the random waypoint model, the random direction model may try to leave the system area. Thus, a "border rule" has to be defined what to do at these occasions. In the literature three basic principles are found:

1. The leaving node is bounced back to the system area, according to a certain rule – e.g. if the last direction was φ_t, the resulting direction could be ($\pi - \varphi_t$).

2. The leaving node is deleted and a new node is created at an arbitrary point inside the system area.

3. The leaving node is wrapped around to the other side of the simulation plane.

Several variations of this algorithm are known which are based on this general behavior.

2.2.3 Discussion

The problem with the random waypoint model is that nodes are not uniformly distributed over the simulation area [2] but nodes are more likely to be in the middle of it. The random direction model on the other hand shows a more uniform distribution with less preference for the center.

In the simulations we used the random direction model with border rule 1. It gives us more realistic conditions than a random waypoint model in terms of node distribution. However, we are still far away from having perfect modelling which produces results that represent real-life behavior.

There are already existing approaches for ad hoc routing protocols. In the following we present the two popular protocols DSR and AODV. After that, we give an overview of locality aware approaches without going into detail of a particular implementation.

2.3 Dynamic Source Routing - DSR

As mentioned above, DSR [18] is a reactive routing protocol. This means routes are not actively maintained except when needed. As its name suggests, DSR uses source routing which means that intermediate nodes do not need to keep state. Each message carries a source route header which includes the whole path towards the destination and so identifies the next hop for each intermediate node explicitly.

2.3.1 Route Discovery and Routing Cache

In DSR, each node maintains its own routing table. As the routing information is not actively maintained the table is more like a cache where all learned paths are stored or replaced if a somehow better one was found or a route is found to be invalid. Initially, the authors of DSR did not define a specific cache structure to be used. As a feature, the cache can be implemented adjusted to the special need of the environment DSR is used in. However, in reference [18] a two-layer cache is presented which has been shown to perform well in most situations.

Active paths can be found in the *primary cache*, while not yet used paths stay in the *secondary cache*. In order to route a message to another node the primary cache is searched for a path to the destination (the route might also be a part of another path stored there). If a path is found this one will be used to route the message and is therefore inserted in the source route header. Otherwise DSR looks up a route in its secondary cache. If one is found the path will be promoted to the primary cache and the message is sent using this route.

If no route could be found at all, DSR will start a route discovery. A route request message is flooded to the network and the source path towards the destination is recorded in it. At the destination a route response message is sent back to the original source of the request including the full source path. As there might be disjoint paths to the destination, all of them are sent back and saved by the source in its routing cache to have alternate routes if paths break. While the currently used path is saved in the primary cache, the alternate paths go into the secondary one.

The two-layer cache design prevent opportunistic routes from competing for cache space with routes of known value to the node. Note that DSR has no mechanism to discard old and therefore likely to be no-longer valid routes. Route removal only depends on the cache replacement strategy employed.

2.3.2 Route Maintenance

Ad hoc routing protocols have to deal with node mobility which changes the network topology. Thus, links break and new links appear on its way as a node moves around. But links can also break temporarily due to congestion because the bandwidth in MANETs is still very limited.

In its original specification DSR employs only one repair mechanism. If an intermediate node detects a link failure (by a timeout), a route error message will be sent back to the source which then chooses an alternative route from its routing cache or – if none was found or all failed already – issues a new route discovery. Intermediate nodes will also remove the failed route from their caches when the route error message passes by.

Additionally, a local repair mechanism was introduced. When an intermediate node detects a link failure it searches its own routing cache for an alternative route to the same destination. If one exists, it will replace the source route with the route from its cache and retransmit the package. Otherwise the packet will be dropped. In both cases a route error message is sent back to the source which will then choose a new source path from its own cache. This mechanism is called *salvaging*.

To get to know more alternative paths to a destination a node can overhear messages sent in its neighborhood but not dedicated for itself by using the promiscuous mode of the network interface and save the thus learned source paths. Of course, this results in increased battery consumption of the device as it cannot enter sleep mode to save

battery power while there is no message for itself [12]. Obviously, this approach can put even more stale routes into the routing cache, too.

2.3.3 Discussion

DSR does not use periodic messages to maintain its routing caches because it is a reactive routing protocol, but produces initial discovery delay and overhead for flooding the discovery message. Furthermore, multiple answers are sent back to the source increasing the overhead further [9]. But this is compensated a little as this prevents further route discovery runs as long as alternative paths are available. Each node stores its own paths and also alternatives that might be used if a failure occurs. Of course, the mandatory source route headers increase the size of each data package and so also increase the overall network overhead [17].

As the DSR routing cache does not know any mechanisms to prune stale routes after an expiration time (and not caused by a route error message or by the cache replacement strategy), they could be used as alternative paths although they might be already invalid [16]. This can grow even worse when using message overhearing.

Link breaks are only reported back to the original sender of a message. Other nodes also using this a broken link cannot benefit from this new information and have first to send a message (which will of course fail) in order to prune the failed route in their cache and switch to another one.

2.4 Ad-hoc On-Demand Distance Vector Routing - AODV

Like DSR, AODV [23] is a reactive routing protocol. It is standardized by the IETF [21]. Unlike DSR no source routes are used. As AODV's name suggests, distance vector routing mechanisms are used.

2.4.1 Route Discovery and Routing Cache

A path discovery process is initiated whenever a source node needs to communicate with another node for which it has no routing information in its routing table yet. A route request is sent to the source node's neighbors and sent on by them until the message eventually reaches the destination or a node which knows a current path to the destination. The message contains a hop counter to measure the length of a specific path besides fields for the source and destination address. There are also two sequence numbers, the source sequence number and the last destination sequence number known to the source which are used to guarantee route freshness and to prevent routing loops.

As AODV uses no source routes and a node does not use proactive mechanisms to build up its routing table to get routing information towards arbitrary destinations, the discovery also has the purpose to establish the path in every intermediate node. Therefore the path discovery process can be divided into two parts, the *reverse path setup* and the *forward path setup*. By this, a bidirectional path identified by the source and destination address of a connection is established (note that bidirectional links are assumed). As AODV's name suggests, instead of storing the whole path each node saves only the next hop towards source and destination respectively to be able to forward messages

In detail, the route discovery happens as follows: When a node receives a route request message it saves a reverse path entry including source and destination address,

the source sequence number and an expiration time when to discard this entry. Also the address of the last hop of the message is saved as a reverse path pointer to find back the route to the source node. The source sequence number is used to maintain freshness information about the reverse path, the entry to a specific source will only be overwritten or updated when the sequence number is greater than the former entry.

Then the message is rebroadcasted until it eventually reaches the destination or a node that knows a path to the destination from its routing table that is current enough to be used to answer this request. Entries with a destination sequence number higher or equal than the one provided in the request are acceptable while a lower destination sequence number disallows the usage of a table entry and forces the intermediate node to rebroadcast the request instead of replying back the existing entry. As there might be alternative paths, only the first one is sent back, but also later arriving request that establishes a shorter path than any one before.

The destination node or a node that knows a current path to it sends back a route reply message using the reverse path established by the first part of the discovery procedure. Each intermediate node saves the last hop of the reply message as the forward pointer for the forward path. The destination sequence number carried in the response message is also saved to maintain the forward route freshness. This is the same procedure as for the reverse path.

The reverse path is discarded if within the according expiration time no reply message for the source of this entry passes the node (and so this node is no part of a path from the source node).

2.4.2 Route Maintenance

Originally, AODV uses no specific repair mechanism. When a link breakage was detected (either by the source or any intermediate node), a route error message is sent back the path, not only for the message that failed to be forwarded but also for every path that uses this very link. Each node maintains a so-called precursor list for this purpose where all predecessors can be found that use a certain link. Thus, every route using this broken link is erased at once. The predecessor lists are used to contain the extent of flooding the RERR message. After that a new route discovery is issued by the source node, as it does not know an alternative path to the destination. Furthermore a new path between the two nodes has to be established again.

Furthermore a repair technique was presented in reference [1] using limited broadcasts to discover new routes or part of routes. After a path breakage and when a node runs out of alternative paths, a limited (by the TTL field in the IP header) broadcast is sent out as a new route discovery with the range increased by one compared to the former broken path. Thus, only a part of the network is flooded. If no new path is found, the range will be further increased until a path is found or the whole network is flooded. This mechanism is called *expanding ring search*. This method can also be used for local repairs by intermediate nodes, setting the range of the broadcast message to two in order to find a bypass for the broken link.

This mechanism focuses especially on the two major causes for link breaks in MANETs: Mobility and congestion. A node that moved out of range is very likely to be still in two-hop range. Also failures caused by temporary congestion can be detected and thus avoid unnecessary repair effort. However, this mechanism itself can cause congestion if the local traffic is already high.

Like DSR, AODV can use overhearing (listening to packets not dedicated to a node through the promiscuous mode of the network interface) to update routing table entries

to prevent entries to expire if the route was not used for a while.

2.4.3 Discussion

AODV is a reactive routing protocol and so only maintains routes when they are actually needed. By this, periodical maintenance is avoided accepting flooding to discover a path from a source to a destination.

The AODV routing mechanism needs every node so store soft state information about the paths a node is a member of. As only one path is stored per route, a route error forces a new route discovery and so produces significant overhead per failure [9] and can cause network congestion.

Route error messages are delivered to every node that uses a path containing the broken link. Thus, failed routes are pruned fast and using invalid links is avoided if they have failed for another route before.

2.5 Landmark Routing

Coordinate-based protocols focus on node location and thus also on node mobility. The current location is included in addresses which acts as routing hints and so results in a hierarchical address scheme. Note, that the used coordinates do not necessarily correlate with the global physical position but with the position relative to a node's position in the network topology.

As in *L+* [5], each node has two addresses: A unique node ID and a location-based address. For the location based address, a distributed lookup service offers a ID-to-address translation. This is additional effort and overhead for routing, but the location focused routing scheme does not need local repair mechanisms, but a new address lookup if routing fails.

Safari [11] on the other hand uses only one fixed address per node. The network is divided into hierarchical cells. Within a cell a node acts as a beacon disseminating locality information within a well-defined scope. The information which node is in which cell is stored in a network-wide DHT. Packages are routed towards the cell of the destination nodes. Inside the cell a route discovery is started on-demand to find a path to the destination by flooding a route request message. Thus, proactive mechanisms are used to spread locality information between cells and reactive mechanisms to route inside cells. This approach can fail at two points, at the intra-cell or at the inter-cell communication. Both necessitate individual local repair mechanisms. If the inter-cell routing fails, Safari will flood route request messages to find the destination again. This is also true for a failure in intra-cell communication and so both repair mechanisms share the same issues as DSR.

Both approaches try to include node mobility in their addressing and/or routing scheme. However, they still have issues dealing with topology changes caused by node mobility. Concepts to overcome this are similar to those employed by DSR and AODV, although this approach claims to have advantages over them by its location based view which helps to reduce the number of hops towards the destination to a minimum.

2.6 Chord

P2P in MANETs are an interesting application. Although P2P systems designed for the usage in the Internet do not work well in mobile scenarios, we present here Chord as a popular example for a structured P2P overlay and also because SONAR borrows its structure from it.

There are many definitions around to describe peer-to-peer (P2P) systems. In general, a P2P network does not have fixed clients and servers but a number of peer nodes that function as both client and server to any other node of the network, so any node is able to initiate or complete any supported transaction. Furthermore, P2P implies the independence from any central components, the participating nodes are fully self-organized and form the network by themselves completely decentralized.

2.6.1 Structured P2P Networks

Speaking of structured P2P networks, *distributed hash tables* (DHTs) gained a very high popularity in the research community in the last years. In a DHT, each node assigns an address from an identifier space by using a hash function (using the hash of information like the MAC address, IP address etc). So, the addresses are meant to be unique and distributed equally over the identifier space.

Several algorithms have been proposed to route in this environment, they differ in the way the virtual network is seen and how to find the next hop towards the destination. Popular examples are Pastry [27], CAN [25] and Chord [29]. As ISPRP initializes a Chord-like network, we go into details about this approach.

2.6.2 Protocol Details

Chord structures the virtual address space into a ring which overlaps at the highest possible address to zero as you can see in figure 2.2. Assuming that we represent identifiers as binary numbers and each identifier has m bits, we have 2^m distinct addresses (and so a maximum of $(n = 2^m)$ nodes) and the identifiers are ordered in an identifier circle $mod 2^m$.

Each node maintains its successor in the circular identifier space (and so always knows its predecessor, too, by the way). By this, routing correctness is achieved as every node can route towards the destination by its successor. Greedy routing is used which means that a node uses the next hop as close to the destination as possible. However, each node just has a local view of the network and there is no need to know the whole network for the ability to route to an arbitrary destination. But using only successors to route messages leads to $O(n)$ hops to a destination.

Therefore, Chord introduces *virtual neighbors* (called the *finger table*). The i^{th} neighbor of node p is the node which identifier succeeds p by 2^{i-1} on the identifier circle (see Figure 2.3). So, only $O(\log n)$ hops are needed for routing while each node has to maintain $O(\log n)$ states.

To keep nodes' successor pointers up to date, a basic "stabilization" protocol is used. The successor pointers are then used to verify and correct finger table entries which might change due to nodes joining the network.

Periodically, a node i requests the predecessor pointer of its successor k. If it still points to i, the successor is still correct. Otherwise, node i requests the predecessor pointer of k's predecessor. This is done until the node i gets back a pointer to itself from its new and correct successor.

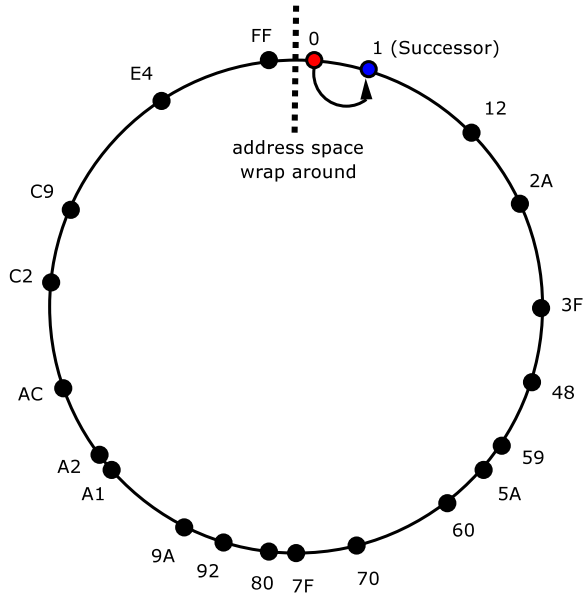


Figure 2.2: The circular address structure of Chord for 8bit long addresses. The address space wraps around after the node address *FF* to 0.

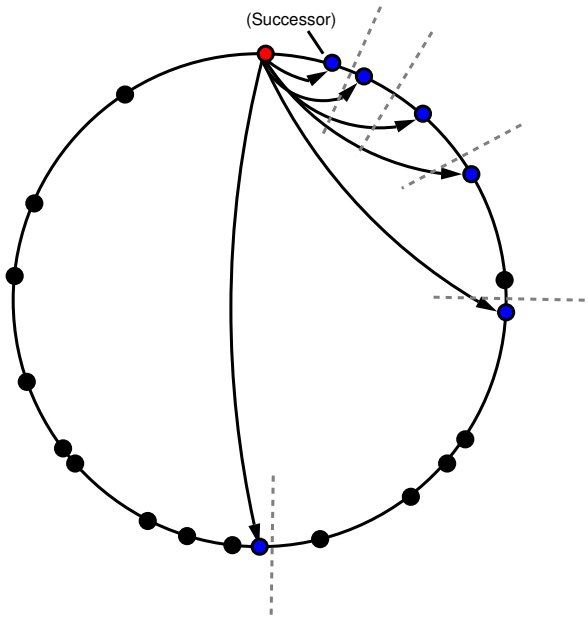


Figure 2.3: Each Chord node divides the address space in exponentially growing areas. The first existing node in each of these areas is called a virtual neighbor and acts as a shortcut to this area.

To keep the finger table up to date, each node randomly chooses one entry and looks up the the best existing node for this entry. For every successor pointer update also one finger table entry is updated.

2.6.3 Limitations

Chord was build for usage in the Internet and relies on an underlying routing protocol like OSPF or BGP. However, it was not designed for the limitations of MANET environments. Main issue is the overhead to maintain the ring structure (i.e. successor and virtual neighbors), as MANETs are very bandwidth critical. This is also a problem as the dynamic nature of MANETs (caused by mobility and packet losses due to congestion) forces Chord to repair its structure rather often and thus produces more overhead.

2.7 P2P in MANETs

There are a number of publications that are concerned with P2P protocols for ad hoc networks. Many of these are mainly conceptual, presenting architecture proposals without any evaluation.

Hu and others [15] proposed DPSR, which is a combination of the structured P2P overlay Pastry [27] as a network-layer routing protocol and the MANET routing protocol DSR. At the moment, this work lacks evaluation results. Ekta [24] is a DHT for MANETs based on DPSR. However, it uses flooding-based route discovery

Structured P2P networks like DHTs establish their own overlay addressing scheme where addresses do not reflect the underlying physical topology. To take advantage of proximity a technique called *proximity neighbor selection* (PNS) was introduced [14]. Here, the entries in the routing table are chosen based on their proximity. Especially in overlay networks where a hop in the overlay results in multiple hops in the underlying network, PNS can reduce the number of hops towards the destination significantly.

Contrary to references like [24], PNS also works in MANETs [8] and can help to reduce the path stretch (the increase in number of hops from source to destination routed in the overlay compared with the shortest available physical path) and so improve the efficiency of the routing.

There are a number of approaches to run DHTs that were originally developed for the Internet in wireless networks as shown before. However, these protocols are not tuned for the MANET environment. They consume a lot of bandwidth to establish and maintain their structure and are unaware of the effects of node mobility. And of course they suffer additional overhead from the underlying routing protocol.

This thesis shows that taking the specialties of MANETs into account while designing the protocol – mainly the structure maintenance – produces a more adequate behavior and bandwidth usage. The cross-layer design reduces overhead and offers a routing protocol and P2P system of a piece.

In the next chapter we analyze the requirements to run ISPRP in mobile wireless networks and what challenges arise from the mobility.

Chapter 3

Analysis

Running services in a wireless network, one has to focus on the special requirements of this environment. Such networks suffer from low available bandwidth and high failure ratios due to noise disturbing data transmissions. In MANET environments, we furthermore cannot rely on available infrastructure (for example access points) that handles the routing, but each node acts as a router. This increases the minimum protocol overhead significantly as nodes have to exchange information to be able to route packets. To generate as little protocol overhead as possible to avoid congestion is a hard but important challenge.

Also link failures due to node mobility occur quite often which challenges link repair mechanisms and make different approaches necessary compared to wired networks.

In this chapter we analyze the needs and requirements for an ad hoc routing protocol that consumes as little as possible bandwidth while offering good routing performance (in terms of short routing paths and thus low latency).

3.1 Overview

Most of the current implementations of P2P systems in MANETs [15, 24] build on an existing ad hoc routing protocol like DSR or AODV. Thus, the P2P network also shares the shortcomings of the underlying routing layer, in the case of DSR and AODV for example expensive route discovery floods. As the P2P system itself offers mechanisms to route data to a node which is member of the network, routing is done twice and protocol overhead is higher than needed.

SONAR integrates both parts, P2P system and ad hoc routing protocol, into one cross-layer protocol. Routing decisions are made based on the P2P structure and thus explicit route discoveries are unnecessary. Instead, SONAR maintains the P2P structure in a similar way as Chord in order to successfully and reliably fulfill routing tasks.

Furthermore, as described in Section 2.1, ISPRP is a message-efficient, scalable protocol to initialize a Chord-like P2P network between nodes having but link layer connectivity. Thus, it issues little control traffic to initialize the network, that is the P2P structure, and does not even depend on an underlying routing protocol to establish its ring structure.

In the following we analyze the functionality that is needed to enhance ISPRP with methods to work in MANETs, focusing in particular on methods to meet the require-

ments of node mobility and low available bandwidth. Furthermore approaches to optimize the protocol behavior are discussed.

3.2 Bandwidth Usage

Especially in the area of ad hoc routing protocols we see pure proactive and reactive protocols where in many scenarios, reactive approaches seem to outperform proactive ones. This is because early proactive routing protocols were just transfered from already existing approaches for wired networks. In these protocols, each node floods periodical route information messages to the network, hence causing a lot of traffic. These protocols show good results if routes are looked up very frequently as the already existing routing information can be used. However, their convergence speed is bound to their update period.

Reactive approaches work in a similar way with the difference that they only flood the network when they actually need a specific route to transmit a message. Hence, they still cause a lot of protocol overhead, especially with frequent discoveries.

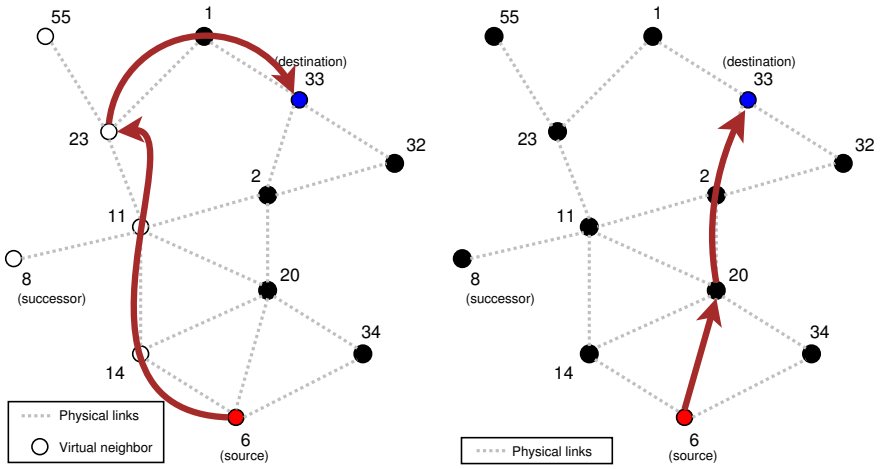
SONAR does not need route discoveries or global network information in order to find the next hop of a message. Instead, the P2P structure is used for routing. To maintain the P2P structure, SONAR employs proactive mechanisms, but it does not rely on flooding for route discoveries.

3.2.1 Routing in the P2P Structure

Routing correctness means that each message reaches its designated destination. Routing in SONAR is done in the P2P overlay network established by ISPRP. As ISPRP borrows its ring structure from Chord, each node has to maintain its *successor* in the ring. Thus, each node can route towards the destination by using its successor which guarantees route correctness as every routing decision brings the message closer to its destination.

Obviously, this simple scheme results in $O(n)$ virtual hops until a message arrives at its destination with n being the maximum number of nodes in the network. This can be improved in two ways: Chord introduces *virtual neighbors* which divide the whole address space in exponentially growing areas as described in section 2.6.2. This leads to $O(\log n)$ virtual hops towards the destination, but also to $O(\log n)$ state per node that has to be maintained. That is, $O(\log n)$ virtual neighbors have to be discovered by each node periodically as through node arrival and departure the correct node might change. To avoid network floods for discovering each virtual neighbor and thus significant protocol overhead, a modified version of ISPRP can be used to limit the number of messages per node (see section 4.1.3 for details). The initial overhead to find all virtual neighbors of a node stays significant though.

Another approach is to use only nodes from the physical neighborhood, that is nodes that are within 1-hop or 2-hop range. Messages to exchange neighborhood information are strictly local. Gathering knowledge of the 2-hop neighborhood is very easy to implement: Every node periodically sends 1-hop broadcasts that include a copy of all its 1-hop neighbors' addresses. Thereby, each node gets an up-to-date view of its 2-hop neighborhood. The freshness of this knowledge depends apparently on the frequency of these broadcasts. This supports proximity neighbor selection (PNS) as described in [8].



(a) Node 6 sends the message to node 23 which is the closest known node to 33 as it is its virtual neighbor. Node 23 knows a path to node 33, as it is in 2-hop range.

(b) Node 6 sends the message to node 20, as it is the closest known node to 33. Node 20 knows a path to node 33, as it is in 2-hop range.

Figure 3.1: Routing example in a network with 6 Bit long addresses: (a) with virtual neighbors and (b) with 2-hop knowledge only. Node 6 wants to send a message to node 33 and does not know a source path to it.

Routing is done in a greedy style: The original sender or an intermediate node chooses the next hop from its local routing cache, which is the one virtually closest to the destination but preceding on the ring. Whereas without virtual neighbors this will mainly be the successor or a hop in the physical 2-hop neighborhood. Thus, an overall routing length of $O(\log n)$ virtual hops cannot be guaranteed. If virtual neighbors are available, this can be guaranteed. Furthermore we still can use a node from the physical 2-hop neighborhood if it is closest to the destination from the local point of view, which combines PNS and virtual neighbors. Figure 3.1 gives an example of routing with and without virtual neighbors. Each node has 2-hop knowledge and a path to its successor in any case. As in this case the physical routing path for the example with virtual neighbors is longer (the virtual distance however is the same with 2 hops), apparently you can construct contrary examples, where the routing path with virtual neighbors is shorter – for example, if you just switch node 1 and 33, example (b) without virtual neighbors will route (6 – 20 – (2) – 32 – (1) – 33) (nodes in brackets are physical hops to reach a virtual hop) while example (a) is this time only 5 physical hops long: (6 – (14 – 11) – 23 – 33).

Evaluations have to show if the guaranteed $O(\log n)$ maximum path length of the virtual neighbors approach justifies the significant higher overhead compared to the usage of only 2-hop knowledge (see section 6.2).

3.2.2 Source Routes and Optimization

Both, successor and virtual neighbors make source routes necessary in order to be able to route to these nodes. These paths have to be maintained to keep them valid in the

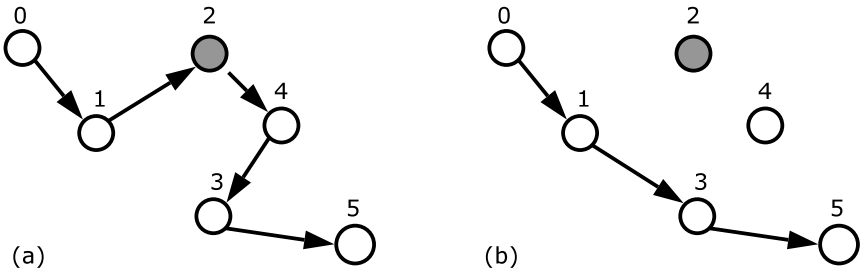


Figure 3.2: Optimization prunes current node from path

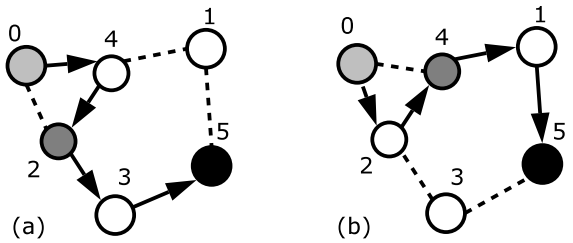


Figure 3.3: Dijkstra outputs a path that is the shortest possible and not longer than the original one, but it can output paths that are of the same length while being different.

face of node movement and network membership changes.

Maintaining ring consistency primarily involves sending periodical messages to a node's successor in the ring and is mandatory to guarantee a valid ring and thus routing correctness. Each node regularly tests its successor path by sending an update message to its successor including a source route header. The successor then replies with an acknowledgement.

The maintenance of the virtual neighbors is done in a similar way, although virtual neighbors are not a prerequisite for ring consistency. Updates can be less frequent than for the successor path. For example, Chord updates one neighbor per stabilization interval. However, to have valid paths to each of them and the correct and thus best virtual neighbor, periodical updates have to be made. Due to membership changes, a node which is closer to the ideal address of a specific entry in the virtual neighbor table can join the network. As the virtual neighbors act as shortcuts to a specific area of the address space, the node would assume that there is no other node for this area preceding the known virtual neighbor. Apparently this can lead to requests being routed to the wrong node or taking a longer route than necessary.

As the P2P overlay is used for routing, there can be a difference between the shortest physical route and the virtual one given by ISPRP. To overcome this, route optimization routines for the path can be implemented. As each node only has limited, mostly local knowledge of the network, the optimization ability is also locally limited. Therefore, optimization can be done in a distributed fashion by every node of the source path while forwarding the message. The Dijkstra algorithm [10] is used for optimization, taking the path already optimized by the preceding hop and the local routing cache as input.

The optimized path cannot be used instantly as the optimization could alter the

path in a way the current node is unable to find a next hop for the currently processed message. In Figure 3.2(a), node 2 optimizes the current source path (0 – 1 – 2 – 4 – 3 – 5) which results in the new path (0 – 1 – 3 – 5) which is shown in figure 3.2(b). Node 2 has pruned itself from the path and is now unable to forward the message using the new, optimized path. Even if it uses the former next hop 4, it is possible that this next hop cannot route on corresponding to the given path..

Therefore, the optimized path is saved in the packet besides the one used for forwarding. Each node optimizes this path and replaces the former proposal for a better path. The optimized path can then be used by the destination to send back an acknowledgement and by the source for further messages. This strategy is not limited to SPS messages, but can also be used for ACK and NACK messages.

Dijkstra can output a path of the same length as the input path, which is however different to it. This could lead to oscillations as shown in Figure 3.3. Node 0 wants to route a message to node 5. When the message arrives at node 2, it is (0 – 4 – 2 – 3 – 5). The Dijkstra algorithm outputs the path (0 – 2 – 4 – 1 – 5) that can be seen in (b). Thus, the next hop is again node 4 which will optimize the path and receives the path from (a) again. But such rooting loops can be easily avoided by not replacing the original path by an optimized path that has the same length. Not using the optimized path immediately also avoids routing oscillation.

Source routes are only used for messages to a successor or virtual neighbor, be it data or control packets. As a source route increases the packet size and thus also the protocol overhead we conceive of a technique to avoid them. Each member of a source route stores all paths it is a member of in its local routing cache. After the path converged – which means that further optimizations lead to no shorter alternative – the sender stops to add a source route to the header, but sets a flag in the packet header to tell the intermediate nodes that a message should be routed as indicated by the formerly stored path [17]. Thus, each node looks up the next hop from its route cache and forwards the message. After a specific time, the source could re-send again a source route header to again optimize the path which may be necessary due to node movement. Otherwise, the protocol will use source route headers again if a link of the path failed until it is optimized and all intermediate nodes have stored the new one.

3.3 Node Mobility

Structured P2P systems like Chord or Pastry are designed to handle so-called churn [26]. This is a steady change in the membership of a P2P network caused by nodes joining and leaving the network, mostly at a rather high rate. This is quite similar to the effects of node mobility in MANETs with the difference that in MANETs nodes often only change their *location* instead of leaving the network at all. Both effects require repair mechanisms for the P2P structure used. Note that we focus on mechanism to overcome link breaks caused by node mobility and not by membership changes.

Current implementations of DHTs are built for wired networks, mostly assuming almost unlimited bandwidth. Mechanisms to repair link breaks utilize a lot of bandwidth as they try to detect breaks very fast and collect information in advance to overcome failures almost immediately. Thus, they are not usable in MANETs without adjustments. Also, they should take into account node mobility. A moving node is different from a node disappearing as some links break and new ones appear.

This leads to slightly different repair methods if we assume that a node just moved instead of disappearing. As a node just moves with finite speed it might still be reach-

able through another node which is still in communication range. Previously proposed DHTs just look for a replacement or rely on the routing mechanisms of the underlying (ad hoc) routing protocol. We propose three different repair approaches described in the following. They are strictly local, and each repair run increases the path length at most by one hop. These mechanisms are reactive and thus are only executed if a source route (e.g., for a successor pointer or a virtual neighbor) breaks.

3.3.1 Repair Mechanism Approaches

Link breaks occur as a result of node mobility, congestion or the node leaving the network at all. Especially the first one is characteristic for wireless networks. For the latter it is a typical assumption in MANETs, that memberships does not rapidly change. A link to a specific node will break if it leaves a node's radio range. But this node can still be reachable over another node which is still in range, thus the node is likely to be still in the 2-hop neighborhood. Repair mechanisms will try to find this alternative 2-hop route to the node to overcome the link break, if possible. Congestion has the same symptoms as the other two causes for link breaks and therefore can hardly be distinguished from them. Specific symptoms to handle congestion are not part of this work. However, for temporary congestion the presented repair mechanisms here will also find an appropriate bypass path.

The first approach tries to find a bypass from a node's routing cache, in particular to find a 2-hop path to the node not reachable via 1-hop anymore. Therefore the failed node is searched in the 2-hop cache. This is based on the presumption that the node just moved on, out of a node's radio range but still is in range of one of its direct neighbors. If there are several possible bypasses, the most recent one – which corresponds to the last received HELLO packet from the bypass node – is used. This method relies on known alternative paths that are still valid despite node movement. However, it is not possible to distinguish which paths in the routing cache are still valid and which are not. Choosing the most recent entry is an indicator for validity but no guarantee for it.

The second approach is very similar to the first one, because the bypass is chosen from a node's 2-hop cache, too. Each entry in the cache is refreshed by a HELLO packet every HELLO_INTERVAL seconds if still valid. The algorithm checks the timestamp of the chosen entry and if it is older than half of HELLO_INTERVAL, it waits (HELLO_INTERVAL/2) seconds for routing cache updates before finally choosing a bypass. If the entry is recent enough, the bypass is used immediately and thus shares the same characteristics as approach 1 in this case. The idea is that a node does not disappear but moves on and so turns up as a new 2-hop neighbor. The update interval determines accuracy, short intervals result in frequent updates to the locally stored topology in the routing cache. The re-appearance of a node depends on the update timing. By waiting some time, a node receives updated HELLO messages, potentially telling it where to find the lost node now. Of course, this approach introduces a repair delay of (HELLO_INTERVAL/2) in the case that the routing cache is not as recent as required.

The third approach is taken from an optimization to the AODV protocol as described in section 2.3.2. On a link break, a node sends a 2-hop limited discovery broadcast to re-discover the lost node again. Only the node searched for replies, sending back the source route that was collected in the broadcast message. If the node just moved on and did not disappear, it left a node's radio range but is very likely to be still in its 2-hop neighborhood. Besides a short delay while waiting for a reply to the broadcast, this method also introduces additional traffic within a range of 2 hops around the

node which tries to repair a link break.

In order to evaluate which of those three approaches perform best, we implemented and simulated all three of them in GloMoSim. The simulation parameters and results can be found in section 6.1.

In the next chapter you find the final design of SONAR which we used to evaluate the protocol and to compare it to already existing approaches.

Chapter 4

Design

There are several ways to implement a mobility enhanced version of ISPRP. We chose those mechanisms that should give us a good performance while keeping the protocol overhead low. In this chapter we present the design details of SONAR, as it was used to implement it in the GloMoSim network simulator.

We divided this chapter into two parts: First, we take a look on the virtual topology introduced by the P2P structure. It deals mainly with the structure establishment, maintenance and routing. Secondly, we focus on the physical topology which deals with node mobility, link breaks and physical locality issues.

4.1 Virtual Topology

The structured P2P system establishes a virtual addressing scheme for the network. A unique node ID is assigned to each node in the way that the used IDs are equally distributed over the address space. However, this is independent from the physical location of a node. Nodes which are close to each other in the virtual overlay may be several hops apart in the physical topology. Thus, the virtual overlay introduces a path stretch as an 1-hop distance in the overlay can result in a multi-hop distance in the physical topology.

The ring structure is maintained by every node periodically updating its direct successor in the ring. As this process is the reason for a big part of the overall protocol overhead, this has do be done efficiently.

4.1.1 ISPRP

To keep the protocol overhead low, ISPRP is used to initialize the network and to maintain the P2P structure. As described earlier in detail in Section 2.1, a node chooses a potential successor from its local routing cache and sends an SPS message to it. The SPS messages are routed along the included source route. The destination will then check its local knowledge if it really is the successor of the sender and send back an acknowledgement if so. Otherwise it sends back a negative acknowledgement including a source route to a node that is the original node's correct successor from its local point of view. So far, ISPRP assumes bidirectional links, as the same path is used to send SPS and ACK/NACK messages.

After receiving an ACK message, a node switches to maintenance node, periodically checking the validity of the successor path by sending a SPS message. If the predecessor of a node changes (for example by a node joining the network), the former predecessor will be informed of this by a NACK message. As this NACK might get lost due to link breaks, the periodic SPS updates furthermore determine the validity of the successor itself.

Adding timeouts and regular SPS transmissions to update source routes is a requirement to enable ISPRP for MANETs. After a timeout the node assumes a problem. In maintenance mode, the node chooses a new successor from its routing cache after the timeout occurred several times. Then, a problem with the source path is assumed, for example congestion or a link break. We set that parameter to a maximum of two timeouts. After that, the node removes the failed path from its routing cache and uses ISPRP to find a new successor. If a timeout occurs during the ISPRP run, the node will prune the current successor from its routing cache and will re-start ISPRP with a fresh successor.

In the case of link breaks a repair mechanism (which will be described later in detail) helps furthermore to avoid additional SPS transmissions. It tries to locally repair broken paths and informs the sender of the changed path, furthermore repairing the path in the failed message before forwarding it.

4.1.2 Routing

Routing decisions are based on the P2P structure. If the destination is not directly reachable, the next hop will be chosen in a greedy style as the one with the smallest virtual distance to the destination but preceding it on the ring. The availability of a node's successor guarantees that every routing step brings the message closer to its destination. This is the same routing scheme as used by Chord. The route length highly depends on the information found in the local routing cache.

Chord also introduces virtual neighbors to shorten the maximum path length from source to destination to $O(\log n)$ hops. To initially find a virtual neighbor requires the same effort as to find a node's successor. But there are $O(\log n)$ virtual neighbors significantly increasing the overall effort.

4.1.3 Virtual Neighbor Lookup

To efficiently initialize and furthermore maintain the entries in the virtual neighbor table, we conceived a modified version of ISPRP, called *Fuzzy-ISPRP*. It shares the same characteristics as the original ISPRP in terms of message efficiency and scalability. But by slightly modifying the protocol algorithm, it is able to find a node closest to a given address instead of being closest to the sender itself.

Initially, the virtual neighbor table is filled with the best locally available nodes for each entry. These are mainly from the successor path and the 2-hop neighborhood in the first place. In a fully populated address space, the ideal successor would be entry 0 (with virtual distance $2^0 = 1$). The real successor will mostly have a higher virtual distance than one because not all addresses in the virtual address space are used. Thus, the successor is also used for the entries where no real virtual neighbor exists.

The sending node chooses the next available entry in its virtual neighbor table and calculates the ideal node that would fit there. This procedure is not done for entries where the current successor is stored in as the normal ISPRP routine will update those entries in case the successor changes.

The Fuzzy-SPS message is extended by a field to store the address of the ideal address of a specific virtual neighbor we try to look up. This field is called the *sharp destination*. All the other message fields, as there are source and destination addresses and source route, stay the same.

If the sharp destination exists, Fuzzy-ISPRP will work just like normal ISPRP as the ideal entry for the virtual neighbor table exists. Otherwise, Fuzzy-ISPRP will find the best alternative node fitting in that particular entry. The message is forwarded to the node closest to the ideal address but succeeding it on the ring. The process is the same as for successor lookups. The sender chooses the best known node for a specific virtual neighbor table entry and sends a Fuzzy-SPS message to it. If this node knows a closer node to the sharp destination (or even a path to the sharp destination itself), it will send back a NACK message including a new source route to it which can then be used for the next Fuzzy-SPS message. Otherwise it sends back an ACK.

Assuming virtual addresses of length 32 bit, this results in 31 virtual neighbors per node (virtual neighbor number 0 is the successor itself, which is updated by ISPRP anyway and is not part of the Fuzzy-ISPRP routine). As the bandwidth is very limited, we cannot update them at once, but one after the other. However, this introduces a rather high delay until a virtual neighbor path is updated. Decreasing this delay automatically results in significantly more overhead.

Optimizations to limit the overhead of this function is very hard to find. As it is very likely that the “real” virtual neighbor will not even be a part of the network, the next best fitting node will fill its position. But even as we hear regularly from that node (for example through HELLO messages) we cannot assume that this virtual entry is still valid as a more suitable one could have appeared in the meantime.

4.2 Physical Topology

It is a significant advantage for a node to know its physical neighborhood. It provides a basic knowledge of the network. If a node joins the network, it will send a bootstrap message to its direct physical neighbors introducing itself. Its 1-hop neighbors then add this node to their routing cache and reply with a HELLO message. This gives the new joining node an initial 2-hop knowledge of the network from which it chooses its initial successor to send an SPS message to.

Physical knowledge can also help to choose short routing paths, to optimize them or to find local bypasses for broken links.

4.2.1 HELLO Messages

As described before, we use periodical HELLO messages which are broadcast to a node’s physical neighbors. Included in these messages are the addresses of the physical 1-hop neighbors of a node. By this, every node has a recent, regular updated 2-hop knowledge of the network. The cost for these messages is apparently bound to the number of included addresses and the frequency of the broadcasts. However, they are only sent within a 1-hop range.

The IEEE 802.11 MAC protocol uses a carrier-sense mechanism for broadcast transmissions to avoid multiple usage of the medium and thus collisions (find details in Section 5.2). But this mechanism can fail due to the hidden-node problem. Furthermore, long broadcast packets will block the medium for a long time thus causing congestion.

In very dense networks, including the complete physical neighborhood of a node will produce big and therefore expensive HELLO messages as this also increases the protocol overhead. But this will give each node the knowledge of a big part of the network, too, making other mechanisms like SPS messages mostly unnecessary as their job is already done by the HELLO messages.

Apparently, it is also possible to get 3-hop knowledge of the network by this mechanism if each node also includes its 2-hop knowledge in every HELLO packet. This causes the HELLO messages to grow exponentially in size compared to only including 1-hop neighbors. As our main goal was it to limit protocol overhead and the heavily increase message size could lead to congestion, we decided not to use this and instead only include 1-hop neighbors in HELLO messages.

4.2.2 Source Routes

Source routes are used mainly for ISPRP protocol messages. Furthermore, they are used for routing data packets if needed. As the routing is done in the virtual overlay, the chosen virtual next hop may not be in physical 1-hop range. In this case, a source route header which is used to route the data packet is added to the message. At the end of these intermediate source routes, that is when the virtual next hop is reached, the source route header is removed and a new routing decision is made.

To route a packet from source to destination in the virtual topology, several alternative paths in the physical topology may exist. It is a main goal to keep the physical routes as short as possible. Therefore, we conceived a distributed route optimization procedure as described in Section 3.2.2. When an SPS or a Fuzzy-SPS message is sent, it also includes a field for an optimized path. Each in-between node optimizes this path and stores the result in this field. The optimized path is then used at the destination for all further routing decisions, also for the ACK message that is sent back to the sender which will then use the new, optimized path. As in rather small networks the paths are short despite the usage of an optimization routine, it is only used for SPS messages. However, the optimization could also be used in NACK and ACK messages to speed up the convergence for a (local) optimum.

4.2.3 Link Repair Mechanism

As the simulation results in Section 6.1 show, using a two-hop-limited broadcast in order to overcome a link failure and bypass it performs best among the three presented solutions for this problem. The additional overhead does not seem to cause severe additional congestion. Of course, if the link failure is due to congestion, our approach will even make the situation worse.

For this 2-hop search a node sends a limited 2-hop broadcast to re-discover a node which became unavailable before. Until the repair succeeds or fails, all incoming messages which try to use the same broken link are buffered. When the broadcast eventually reaches the search node, the discovered bypass-path is sent back to the node that originally has sent the broadcast. It then uses the new path to update its routing cache and to fix the broken path in the messages that tried to use the broken link and were buffered. It furthermore sends back a route error message informing the original sender of the message of the repaired path to avoid further failures due to the usage of the old path.

If the repair fails which means that no answer to the broadcast arrived before a timeout, the message is dropped and a route error (RERR) message informs the orig-

inal sender of the broken route. The original sender of the message then has to find a new path to the destination. SONAR takes care to avoid RERR swamping: The sender of one or more failed messages is informed by only one single RERR message about the failed link instead of sending one for every dropped message. This also avoids a negative feedback cycle: If congestion was the cause for the link break, sending multiple RERR messages would make the congestion worse.

In the next chapter you will find a brief description of the SONAR protocol as implemented in GloMoSim. This should give an insight in important details of the implementation for future extensions.

Chapter 5

Implementation

In order to evaluate SONAR, the protocol was implemented in the network simulator GloMoSim. This chapter gives a brief overview of the implementation details.

5.1 GloMoSim

GloMoSim [13] is a scalable simulation environment for wireless network systems. It supports a wide range of parameters for setting up a simulation environment and was designed to scale with the number of nodes. Besides *ns-2* [20], it is the standard simulator for wireless ad hoc networks.

The simulator already contains implementations of various protocols on all layers and is easily extensible. For example, there are IEEE 802.11 and TSMA on the MAC layer, AODV and DSR as two very popular ad hoc routing protocols and IP on the network layer. Furthermore, TCP and UDP on the transport layer and clients and servers like HTTP, FTP or Telnet on the application layer.

GloMoSim includes the random waypoint model, and other mobility models can be used through mobility trace files. As there is no implementation of the random direction model, we generated mobility trace files for our simulations.

5.1.1 Addressing Scheme

GloMoSim only offers a fixed addressing scheme, enumerating each node address starting with zero. This is not a sufficient starting point for a P2P protocol which relies on the addresses being uniformly distributes (as all DHTs and so also SONAR do). Therefore, we assigned a 32-bit long virtual address to each node (while one address is reserved). SONAR uses a mapping table for translating addresses back and forth between MAC addresses and virtual addresses. No additional address resolution mechanisms are needed as a node periodically hears from all its physical 1-hop neighbors and thus gets the MAC address of all its physical neighbors (besides their virtual address included in the HELLO packet itself). This is sufficient to avoid extra ARP calls. The virtual addresses are randomly generated and are read from an address map file, hence being identical over all simulation runs.

In real world implementations, virtual addresses can be assigned in several ways. For example they can be chosen randomly by the new node itself [6] or they can be computed by hashing the MAC address of the wireless interface or from other available

identifiers that result at best in an unique virtual address [29]. Furthermore, the hash of a public key or a certificate given by a trusted certification authority can be used as the virtual address [4].

5.2 MAC Layer

In our simulations, we used the IEEE 802.11 MAC protocol. Thus, the MAC layer reports back after four failures to transmit the package. Without this mechanism, an additional timeout mechanism and acknowledgements implemented in the routing layer would have to be used to detect link failures. We used this MAC protocol for all of our experiments.

The IEEE 802.11 MAC protocol handles broadcast and unicast packets differently. For broadcast packets, only a sensing technique is employed (CSMA: Carrier Sense, Multiple Access) to avoid collisions. A node will listen to the medium before sending the broadcast packet to detect whether the medium is idle. Collisions can happen anyway in the case of the hidden node problem or the exposed node problem.

For unicast packets, the node furthermore uses a handshake mechanism called CSMA/CA (Carrier Sense, Multiple Access / Collision Avoidance) to ensure that the medium is idle. If the medium appears to be idle (i.e. the sender cannot sense any activity), the sender will broadcast a *request to send* (RTS) message to the destination. The destination answers with a *clear to send* (CTS) message if it is ready to receive a message. Both messages also include the time the medium will be occupied for the transmission to tell other nodes in range when the medium will be available again which avoids hidden terminals for unicast.

5.2.1 IP Protocol

As SONAR uses its own addressing scheme, it supersedes the network layer and is independent of IP. However, as it is almost impossible to remove IP from GloMoSim, we were forced to still use it. We only used the IP interface to higher layers and the IP fragmentation mechanism to honor maximum segment sizes.

However, the IP header with 20 bytes is always present, even in SONAR control packets. To add a path to data packages which should be routed by SONAR, a new IP option field was created which carries the source route if needed; if the next hop is just one physical hop away, no source route header is added.

5.3 Structure

The implementation is divided into three main parts: First, there are the *periodical HELLO messages*. As described before, each nodes sends regular HELLO broadcasts to its physical neighbors including its own 1-hop neighbors. By this, every node gains 2-hop knowledge. Whenever a node sends a HELLO broadcast, it first prunes its 1-hop neighbor cache to remove stale neighbors.

The second part is the implementation of the *ISPRP protocol*. The protocol was not altered at all compared to the description in Section 2.1, but slightly the way ISPRP is used. If a message fails to be transmitted due to a link failure, a repair function is triggered that tries to find a bypass. We do not use strict source routing here, as repairs might change source routes in-between if needed. Failed source routes are reported

back to the sender. If a bypass was found, the repaired path is included in the message for further transmissions. Furthermore, regular updates are scheduled to cope with mobility and packet drops.

To avoid bursts of control messages which would appear especially in the joining phase of a node, we implemented a sending rate limitation for SPS and Fuzzy-SPS messages. This can result in a longer period of time until a node finds its correct successor, yet will also limit temporary congestion caused by ISPRP runs.

Third, the routing functionality: Packages that arrive from the transport layer are routed towards their destination in the P2P structure as established by ISPRP. Routing is done in a greedy style: The node virtually closest to the destination but preceding it is chosen as the next hop. This might require a source route to be added to the data package in order to reach the next virtual hops through several physical hops. However, source routes are only used if needed.

5.4 Message Types

5.4.1 Control Messages

We can divide the control messages into three main categories: ISPRP packets, repair messages and HELLO broadcasts.

ISPRP uses three different messages: SPS, NACK and ACK. They all consist of a field for the source path and a sequence number. SPS and NACK furthermore have a field for an additional path. In the case of the SPS message, this path contains the optimized path and the NACK message carries the path to the new suggested successor. The fuzzy counterparts only add another field for the “sharp” destination address.

The repair routine uses three different messages. The repair broadcast uses a sequence number to avoid duplicate transmissions and a field to store the source path towards the destination. The repair reply message includes this source path back to the node that started the repair broadcast. To inform the original sender whose message caused the route break to be detected, a route error (RERR) message is used. This either contains the new, repaired path if the repair was successful or otherwise the broken path. Thus, the original sender can prune this path from its routing cache.

HELLO messages contain all the virtual addresses of the 1-hop neighbors of a node, as well as its own. This message is broadcast to a nodes 1-hop physical neighbors to gain 2-hop knowledge. When a node joins the network, it sends a bootstrap message. This only contains the address of the new node and is broadcast to its physical 1-hop neighbors.

Sequence Numbers

Sequence numbers are needed to avoid multiple transmissions of the same message over a link for multi-hop broadcast – which are repair packets and the ISPRP flood message.

ISPRP also uses sequence numbers to detect packet loss and reordering. Especially for SPS messages, delayed NACK packets could supersede an ACK packet from a later sent SPS message. This results in further SPS transmissions to find a node’s successor which are totally unnecessary, as a valid path to the successor was already found. Thus, the destination copies the sequence number from an SPS message to the

ACK or NACK, respectively, and the sender can decide if the message is still valid or outdated.

We use no sequence numbers for RERR and repair response messages, as these are source-routed messages and out-of-order arrival is no problem. These messages are only sent once per link break (in case of the repair response message only if the repair broadcast reaches the lost node).

5.4.2 Data Messages

GloMoSim encapsulates data messages from upper layers in IP packets. As said before, we do not need the additional IP header, but cannot remove it so far, as GloMoSim relies on it.

In general, routing is done hop-wise and we do not employ the IP source route header, as it only supports strict source routes without the ability to repair routes in-between if needed.

If the next virtual hop is just one physical hop away, the packet is sent there directly. The next hop then routes the message according to its own routing cache. In case the next virtual hop is multiple physical hops away, we attach an IP option header to the message where the source path is saved in. The message then is forwarded to the next hop on the path, and so on. The IP option header is removed when the destination of the source path is reached and a new routing decision has to be made.

5.5 Routing Cache Implementation

The routing cache is organized as shown in Listing 5.2. All known nodes are saved in a double-linked list. The head of the list is the successor, the tail the predecessor. Listing 5.1 shows the SONAR_node structure, containing the pointers for the double-linked list, the node's virtual address, and a reference counter to save how often a node is a part of a cache entry. This is necessary as a node can be a 1-hop, a 2-hop neighbor or part of a stored source path or any combination of these. A node will be removed if its use counter is zero, meaning the node is no longer reachable using the information in the routing cache.

```
typedef struct SONAR_node_str{
  unsigned int address;
  unsigned int use_count;

  struct SONAR_node_str *next;
  struct SONAR_node_str *prev;
}SONAR_node;
```

Listing 5.1: SONAR node structure

```
typedef struct{
  SONAR_node *successor;
  SONAR_node *predecessor;

  SONAR_node *virtual_neighbors[VADDR_SIZE*8];

  struct SONAR_onehop *onehop_set;
  int one_hop_num;
  struct SONAR_twohop *twohop_set;
  int two_hop_num;

  struct SONAR_source_path own_paths[2];
  struct SONAR_source_path *succ_paths;
  int path_num;

  unsigned char ISPRP_status;
}SONAR_routing_cache;
```

Listing 5.2: SONAR Routing cache

Furthermore, virtual neighbors are saved separately and point into the double-linked list at the nodes that are the best available nodes for a specific virtual neighbor entry.

Next are linked lists for the 1-hop and the 2-hop neighbors. Own_paths contain the path to a node's successor in the first entry and the path to a node's predecessor in the second one. Succ_paths contains all further paths a node is a member of. These are successor paths of other nodes or paths to virtual neighbors (if this extension is used).

ISPRP_status indicates if a node still tries to find its real successor or whether the present successor is valid and maintenance mode is active.

Whenever a node is inserted, no matter if as part of a source path or as 1-hop or 2-hop neighbor, it is also added to the linked list of known nodes and a reference counter is set, as a node can be part of multiple entries in the cache.

1-hop and 2-hop neighbors and source paths are saved together with a timestamp to store when a specific entry was last updated. This is used to calculate the expiry time of an entry. If an entry expires, it is pruned and the use counter of the corresponding node structure is decremented. If the use counter is zero, there is no path to it available anymore and thus the node structure is removed from the double-linked list.

5.6 Path Optimization

In order to get as-short-as-possible paths, a local path optimization routine is used. As each node just has local knowledge of the network, running optimization only on the endpoints of a path would be ineffective. Thus, we chose to distribute the optimization to any member of a route.

Optimization is done only for SPS messages to limit the computational effort of running the algorithm. Furthermore, the optimized path is only used by the two endpoints of a message and never by intermediate nodes. Otherwise, it could happen that a node removes itself from the path as an optimization and so will fail to find the next hop towards the destination (see section 3.2.2). Hence, besides the actual source path each SPS message also contains a field for the optimized route.

An implementation of the *Dijkstra algorithm* [10] is used to optimize source paths. Therefore, all locally available routing information is used from the routing cache. The local view of the network graph is built from the 1- and 2-hop knowledge, available successor and predecessor paths and miscellaneous paths the node is a member of. Furthermore, the original path and also the optimized path (which is also saved in the SPS message next to the currently used one) is included. The algorithm calculates a new shortest path which is then included in the message as the optimized path before forwarding it again; the original path is not modified.

When the message eventually reaches the destination, the optimized path is saved and used to send back an ACK message if appropriate. The original source node of the SPS message then uses the path it gets from the acknowledgment packet as the new route to its successor or any other virtual neighbor.

If the destination rejects the SPS message – which means it knows a more appropriate successor or virtual neighbor j for the sender –, it will just attach the path to j and will then optimize this new path which will also prevent any routing loops. The new path is not immediately used because the optimization might remove the former destination node itself from the path and so routing back would be impossible as the node cannot find the next hop. So, the original path is used to route the message back to the source which then considers the newly provided path for its next SPS try.

However, one could also add optimization for ACK packages, thus optimization could converge up to twice as faster to the shortest path. This includes of course additional overhead for a second path field added to each ACK package.

5.7 Link Repair Mechanism

The repair mechanism is implemented as described before in section 4.2.3. The MAC protocol triggers an upcall to inform us that it was unable to transmit a specific packet (it will not report back if it fails to transmit broadcast messages). SONAR then buffers this message and whether it already is aware of this link break. Only if this is a “new” link break, a repair broadcast with a maximum hop count of 2 is sent. A sequence number avoids that the same message is broadcast more then once by a node.

If the broadcast arrives at the node searched for, this node will send back a repair response message to the sender of the broadcast. The bypass path is stored in this message. The path is then used to repair the formerly broken source path or to bypass a broken single hop route. When the sender of the repair broadcast receives the repair response, it uses the included path to repair and forward all intermittently buffered packets waiting for the path. Furthermore, an RERR message is sent back to the original sender including the repaired path. This message is sent only once per sender and not per message.

The sender of the repair broadcast sets a timeout to wait for a repair response. After a timeout, the node removes all messages in the buffer that failed due to this specific link failure as there concurrently might be multiple repair attempts. Furthermore, an RERR message is sent to the original sender which will then use an alternative path to the message’s destination.

In simulations, we observed that immediately reacting to RERR messages for successor paths by rediscovering a path to the successor using SPS messages is a disadvantage. Instead of improving the ring consistency, it causes additional overhead and thus congestion. We prefer proactive repair over reactive repair for this reason [26].

SONAR does not start a repair attempt for RERR or repair response messages, as

these are already part of a repair attempt. SONAR does not rely on the RERR messages, thus it is not a problem to lose them. Repair responses use the same path back that was discovered by the repair broadcast. If this path breaks, the whole repair attempt failed. This will mainly be a result of congestion.

After a successful repair attempt, a node caches the message type, the original sender and the sequence number of control messages for some time to detect double failures. This means that the newly discovered next physical hop of the bypass also failed. As the probability to discover another bypass is low, we drop such messages which also avoids loops. To avoid looping for data packets, we use a bit in the packet header to tag previously repaired source routes.

Chapter 6

Evaluation

The evaluation section of this thesis is divided into two parts. First, we simulated three different approaches to locally repair link failures. Second, the implementation of SONAR was simulated to evaluate its performance. A specific sub-goal was to assess the effect of using virtual neighbors for improving the routing performance. That is the routing stretch implied by routing in the virtual graph compared to in the physical graph.

6.1 Local Repair Alternatives

We implemented three different types of local repair mechanisms in the GloMoSim network simulator for selecting the best alternative to use in the SONAR architecture.

6.1.1 Overview

As described in detail in Section 3.3.1, to overcome a link failure the first approach only uses information found in the local routing cache. As node mobility causes links to break, other, redundant links may still be available. Information about these redundant links is disseminated by the periodical HELLO messages. From all available bypasses, a node selects the one updated most recently. This reduces the likelihood of choosing a stale and probably broken link.

The second approach is very similar to the first one. But if a node's most recent bypass entry is older than half the HELLO sending period, the node waits for the next HELLO to arrive. By this, there is the chance that all stale routes are pruned from the cache and new paths to the former unavailable node become available.

The third approach is taken from an AODV optimization [1] and uses broadcasts limited to a range of two hops to rediscover the unavailable node. A source route is saved in the message and if the destination is reached, it will report back to the sender. If the node has left another node's radio range due to node mobility, it is very likely that it is still within the 2-hop neighborhood.

6.1.2 Simulation Environment

We implemented a reduced environment to evaluate the three repair alternatives as described in the following. This should eliminate other effects, while we concentrated

on node mobility.

Each node maintains a local routing cache. Each node sends periodical HELLO messages including its local 1-hop knowledge of the network. By this, every node gets a 2-hop view of its neighborhood. This is the same mechanism later used for SONAR.

Furthermore, a random node is chosen to send data to another randomly-chosen node. The sender starts a route discovery to the destination after an initial time of 100 ms. Therefore, a discovery message is flooded through the network while saving a source route in it. The destination only answers the first arriving discovery message. After the sender receives the discovery reply, it sends 512 byte packets at a constant rate to the destination until the end of the simulation run. The sending rate is a parameter we varied in our simulations. We implemented no interface to the transport layer and did not use any higher layers in this simulation.

If a link from the used source route breaks, the node that detects the break (by an up-call from the IEEE 802.11 MAC layer) starts a repair try. If this succeeds, the source path will be repaired and the packet is salvaged. The source node is informed of the break providing the repaired source route. It will use the new route for further transmissions. Otherwise, a route error message is sent to the source which then initiates a new route discovery to find a new path to the destination.

Parameters

Several simulation parameters can be chosen to vary the simulation behavior. These are network density, the sending rate of the source-routed packets and the duration between two HELLO messages.

We simulated networks of with 50, 100 and 200 nodes, the first two in an area of $800m \times 800m$, the latter in $1600m \times 1600m$. This results in 2–12 measured physical neighbors per node on average for the 50 node set and 10–30 measured physical neighbors on average for the 100 and 200 node sets and offers a range from sparse to medium network density. The simulated time is 40 minutes.

As mentioned before, we used the random direction mobility model with a minimum velocity of $0m/s$ and a maximum velocity of $3m/s$. The average time a node moves into one direction is 70 seconds. These parameters model pedestrians walking around, for example on a campus, and avoids nodes busily moving around in a very small area for long times. The number of nodes stays constant over the simulation time as no nodes disappear and no new nodes are created.

Each node periodically broadcasts HELLO messages to its physical 1-hop neighbors. The time between two HELLO messages is configurable and we chose one, two, three and five seconds for our simulations. Note that only the first two repair approaches rely on entries from the routing cache while the third one does not. However, as we use periodic HELLO messages for SONAR anyway, we employed them in all three simulation parts to allow a fair comparison under similar conditions.

Sending HELLO messages every second provides a very up to date view of the neighborhood while consuming significant bandwidth and probably causing congestion. Sending HELLO packets every five seconds results in low protocol overhead but potentially stale routing cache entries.

After the selected source node starts a route discovery, it waits at most 500 ms for an answer before starting a new discovery. The sending rate for the source-routed packets is another configuration parameter and is either 3 or 5 packets per second.

The simulation was run with all possible parameter combinations, ten independent simulation runs were averaged to achieve reasonable statistical significance.

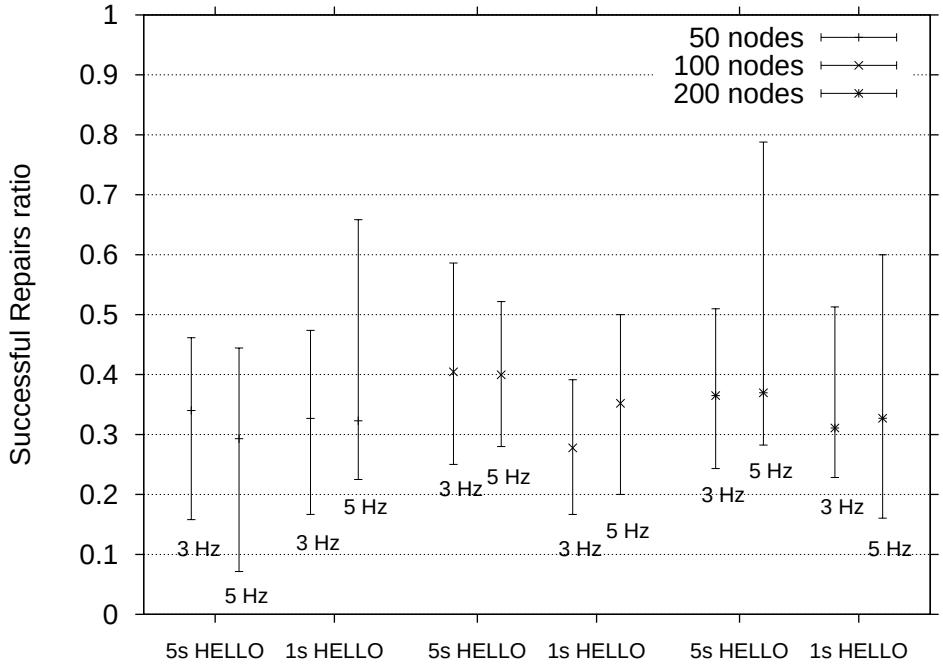


Figure 6.1: Repair ratios of “find bypass in 2-hop cache” approach

6.1.3 Results

From the simulations described above we present only a selection which show best the significant characteristics of each of the three repair mechanisms. The omitted results are similar to the results presented here and give no further insight.

Each of the graphs show the results of one repair alternative. Each graph is separated in three parts for 50, 100 and 200 nodes. The first two values are for a simulation with a HELLO message every 5 seconds combined with 3 and 5 packets (each 512 bytes in size) per second send by an arbitrary source to an arbitrary destination as described before. The next two values are for a HELLO message sent every second combined with 3 and 5 512-byte packets per second payload traffic.

The Y axis shows the ratio of successful repairs to the overall number of link failures. Besides the average over ten runs, the graph also shows the maximum and minimum repair ratios.

Bypass from Routing Cache

In Figure 6.1 you can see the results for the first repair approach. Bypasses for a broken link are searched for in the local routing cache.

The average successful repair ratio lies between 27.8% and 40.5% while it varies between 7% at worst and 79% at best which is a high variability. Varying the frequency of the periodical HELLO messages has only little effects on the average repair ratio. The ratios also seem to stay rather constant when varying the network density (especially when comparing the results for 50 and 100 nodes), although results for dense networks (100 and 200 nodes) are slightly better. This can be explained with the exis-

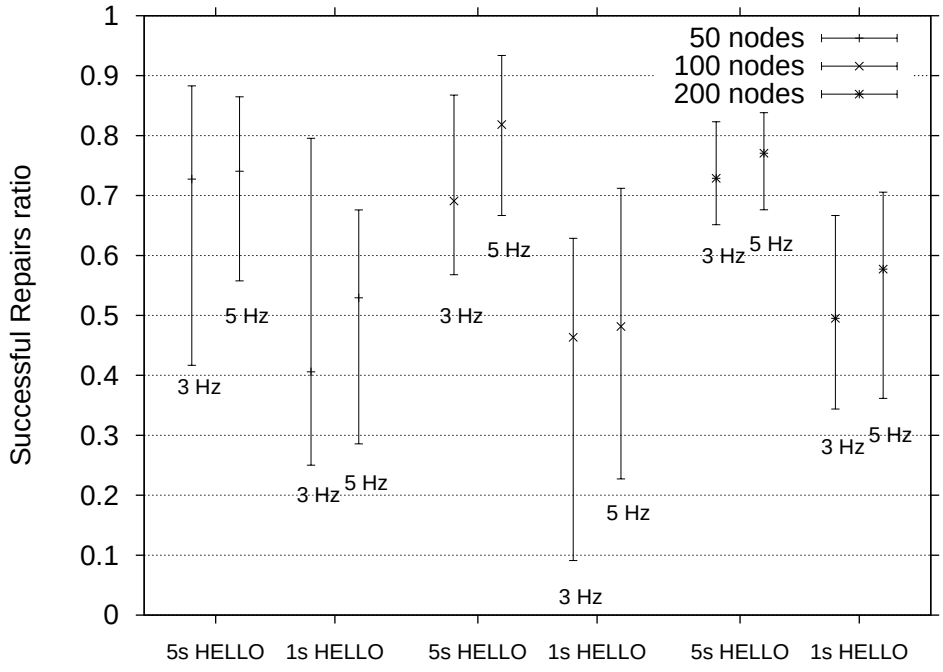


Figure 6.2: Repair ratios of “find bypass in 2-hop cache or wait” approach

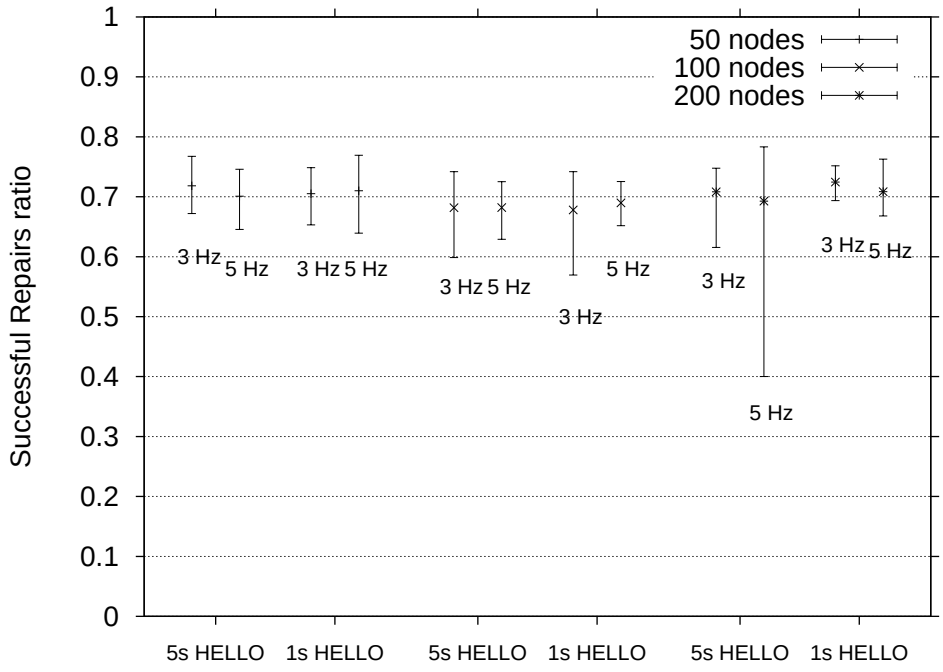


Figure 6.3: Repair ratios of “find bypass by broadcast” approach

tence of alternative paths to a node in the routing cache. Alternatives result in a higher likelihood that some of them are still valid despite the node movement. However, if the HELLO message frequency in dense networks is rather high, the repair ratio is significantly decreased. This is caused by the increased protocol overhead which concerns more nodes than in sparse networks (50 nodes). Congestion and collisions of broadcast (HELLO) messages lead to more link failures. HELLOs get lost and thus the routing cache is not updated which also reduces the ability to repair link breaks.

Bypass from Routing Cache plus Waiting

In Figure 6.2, you see the results for the second repair approach which also uses the local routing cache to find a bypass. Furthermore it waits for new arriving HELLO messages if the most recent alternative path is not fresh enough (cf. Section 6.1.1).

Compared to the first repair approach, it shows a significantly better average repair ratio lying between 40.5% and 82%. But as one can see, the average ratios differ a lot if the periodical protocol overhead is changed: If more HELLO messages per period are sent, the average repair ratio will decrease between 28% and 35%. This is because the time to wait for updated HELLO messages is shorter and therefore the information is not recent enough to overcome the link break. As with the first repair alternative we see more collisions of broadcast packets and hence, lost HELLO messages. By this, recent routing information does not find its way into a node's cache.

The graph also shows a high variability ranging from 22.7% to 93% successful repairs. There is a slightly better result for dense networks (100 and 200 nodes) than for sparse ones (50 nodes). This is logical as in a dense network more alternative paths exist and a higher ratio of them will stay valid despite of node movement. But note that dense networks also suffer more from higher protocol overhead that causes higher network load and thus congestion.

Waiting for routing cache updates helps finding a valid bypass for a broken link and produces significantly better results than the first repair method. However, the very high variability renders this approach problematic.

Limited Repair Broadcast

Finally, Figure 6.3 shows the results for the third repair approach which uses 2-hop limited broadcasts to find bypasses for broken links. As one can see, the average repair ratio lies above 70% in general. This result guarantees a good average repair ratio with very low variability which is also almost independent from the existing traffic and the network density. We have no explanation for the outlier for 200 nodes, but as the average ratio indicates, it is insignificant.

However, this approach is problematic in networks with high loads as it might cause congestion or make it even worse. This effect is of course limited to the 2-hop neighborhood of a node issuing this repair mechanism, and seems to have no significant effects in our configurations.

6.1.4 Discussion

Approach one and two use the periodical HELLO messages to store and find alternative routes. The third one on the other hand takes no advantage of this available information and issues limited broadcasts to find a bypass for the broken link. This causes additional protocol overhead, although only limited to a 2-hop range.

While approach one causes no additional repair delay, the latter two do: The second alternative has a fixed delay of half a HELLO interval if the most recent bypass path is older than this half of a HELLO period. The broadcast approach on the other hand has a variable delay. It is determined by the time to reach the "lost" node and get back the new path towards it. This is limited to a four hop transmission. In case there is no bypass for the broken link available, a timeout places an upper bound on this repair mechanism.

In short, approach one causes no additional delay. Approach two in the best case also causes no delay, the worst case depends on the used HELLO message interval but is the largest (in the order of seconds) out of the three available alternatives. The third approach has a short but variable delay for a successful repair try and a slightly longer but still short delay in the case of an unsuccessful repair attempt.

In a situation of temporary congestion, approach one and two will always assume a link break and increase the path length by at least one by choosing a bypass from their 2-hop neighbor table. The third one, however, can overcome this and acquire the old link again, hence not increasing the path length. However, note that we cannot distinguish between congestion and node mobility with the available mechanism. More sophisticated algorithms are required to detect a congestion situation, but this is out of the scope of this work.

The third repair method can magnify congestion as it consumes additional bandwidth. The other two depend on periodical HELLO messages, and their success rate also depend on the frequency of these messages. Shorter intervals between two of them appear to provide a higher repair ratio while increasing the overall protocol overhead significant. This is the case, as every node periodically sends HELLO messages and an increased frequency increases the overhead in all parts of the network, and not only in a 2-hop range as it does for the repair broadcast mechanism.

In summary, the third repair approach using 2-hop limited broadcasts is the best alternative out of all tested ones for every scenario. It yields a consistently high repair ratio with low variability. Therefore, we use it as the method of choice in the SONAR implementation.

6.2 Evaluation of SONAR

The evaluation of SONAR was split into four parts. First, we ran simulations to determine good parameter values for SONAR that could be used for the further simulation runs. In particular, we wanted to find out if the additional overhead caused by the maintenance of virtual neighbors is justified by the resulting shorter routing paths. Second, we ran SONAR under different network loads. Furthermore, we compared SONAR's behavior with different amounts of mobility ranging from no to high mobility. In the fourth part, we tested SONAR in different network sizes.

6.2.1 Request Generator

User load on the network is simulated by a request generator. Requests are sent from a random source to a random destination. The destination answers with a response packet to the source. The generator consists of two parts: A command-line profile generator and an application-layer simulator module implemented for GloMoSim.

Requests are generated using a Poisson process. The load is varied with mean interarrival time (see resource discovery model in reference [24]). The requests are stored

as tuples (time, source node, destination node) in a file. One can specify an initial cut-off time where no requests are sent. Requests take place starting at the cutoff time until the end of a simulation run to account for the initial warm-up phase in the simulation.

The simulator module reads a specified request profile from a file and schedules all events contained therein. The same request profile can be used for a fair comparison between Chord or any (layer 3) routing protocol. Requests have a size of 512 bytes.

6.2.2 Basic Scenario

We chose a basic scenario to run each simulation in. If not stated otherwise, the parameter values are as follows: There are 50 nodes in the network, the simulation pane is square with a size of $837.41m \times 837.41m$, simulation time is 120 minutes. Nodes move in a random direction model with $v_{min} = 0m/s$ and $v_{max} = 2m/s$ and an epoch time of 300 seconds (which will be called "low mobility" in the following). The transmission range of the 802.11 wireless network interfaces is 250 meters. The request generator was configured with a cut-off time of 3600 seconds and 5 seconds mean inter-arrival time.

6.2.3 Parameter Evaluation

In a first set of simulation runs we wanted to determine good parameters for SONAR. The parameters to determine are

- the interval between two SPS messages in order to update a node's successor entry (which is also the interval between to virtual neighbor updates),
- the maximum allowed sending rate for SPS messages (which also applies to virtual neighbor lookups) and
- whether virtual neighbors should be actively maintained or not.

As mentioned before we used the broadcast repair mechanism as it performed best in our simulations. We fixed all the other SONAR parameters to intuitive good values (some of them are intended by the results from the simulation runs for the local repair alternatives above):

We set the HELLO interval to 5 seconds which is also the period to wait for answers to a bootstrap packet. We used a power-on interval of 50 seconds in which all the nodes randomly joined the network. To detect independent rings, we set the interval for ISPRP floods to 300 seconds. The expiry time for 1- and 2-hop entries in the routing cache was set to 10 seconds which is twice the HELLO interval.

The expiry time for a path entry (both, successor and virtual neighbor paths) was set to $(32 * SPS\_update\_interval)$ which is a necessary duration especially for the virtual neighbor table as for each SPS update message we also update one entry of the virtual neighbor table just as Chord does. If no virtual neighbors are maintained, this parameter should be a lot smaller, as the paths stored in the routing cache will become stale over such a long period. But to allow a fair comparison under similar conditions we chose to use this high value for all simulation runs here, despite whether using virtual neighbor maintenance or not. Especially for high SPS update intervals this parameter value is clearly a bad choice as path freshness is due to node mobility and not the update frequency. We will give more insight on this topic later in this section.

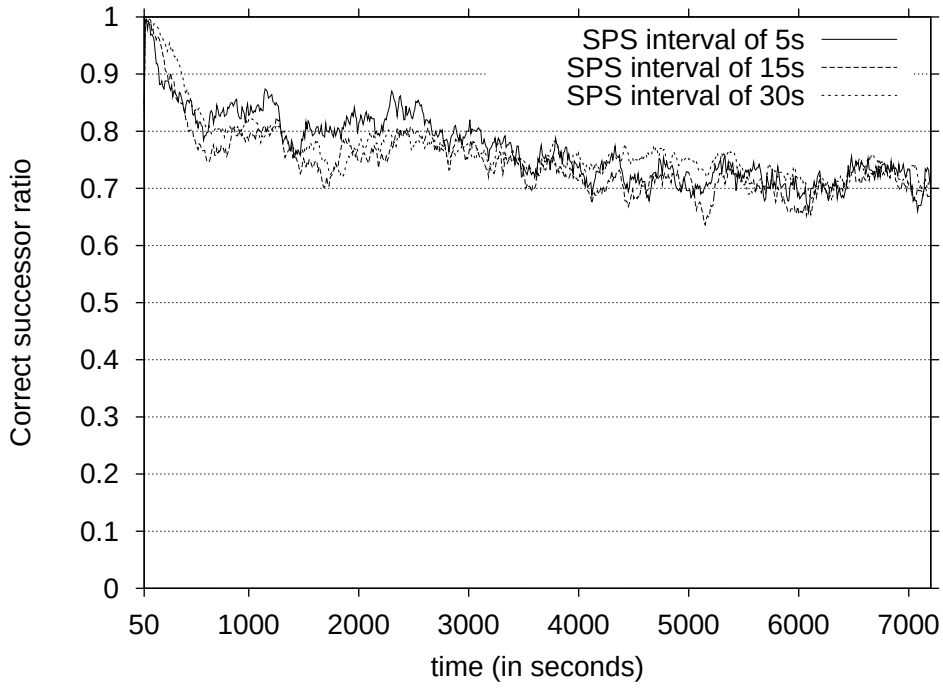


Figure 6.4: Ratio of nodes with a correct successor for different SPS intervals and 3 SPS per second

We tested SPS intervals of 5, 15 and 30 seconds, with the SPS sending rate being 1, 3 and 5 SPS per second. The request generator was configured with a mean inter-arrival time of 5 seconds. The generator initially waited 3600 seconds before sending requests.

We analyze the results with a focus on the ring consistency and the overhead caused by SONAR. Furthermore, we evaluated the average route length to evaluate the benefit of the virtual neighbor maintenance compared to the version without.

Ring Consistency

First we take a look on the ring consistency. Every 10 seconds, we determined the percentage of correct successor relations. We calculated the average development of 10 runs. Monitoring started after the initial start-up phase of 50 seconds, after which all nodes have joined the network. The results indicate, that the consistency is at almost 100% just after all nodes have joined. The development of the ring consistency is not significant before this time, as nodes join randomly and during this time also network partitions can occur. As on average almost all nodes have their correct successor shortly after 50 seconds for all simulations we ran, the ring convergence is very good and new nodes are integrated very fast into the ring structure.

The graphs for the different configurations are very similar and no one shows significant differences to other ones. As a representative example we present the results for an SPS interval of 5, 15 and 30 seconds and an SPS sending rate of 3 SPS per second in Figure 6.4. As one can see, the ring consistency decreases from 100% to approximately 75% and stabilizes on this value.

The results give the wrong impression that the consistency of the ring is independent of the SPS interval. But the very similar results from different SPS intervals can be explained as follows:

A short SPS interval would improve the ring consistency in an environment with unlimited bandwidth because broken links are detected fast and new paths are established within a short period of time. As the bandwidth in MANETs is very limited, a short SPS interval causes congestion thus also an increased number of packet drops, here mainly SPS packets. As a node gets no response to an update request, it will assume that the successor path is broken and will restart ISPRP to find a new path, thus temporarily pointing to a wrong successor (if there is no alternative path to the successor in the first place). This leads to a rather high fluctuation of the ring consistency as ISPRP needs some time to find a new path to the successor and a node will therefore have a wrong one in its routing cache in the meantime.

With a long SPS interval, SPS messages will cause less congestion and updates are more likely to succeed. But broken successor paths will be detected later thus causing additional routing errors when the successor path is actually used. Yet, our local repair mechanism will help to repair path failures and decrease the impact of fewer path updates.

Routing errors occur if a packets ends up at a wrong node and cannot be correctly delivered to its destination. Taking network partition not into account, this means in general, that the node's address is preceding the message's destination address on the ring while its successor succeeds it. Thus, no path to the message's destination is known.

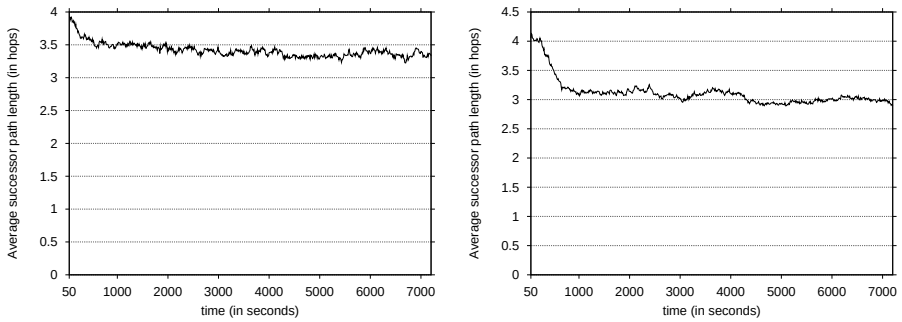
Figure 6.6(a) indicates more successful requests for higher SPS update intervals. And when we take a look on routing failures due to wrong successor pointers, a 15 seconds SPS update interval causes 39% and an interval of 5 seconds even 80% more routing errors than an interval of 30 seconds. This is due to an ISPRP restart after a successor path failure which starts with the next best choice found in the local routing cache (which is mostly not the correct successor if it is more than 2 physical hops away as in this case it is unlikely that the node has an alternative path to it).

Thus, the higher update interval leads to more routing errors which is a result of more detected successor path failures by unanswered update requests. As congestion is the dominating factor for packet losses in our simulations, most of these detected path failures are not due to node movement but a result of congestion. Thus, many ISPRP restarts are unnecessary and instead of improving the routing consistency, it is decreased thus causing additional overhead and ultimately congestion.

With enabled virtual neighbor maintenance, one would expect more overhead thus increased probability for congestion. However, we simulated 50 nodes with an address length of 32-bit. This results in most of the virtual neighbors entries being populated by the successor and thus not needing any additional updates. Obviously the effect in a network with more nodes and hence more "real" virtual neighbors will result in significantly more overhead. As we update an entry with every successor update, the overhead caused by the SPS messages is doubled in the worst case and congestion will be significantly increased.

Successor Path Length

The average successor path length is another point of interest. As the results for with and without virtual neighbor maintenance are very similar, we only analyze the results without it as it makes no difference. Figures 6.5(a) and 6.5(b) show the development



(a) SPS interval of 5 seconds and 3 SPS per second

(b) SPS interval of 30 seconds and 3 SPS per second

Figure 6.5: Average successor path length

over time for an SPS interval of 5 and 30 seconds. Unfortunately the results are not representative for the average successor path length since we only see the successor path lengths for nodes with their correct successor (we will present representative results for no mobility in Section 6.2.4).

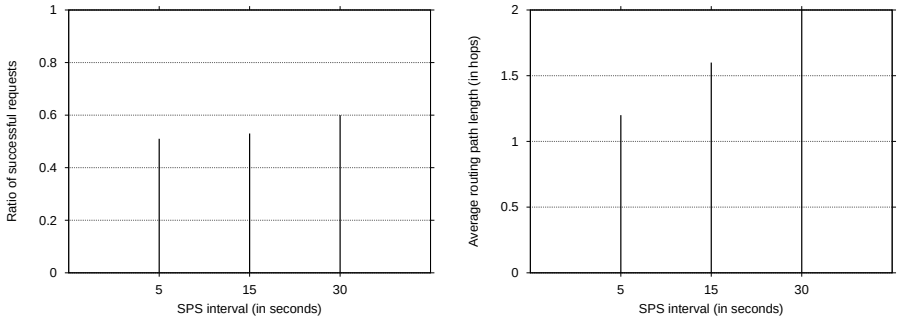
What we can derive from the two graphs is, that for a configuration with an SPS interval of 5 seconds the gradient is uneasier than the one for 30 seconds. This is so as ISPRP is often restarted due to broken successor paths (mainly caused by congestion in this case) and it takes some time to optimize the path again. The graph stabilizes around 3.35 hops. But note that longer paths are more likely to break than short ones and thus do not contribute to the average.

With a shorter SPS update interval the successor path length appears to converge faster to the optimal shortest path. This is due to the distributed path optimization. In the used implementation, paths are only optimized when sending an SPS message. To speed up path optimization, this can be extended to ACK and NACK messages. However, this will slightly increase the overhead as every ACK and NACK message would carry an additional field for the optimized path. But we think this is a better alternative than using a high SPS update frequency which leads to congestion and a more fluctuation ring consistency. However, as said before, we only use the distributed optimization routine for SPS messages in these simulations.

Impact of the SPS Interval

As we presented before, a short SPS interval does not lead to a significant faster ring convergence than for long intervals. This is mainly due to congestion, as other statistics indicate.

We see an increase in packet drops when using short SPS intervals. An interval of 15 seconds incurs 10% less packet drops than 5 seconds, 30 seconds almost 30%. This is correlated to the overhead caused by the SPS messages, thus resulting in congestion. Furthermore, the ratio of correctly repaired link breaks is increased with a decreasing SPS interval. This is logical as the broadcast repair mechanism will not work well in congested environments as mentioned before. Another indicator is the ratio of successful requests by the request generator that can be seen in Figure 6.6(a).



(a) Ratio of requests to responses generated on the application level for different SPS intervals and 3 SPS per seconds

(b) Average routing path length for data messages for different SPS intervals and 3 SPS per seconds

Figure 6.6: Impact of the SPS interval on data requests

The ratio increases for long intervals which is again a result of congestion caused by SPS messages. Furthermore, routing errors caused by incorrect successor pointers due to ISPRP restarts are another reason for the bad performance for high SPS update intervals as described before.

Figure 6.6(b) shows the average routing path length for data packets for an SPS interval of 5, 15 and 30 seconds. As one can see, the average path length is really short which is mainly due to the network density, also only successful end to end transmissions were taken into account. Furthermore we see a decreasing path length for decreasing SPS intervals. As routing is done in a greedy fashion, the node closest to the destination is chosen as a next hop. As we did not vary the HELLO message interval which updates the 1-hop knowledge of a node, this is a pure result of the changed SPS interval and can be explained with the distributed optimization method done in every SPS packet. A shorter SPS interval leads to more optimization runs and thus the path converges faster to the shortest available path. This effect can be observed when a stored source path is used, which happens mainly when the successor is chosen as next hop. To moderate this effect, one can do the optimization also in ACK and NACK packets as said before.

Impact of the SPS Sending Rate

Especially in the initial startup phase, a lot of SPS messages are sent in order to find a node's correct successor. To limit the impact of this, we implemented an SPS sending rate limitation. In the worst case it takes longer to find the correct successor. However, without this limitation ISPRP can cause congestion, especially when a new node joins the network due to SPS bursts. In normal maintenance mode, the limitation will have no impact on the overall performance. Only when actually searching for a successor, the limitation will slow down the process but thus also avoid temporary congestion.

As SPS messages are responsible for most of the protocol overhead, a high sending rate leads to more congestion, however limited to a short period. Furthermore, it shares similar characteristics with the SPS interval itself as can be seen in Figure 6.7. The ring consistency is slightly better for low SPS sending rates. This is because a high sending rate leads to bursts of SPS messages thus resulting in temporary congestion

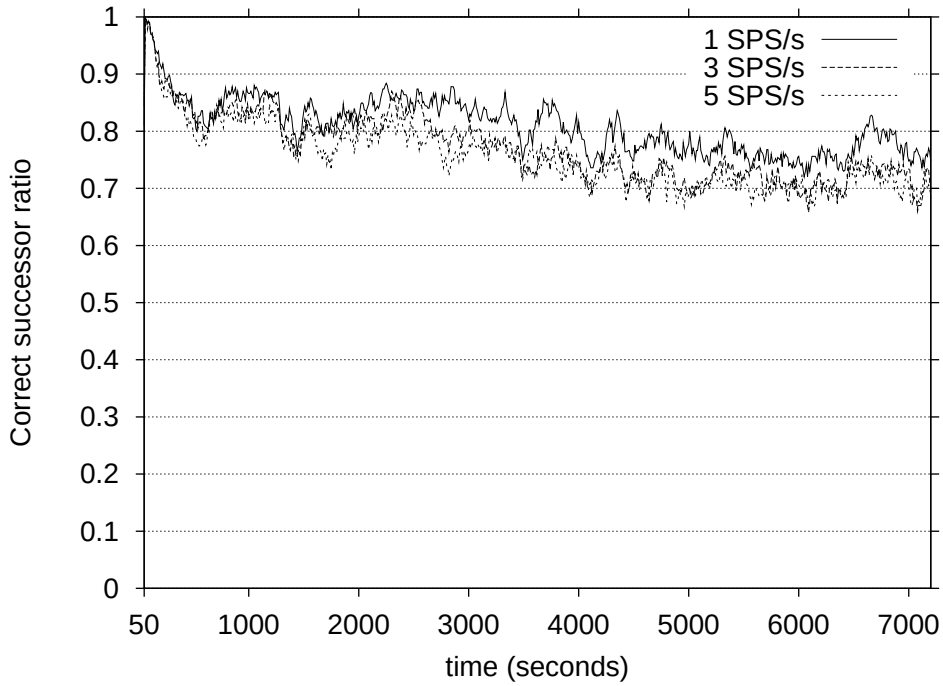


Figure 6.7: Ratio of nodes with a correct successor for different SPS sending rates and SPS interval of 5 seconds

which is also emphasized by other statistics: Comparing a sending rate of 1 and 5 SPS per second, the lower rate sees 50% less packet drops and 50% less protocol overhead (which is 1.01 KB/s on average per node for 1 SPS per second and 2.07 KB/s for 5 SPS per second not including overhead caused by source route headers for data packets). Furthermore we observed 25% less routing errors are caused by false successors. Also, the transmission ratio – which is the ratio of successful requests – is 15% higher for 1 SPS per second.

The results show that SPS bursts cause significant, yet temporary congestion. Instead of improving the ring consistency and speeding up the convergence of the ring, it has a converse effect. Allowing only 1 SPS per second leads to less SPS messages until the correct successor is found. This can be explained with node movement and routing cache updates that offer a wider collection of nodes to choose a better successor from.

Virtual Neighbors

The problem with virtual neighbor maintenance in MANETs is multi-layered. If we use the same number of virtual neighbors as in Chord this will lead to 31 paths to maintain (virtual neighbor 0 is the successor which is already maintained). The probability for a path to break depends on the node mobility, more concrete on the mobility of each member of the path. To preserve path freshness we rely on update messages whereas a high frequency of these will detect link breaks faster. But as the bandwidth in MANETs is very limited, keeping all paths up to date is very expensive and leads to a low update frequency. However, this renders the usage of an actively maintained virtual neighbor table ineffective as most of the entries will be stale due to node mobility.

There are modifications of this functionality that can show good performance in MANETs which we will present in section 7.

We analyze the impact of virtual neighbor maintenance for a configuration consisting of an SPS update interval of 30 seconds and 3 SPS per second (the results for other SPS intervals are similar). The protocol overhead is increased by 4.9% while the routing overhead caused by source route headers is even 8.4% higher. The latter value indicates that SONAR benefits from the virtual neighbor maintenance and uses them for routing. Furthermore we see 1% less routing through a node's successor which is again due to the usage of virtual neighbors.

However, the increased overhead also leads to 14% more packet drops for this configuration (up to 30% more drops for shorter SPS update intervals). The average hop count for routed data decreases with virtual neighbors from 1.2 to 1.1 hops. As we expect significantly more overhead for bigger networks than our 50 nodes network used for these simulations, the cost does not justify the result.

Discussion

The simulation shows that the overhead caused by the regular SPS messages is significant and the SPS interval should be long to minimize the probability for congestion. This will not decrease the ring consistency and even result in a shorter average successor path than for a shorter interval as we avoid unnecessary ISPRP restarts after updates failed due to congestion. Thus, we choose an SPS interval of 30 seconds for the following simulations.

A high SPS sending rate leads to bursty overhead traffic which can lead to temporary congestion. Yet, a high rate slightly improves the duration for the ring to converge. In the normal maintenance mode, this is no problem at all as the SPS update interval is a lot longer than the allowed sending rate. We chose to use an SPS sending rate of 3 SPS per second in the following which will allow a fast ring convergence while limiting the bursts of SPS messages.

As the virtual neighbors maintenance causes more overhead while resulting in only slightly shorter routing paths, we decided not to use it for our further simulations. We already benefit from the 2-hop knowledge provided by the regular HELLO messages which results in good routing performance. Furthermore, by this we also exploit locality and thus short physical paths.

Improving the reliability of routing is a significant problem. Increasing the protocol overhead to overcome link breaks caused by node mobility is a very bad idea as it directly leads to congestion which again increases packet drops (links appear to be broken but in fact they are just not useable due to congestion). To find the balance between causing serious congestion and fast recovery from node movement is extremely hard and we found no "best" solution so far. However, our parameter choices seem to perform at least well and might be a good starting point for further research and improvements.

6.2.4 SONAR in Different Mobility Scenarios

In these simulation runs we tested SONAR in three different mobility scenarios:

- *No mobility*,
- node movement according to a random direction model with $v_{min} = 0m/s$ and $v_{max} = 2m/s$ and an epoch time of 300 seconds (called *low mobility* in the

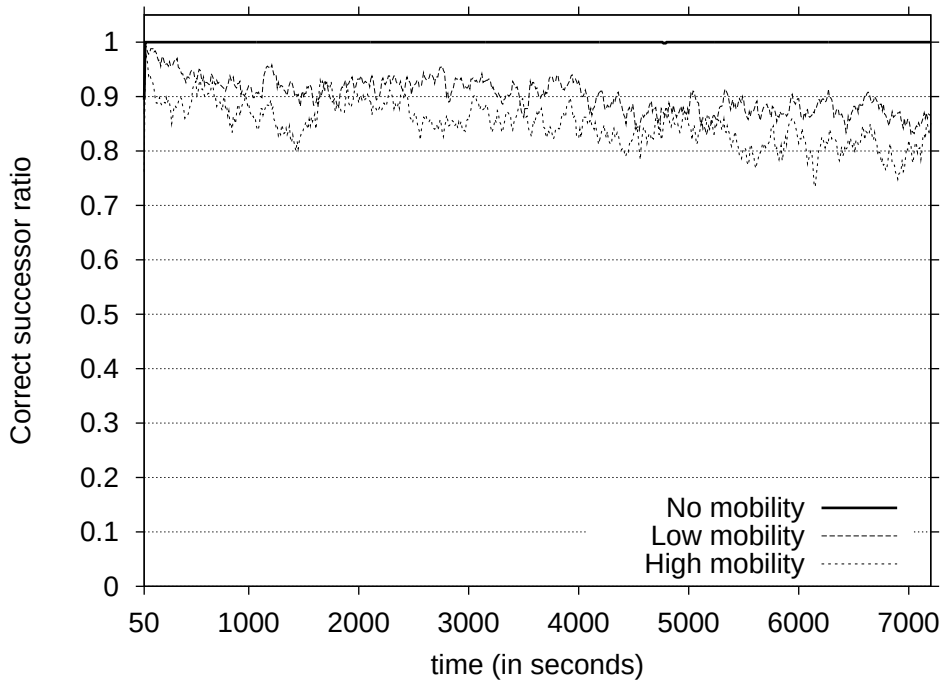


Figure 6.8: Ratio of nodes with a correct successor for none, low and high mobility

following) and

- node movement according to a random direction model with $v_{min} = 0m/s$ and $v_{max} = 5m/s$ and an epoch time of 300 seconds (called *high mobility* in the following).

We used mainly the same configuration as in the first simulation: There are 50 nodes in the network, the simulation pane is quadratic of size $837.41m \times 837.41m$, simulation time is 120 minutes.

We set the HELLO interval and the time to wait for an answer for a bootstrap packet to 5 seconds. The power on interval is 50 seconds, ISPRP floods are sent every 300 seconds, the expiry time for 1- and 2-hop entries in the routing cache was set to 10 seconds which is twice the HELLO interval. As said before, we use an SPS update interval of 30 seconds and an SPS sending rate of 3 SPS messages per second. The path timeout was set to $(2 * SPS\_update\_interval = 60s)$.

The request generator was configured with a mean inter-arrival time of 5 seconds. The generator initially waited 3600 seconds before sending requests.

Figure 6.8 shows the average ratio of nodes with correct successor of 10 runs each. For no mobility the ratio of nodes with correct successor is short after the power on time at 100% and stays there. For low mobility, it is at 99% after 50 seconds and stabilizes around 86%. For high mobility the ratio decreases from 97% just after the power on interval to about 80%. Table 6.1 shows statistics for the different scenarios.

	No mobility	Low mobility	High mobility
Control traffic (per node)	0.087 KB/s	0.70 KB/s	0.87 KB/s
Overhead for SR headers (p. node)	0.14 KB/s	1.06 KB/s	1.42 KB/s
Successful repair ratio	95.289 %	36.622 %	31.750 %
Successful requests ratio	99.985 %	73.010 %	57.309 %
Average hops to destination	2.70	2.22	2.00
Av. number of SPS (p. node)	534.12	3965.34	5131.28
Av. number of SPS-ACK (p. node)	532.03	581.19	432.66
Av. number of SPS-NACK (p. node)	3.34	1250.98	1676.41
Av. number of RERR (p. node)	0.12	1451.47	1921.67

Table 6.1: Comparison of none, low and high mobility. The numbers for SPS, SPS-ACK, SPS-NACK and RERR messages also include forwarded messages.

No Mobility

For no mobility the overall overhead is at $0.23KB/s$. Thus, the successor path stays valid over the simulation time and there is no need to restart ISPRP to find a new path which leaves the overhead low. For 10 runs we see only 524 link failures overall and as there is no node movement these must be due to congestion; the broadcast repair mechanism should be able to repair all of them. But in 5% of the repairs the congestion is still in place and thus repair fails. The average round-trip time for requests is at 29.3 ms which can be assumed as the minimum for our simulations as we see the lowest overhead and thus the lowest bandwidth utilization in this configuration.

Low Mobility

In the scenario with low mobility we see an increased protocol overhead as SONAR needs additional SPS messages to repair broken successor paths. And as node movement causes many link failures, repair attempts also increase the overhead. However, the successful repair ratio is only at about 36%. If we compare these results to the one presented in Section 6.1, this is mainly a result of congestion that prevents the repair broadcasts from reaching its destination.

The successful requests are at 73% which means that a little more than 25% of the requests are unanswered either due to routing errors or link breaks resulting in the request or the respond being dropped. The average routing length of requests and responses is about 0.5 hops shorter. As the average distance between two arbitrary nodes should stay the same even with node mobility (which is because all nodes move randomly), this is a result of the unsuccessful requests. We only count data messages that reach their destination. As messages with a long routing path are more likely to get dropped due to link failures, the average counted paths appear to be shorter. The average transmission latency for data messages is doubled compared to no mobility.

High Mobility

For high mobility where nodes move with up to $5m/s$ we see of course a lot more link failures of which only about 32% are successfully repaired. The high node mobility causes successor paths to break with a rather high frequency and SONAR to restart ISPRP to find a new path to the successor. This is the reason for the increased overhead which also causes congestion and is worse than for low mobility. With an increased

maximum velocity, there is also a bigger chance that nodes are out of two hop range when the link break is detected and repairs cannot succeed.

The successful requests are at 57%. The average hops for a request to reach its destination is again shorter than for the no-mobility scenario which is due to the same phenomenon described for low mobility. The latency for data messages is only 1 ms higher than for low mobility. This is mainly as messages with short paths are more likely to reach their destination.

In a Rectangular Simulation Plane

In a second configuration we run these three different mobility scenarios in a simulation area of $1500m \times 300m$, as it is also used for example for evaluating DSR [18]. The field is now 300m wide at a transmission range of still 250m. The new simulation plane results in two things: Longer routing paths and more congestion, as almost all messages have to cross the same areas of the medium.

Comparing the results from the squared and the rectangular simulation plane, successor path increases by 0.5 hops on average independent from the mobility scenario. The routing length for data packets increases by even 1.5 hops for no mobility. If we look on the results with node mobility, we see an increase in overhead and link breaks. Only 50% of the requests got answered.

Comparing the results for the square and the rectangular plane, we see 50% more link breaks and also the overall overhead increases by 50% per node. Such a slim network shape sets SONAR under pressure as almost all messages have to cross the same network area as the radio range of 250 meters is almost the same size as the simulation plane width. Keeping the protocol overhead as low as possible is the only way to get good results for this configuration.

Discussion

The purpose of these simulation runs was it to observe the behavior of SONAR in different mobility scenarios. To handle mobility well was a central goal of our design of SONAR. However, we can never avoid link breaks and packet drops due to node movement. Therefore, SONAR worked quite well for different mobility levels. Nevertheless, the results intend to look for further optimizations to improve routing reliability.

The results also show the deep impact of the limited bandwidth. It is very easy to cause congestion in the network. To overcome link failures caused by high mobility, SONAR consumes more bandwidth which again causes link breaks – but this time due to congestion. These results hint that the used repair mechanism is a good point for further improvements as it causes additional congestion and thus also packet drops. Techniques to lower the overall protocol overhead will also help to further improve the results, although they are already quite good. We will take a closer look on this in Section 7.

In Figure 6.9 we take a look on the average successor path length for no mobility as only in this scenario we can observe the impact of the SPS sending rate and the path optimization routines. It takes 50 seconds to converge to the shortest path achievable by the used mechanisms which is 3.63 hops on average.

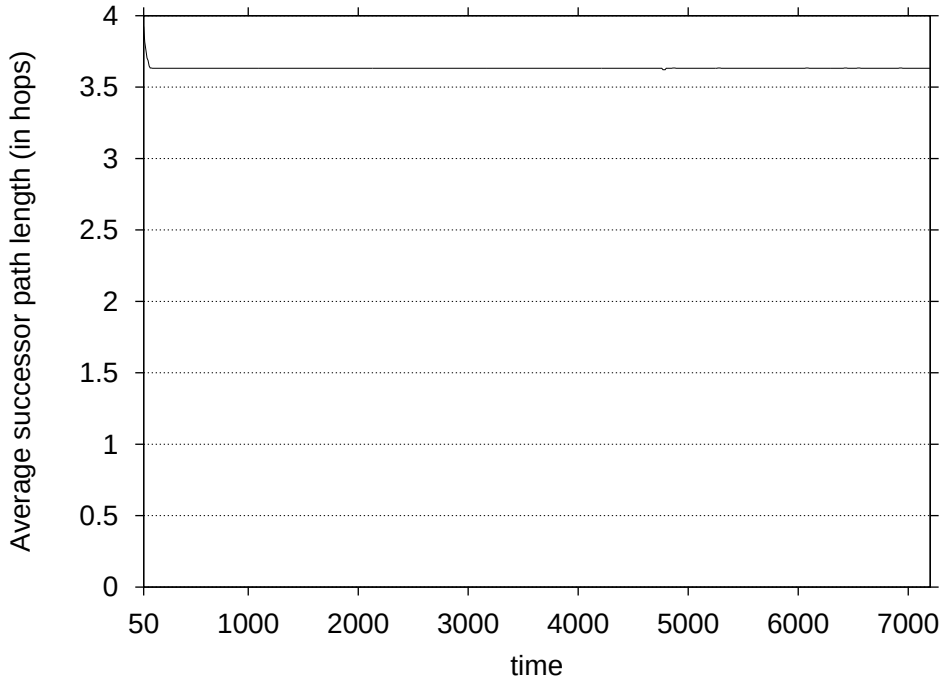


Figure 6.9: Average successor path length (in hops) for no mobility

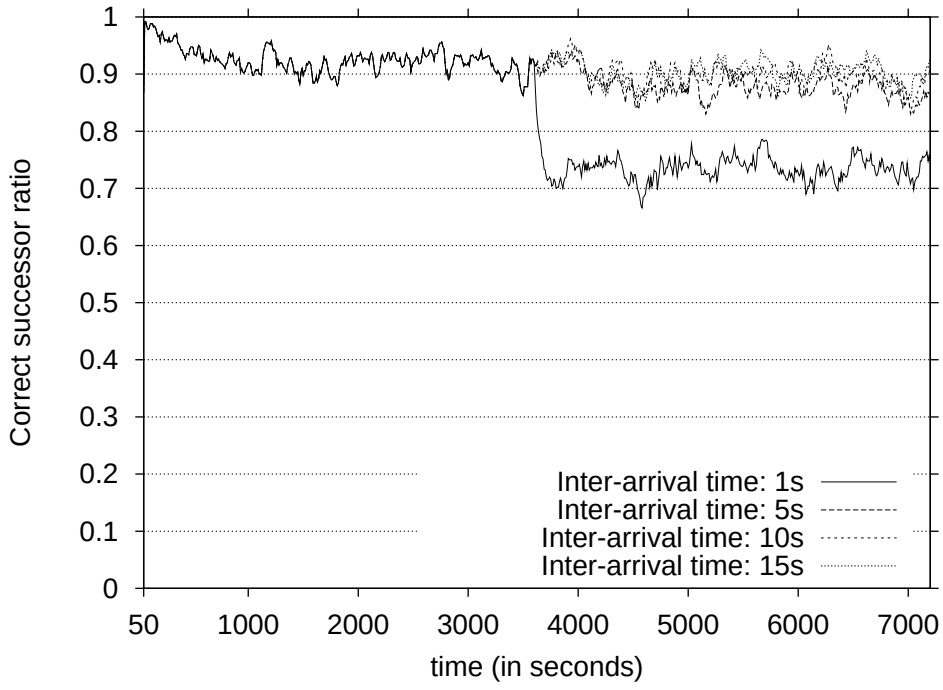


Figure 6.10: Correct successor ratio for different network loads

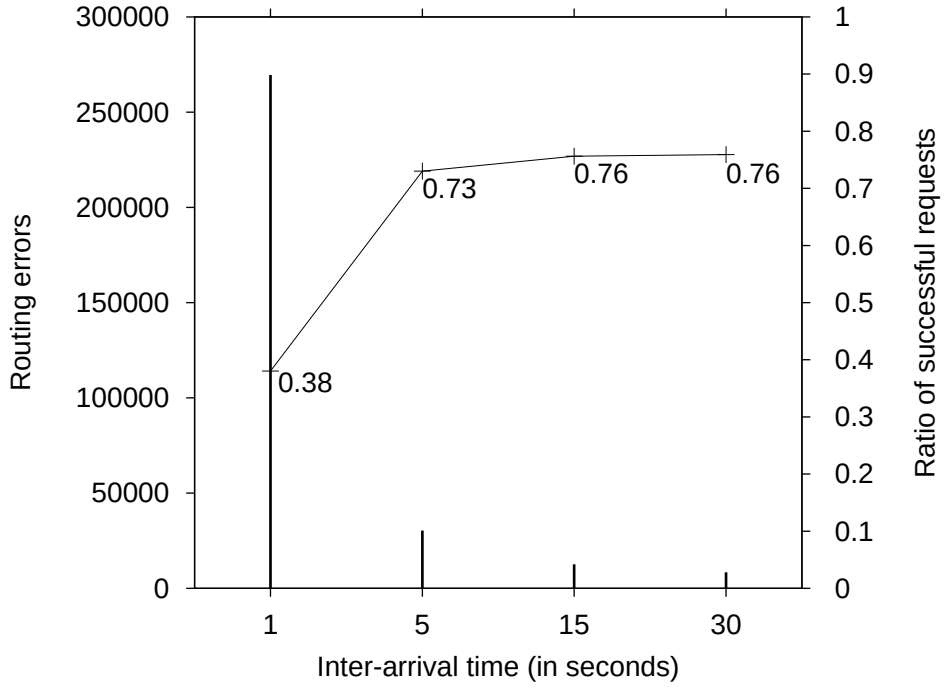


Figure 6.11: Absolute numbers of routing errors caused by wrong or missing successors and the ratios of successful requests for different network loads

6.2.5 SONAR under Load

In the next simulation, we evaluated SONAR with different network loads. The request generator was configured with an inter-arrival time of 1, 5, 10 and 15 seconds. The lower the inter-arrival time, the higher is the caused network load – which is about 1KB/s per node for 15 seconds and more than 8 KB/s per node for 1 second on average. We used the low mobility scenario for these simulation runs. The rest of the parameters are the same as for the mobility simulation.

Again, we take a look at the ring consistency in figure 6.10. The gradients for an inter-arrival time of 5, 10 and 15 seconds are very similar and lie between 86% and 91% on average. However, for an inter-arrival time of 1 seconds, which is the highest load we tried, we see a break-in at about 3600 seconds which drops the ring consistency to 75%. This is a result of the request generator starting at 3600 seconds. The congestion caused by this has a negative impact as SPS update messages do not succeed and SONAR assumes that a still existing link is broken. Of course, due to congestion repair attempts also fail and ISPRP is restarted to find a new path to the successor.

Looking at the number of failed packets, the counts for 5, 10 and 15 seconds are almost the same, while being doubled for 1 second inter-arrival time. Almost the same is true for the successful repair ratio, which is at 15% for 1 second and 37% for the other configurations. Figure 6.11 shows that up to a network load of 3 KB/s per node (down to 5 seconds inter-arrival time) the ratio of successful requests stays almost constant. This can also be seen for the routing errors and the average round trip time which is around 30 seconds. For a high load of more than 8 KB/s per node (for 1 second

inter-arrival time) the successful request ratio drops to 38% and the routing errors are increased by a factor of 9. The request latency is more than 120 times higher for 1 second inter-arrival time than for longer values where we see latencies of about 30 ms.

In a Rectangular Simulation Plane

We again ran the same simulation again in a simulation plane of 1500m x 300m. The performance decrease is significant for 5 to 15 seconds of inter-arrival time. Here we see a double in link failures – obviously due to congestion – and also the control traffic almost doubles. Furthermore 15% less requests got through successful. The overhead caused by source route headers increases by 50%.

However, for 1 second inter-arrival time, the increase is still there, but as the performance of SONAR was also bad for the square simulation area, the further decrease is only half as bad as for the other configurations.

Discussion

These simulation runs proof that SONAR works quite well for a wide range of network loads. For this scenario SONAR works well under a load of 6 to 7 KB/s per node. This is also a result of the high network density. within the transmission range of 250 meters we see 12 nodes on average. In difference to wired networks these nodes share the same medium within their transmission range which limits the bandwidth per node significantly. A DHT layered over SONAR should also provide some kind of load balancing for routing as the node addresses are randomly distributed over the address space. However, this is destroyed by the usage of local shortcuts from the 2-hop knowledge and even more by the shared medium. It is an open question whether we can find a solution for this problem.

6.2.6 SONAR in Different Network Sizes

In this simulation we evaluated SONAR in different network sizes, ranging from a network with 10 nodes up to one with 100 nodes while the network density stays the same as the simulation area grows with the same ratio starting with $374.5m \times 374.5m$. The request generator was configured with an inter-arrival time of 5 seconds and nodes moved in a low mobility scenario as described in the mobility simulation. The rest of the parameters are the same as for the former SONAR simulations.

Successor

Figure 6.12 presents the average ring consistency for the different network sizes. As one can see, for 10 nodes we achieve almost 100% consistency on average and stays beyond 90% for up to 50 nodes. This can be explained with a node's 1-hop knowledge. In small networks almost all nodes are in the local 1-hop or at least in the 2-hop cache. This leads to short paths and mostly HELLO messages will update the successor path and additional SPS messages can thus be avoided (however, as new nodes might join the network, a former correct successor can become wrong. Thus we allowed one update to be left out before sending an update SPS message even if the path was verified in another way before). Furthermore there will be several alternative paths to the successor stored in the 2-hop cache which can be used if the recent used one fails. The gradient almost linearly decreases from 50 nodes to 56% for 100 nodes.

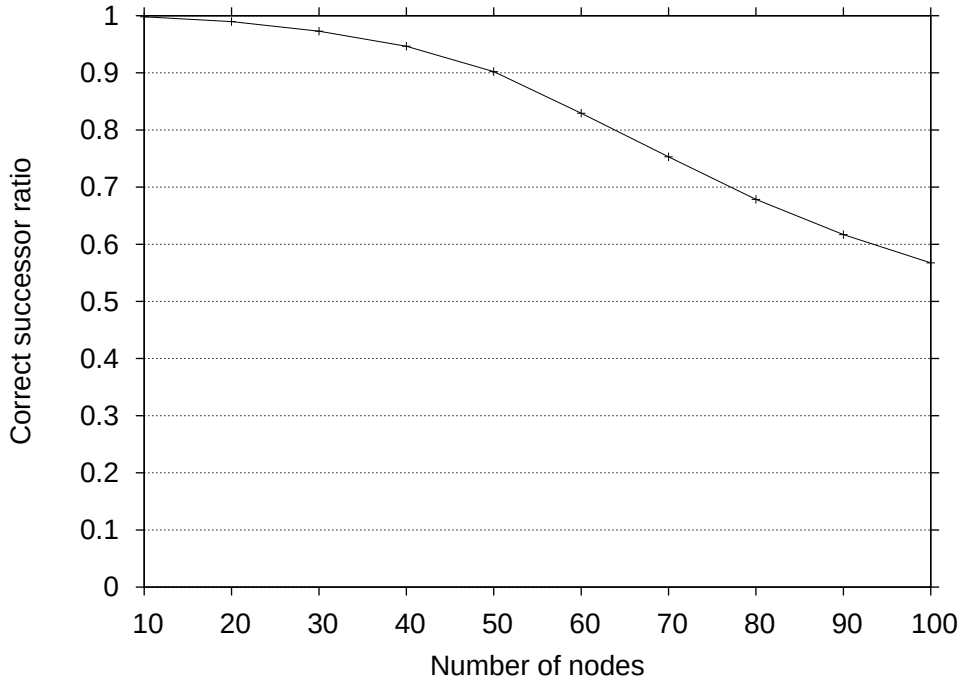


Figure 6.12: Correct successor ratio for different network sizes

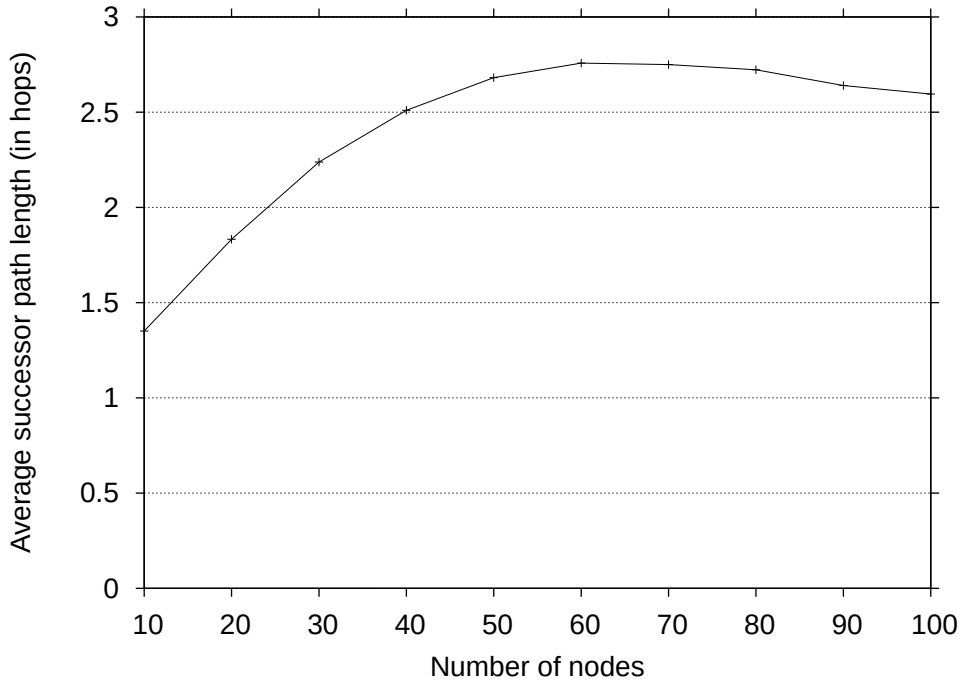


Figure 6.13: Average successor path length for different network sizes

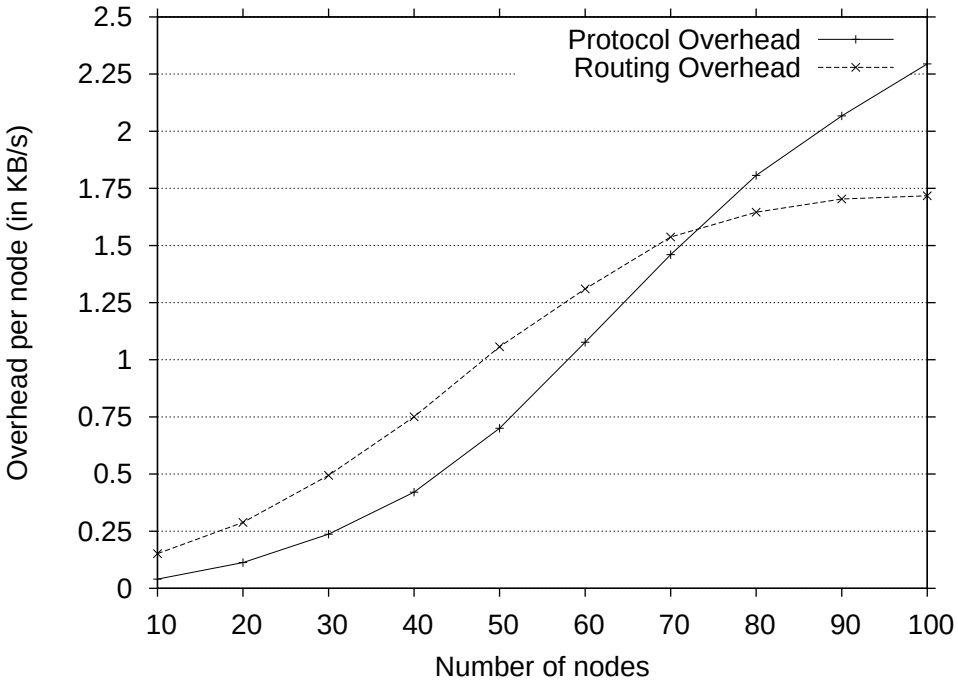


Figure 6.14: Overhead per node in KB/s divided into overhead caused by SONAR's protocol routines and overhead caused by source route headers for data packets

Corresponding to that, Figure 6.13 shows the average successor path length. We only took correct successor paths into account as incorrect ones appear to be shorter on average as a node will choose an alternative successor from its physical neighborhood if SPS updates fail.

For very small networks we see very short successor paths as apparently the simulation pane is only slightly bigger than the transmission range of a node. Almost all nodes are within an one hop range, on average in the 10 node network each node has 6 other nodes in its 1-hop neighborhood. For 20 nodes we see an average of 8 and for 30 nodes of 9 nodes in each node's 1-hop neighborhood. Starting with a network of 40 nodes this value stabilizes around 11 nodes.

Figure 6.13 shows a climax of the average successor path length for a 60 node network with 2.75 hops. After that point the path length seems to stabilize just above 2.5 hops. This does not indicate that we only have such short successor paths (however it is possible due to used network density), as especially for a high number of nodes many successor paths are not taken into account as they are wrong. Furthermore, the numbers we present here are only snapshots taken every 10 seconds before calculating the average path length. But the numbers tell us that especially short paths stay valid which is no surprising result.

Overhead

Figure 6.14 shows the bandwidth consumption of SONAR split up into protocol overhead which consists of the ISPRP messages, repair broadcasts and HELLO messages and the routing overhead caused by source route headers attached to data packets. The

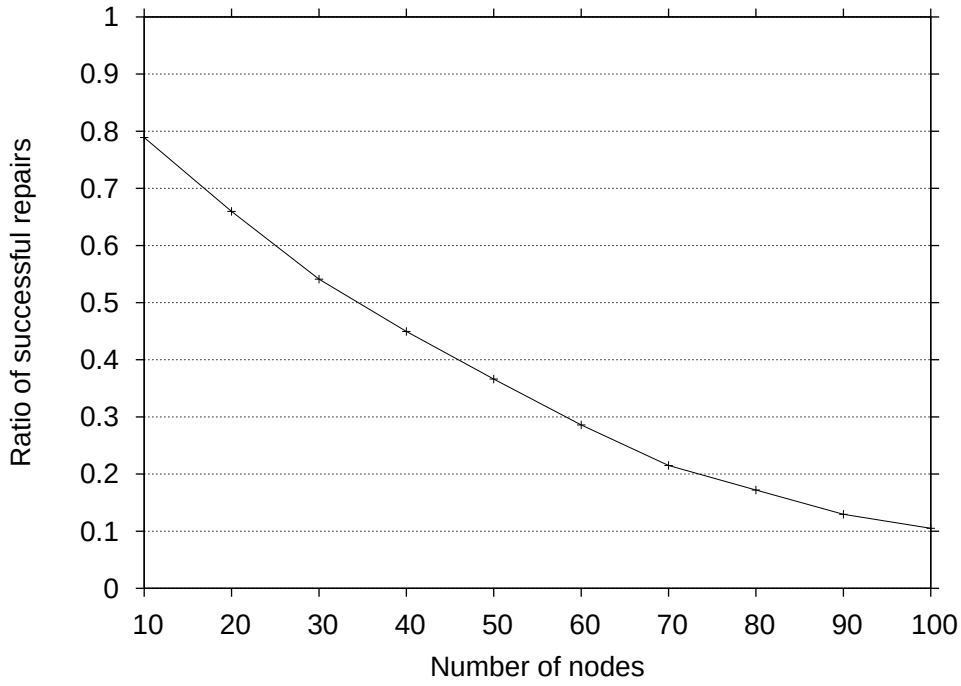


Figure 6.15: Ratio of successful repairs for different network sizes

graph shows that up to 70 nodes the routing overhead is dominating, but converging at 1.75 KB/s. However, the protocol overhead consumes for small networks only little bandwidth while the overhead increases almost linearly from 50 nodes on.

This is caused by congestion which results in more packet drops and so in more frequent SPS messages. As each message is routed the shortest available path the network is not loaded equally and traffic hot spots appear. As the protocol does not include mechanisms to route around such hotspots there is not much we can do about it. However, SONAR does not seem to scale well with the number of nodes for dense networks.

This is also indicated by the repair ratio as it can be seen in figure 6.15. For small networks the number of successful repairs is rather high. This is accompanied with a steady increase in link breaks. But the bigger the network gets, the lower the successful repair ratio is. As the node mobility is not changed, this must be a result of congestion which is further increased by the broadcast repair mechanism.

Data Traffic

For small networks we see almost 100% of successful requests this decreases for bigger networks, down to 30% for 100 nodes as can be seen in Figure 6.16. This has the same reasons as mentioned for the increased overhead and failure ratio.

As the results for routing length only take into account successful end-to-end transmissions, they are especially for bigger networks with a high fraction of unsuccessful requests not meaningful. For 30 to 50 nodes they range slightly above 2 hops. This indicates that we benefit a lot from the 2-hop knowledge in every node.

Figure 6.17 shows the average round-trip time for requests. The latency stays below

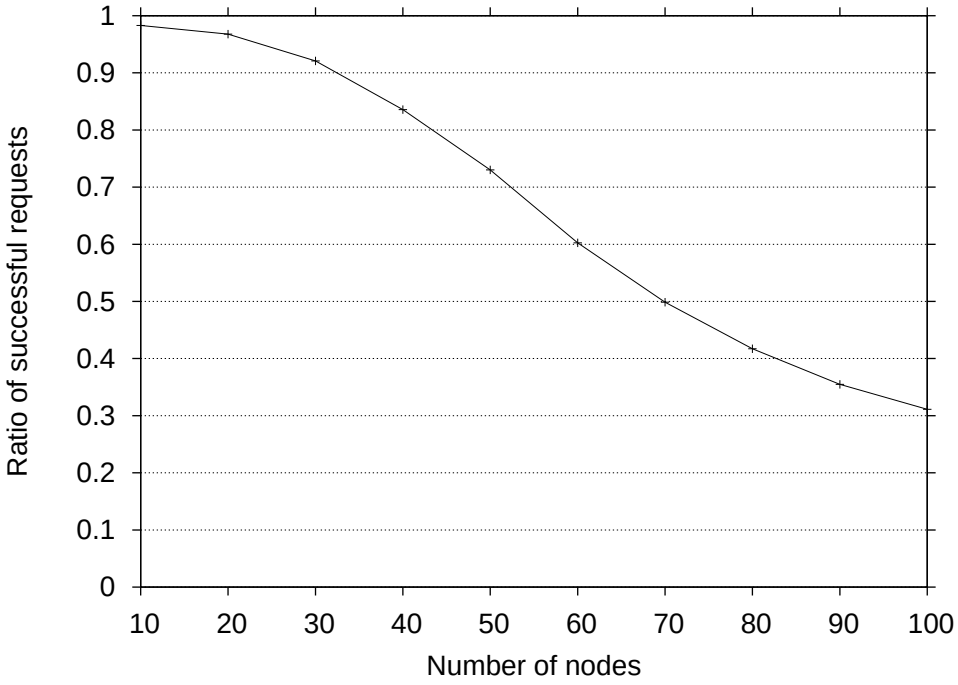


Figure 6.16: Ratio of successful requests for different network sizes

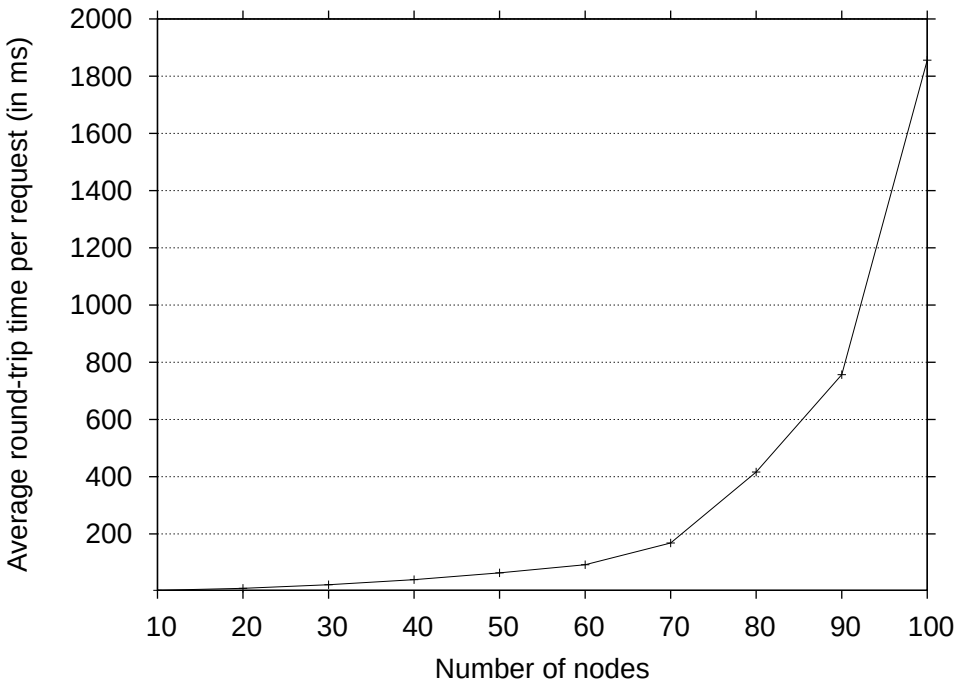


Figure 6.17: Average single-trip latency for requests for different network sizes

100ms until the network size exceeds 60 nodes. After this point the latency increases exponentially. This is a result of the increased overhead for big networks which results in longer delays for accessing the medium.

Discussion

This simulation shows that SONAR is very good in finding a short path towards a destination. However, the ring consistency and routing correctness is not good for networks above 50 nodes. This is so, as in our simulations the network load increases with the size of the network, as the requests are generated per node. In bigger networks more nodes are sending requests, thus increasing the overall network load. This results in saturated links due to the user traffic which causes congestion.

Due to SONARs aggressive repair strategy, this results in frequently detected link failures (and the repair mechanism works poor in already congested environments) and thus new successor lookups by ISPRP. In general, there is no alternative successor path in the local routing cache. Thus a new successor from the local routing cache is chosen and it takes some time until ISPRP finds a new path to the correct successor.

Chapter 7

Conclusion and Future Work

MANETs and P2P networks share some common characteristics: Both are self-organized, decentralized and autonomous. Using P2P techniques to build MANET protocols therefore seems to be a viable approach.

The bandwidth in wireless networks is low compared to wired networks and also packet drops are more frequent due to noise and interference. To be able to route packets and provide multi-hop communication, nodes have to exchange routing informations which consumes additional bandwidth. Thus, limiting the protocol overhead is an important issue. As also a P2P system offers routing mechanism, we combined both, the ad hoc routing protocol and the P2P network, into one cross-layer protocol, thus keeping the overhead low.

SONAR is built around ISPRP, which is responsible for the efficient establishment and maintenance of a structured P2P network. The goal of this work was to enable ISPRP to cope with the MANET environment and to support timeouts to detect and react on packet loss. Periodical updates are introduced to detect stale paths.

To handle node mobility and the resulting route breaks, we employed a 2-hop repair broadcast strategy. In general, a node that has left the physical neighborhood should still be in a 2-hop range and with high probability, this repair mechanism will find a path to it. However, this mechanism does not work well under congestion, as it causes additional traffic and can make congestion worse.

Furthermore, nodes exchange information about their direct physical neighbors which results in 2-hop knowledge in every node. By this, we can exploit locality, especially for routing purposes, but also for optimizing source routes. To shorten the source routes given by ISPRP, we use a distributed optimization mechanism, where each member of a source route uses its local knowledge to optimize a path.

7.1 Performance

Simulations show that SONAR has good performance in different mobility scenarios. The routing consistency is almost not affected by increased node movement, while the control traffic stays rather constant.

Also, increased network load does not affect the performance of SONAR up to a certain point of about 5KB/s per node, where the links get saturated and congestion causes excessive packet drops. This furthermore results in a decreased ring consistency, as SONAR aggressively tries to find new successor paths since the packet drops are

interpreted as link breaks. Not having alternative successor paths in the local routing cache, a new successor path has to be discovered. This process significantly increases the control traffic.

We discovered that while SONAR handles node mobility very good, it cannot adapt to congestion at the moment. Decreasing the control traffic could help to improve the situation. Furthermore we discovered, that source route headers for data messages consume a significant amount of bandwidth. Therefore, SONAR can significantly benefit from the avoidance of source route headers. We describe an approach to avoid them later in this section.

For networks of size 50 nodes we see the number of correct successors constantly above 80% which still leaves room for further improvements despite this result being already promising.

For bigger networks we see worse results. This is a result of the load which also increases with the network size, as in in our communication model each node is sending the same number of requests per time to the network. As here again links are saturated, it is mainly congestion that decreases the performance, and not the actual network size. We still believe that SONAR scales well with the number of nodes but this cannot be safely derived from our simulation study.

Concluding, SONAR shows a very good performance in different mobility scenarios and under various network loads. However, performance suffers from network congestion and as SONAR cannot adapt to this, it often even increases congestion by aggressively reacting on packet losses.

7.2 Future Work

In this first study, we identified several points in the initial design of SONAR that could be improved for better performance. This includes strategies to overcome the loss of a node's successor and the avoidance of source route headers. Furthermore we present some ideas that have not been implemented, yet look promising to lower protocol overhead and increase performance.

7.2.1 Redundant Successors

It turned out that actively maintaining virtual neighbors as in Chord is not viable for MANET environments. This consumes additional bandwidth and keeping all entries in the virtual neighbor table up-to-date is almost impossible with a low protocol overhead. Furthermore, using virtual neighbors showed no big improvement for routing performance or routing reliability, as we already exploit locality by exchanging information about nodes' physical neighbors.

As a modified approach of virtual neighbors, we can introduce redundant successors. Here, besides its successor each node maintains also "vice" successors, that are the next x nodes closest to itself on the ring. If the successor fails, these can act as replacements. This could be significantly faster than choosing the next best successor from the local routing cache and restarting ISPRP and may improve the ring consistency. Furthermore, SONAR could also benefit from this in networks with dynamic node membership. Thus far, we only investigated networks with static node memberships. If the recent successor disappears (either due to a link break or after the successor left the network at all) the next best vice successor will be chosen as a replacement.

This also speeds up the repair of broken paths, as the vice successor is virtually closer to the "real" successor than any replacement in a node's local routing cache.

The additional overhead of this approach is somewhat smaller than that of maintaining virtual neighbors. If the virtual addresses are m bits long, there are at most $2^m = n$ individual nodes in the network. Thus, each node has to maintain $(\log n)$ virtual neighbors.

In the other case we would only maintain two or three redundant successors which then could also be done at a higher frequency than it is possible for virtual neighbors. As maintaining a single virtual neighbor is as costly as maintaining the successor node, we can hardly update more than one virtual neighbor per successor update interval. In the case of 32bit long addresses, maintaining only three redundant successors instead of 31 virtual neighbors (as the virtual node number 0 is the successor) results in 10.3 times more frequent updates.

We assume 3 vice successors and 31 virtual neighbors respectively and an SPS update interval of 30 seconds as used in our simulations. According to Chord, together with the successor also one vice successor / one virtual neighbor is updated per interval. Thus, each vice successor is updated every 90 seconds while a virtual neighbor is updated every 930 seconds (15.5 minutes).

Apparently, the virtual neighbor paths are likely to be stale while the more frequent updates for vice successors result in faster link failure detection and repairs.

7.2.2 Source Routes

The simulations show that especially for networks below a size of 50 nodes the overhead caused by source route headers is significant and dominating. But also for big networks, the routing overhead is very high with up to 1.75 KB/s *per node*.

Therefore, there is a need to reduce the presence of source route headers: After a successor or virtual neighbor path has converged to a stable state which means that optimization will not improve the source route anymore, there is no need to repeatedly include the path. Each intermediate node stores the most recent path in a *soft state* manner and can route the message with this information. If a node wishes the send a message over a specific path it will store the source and destination of this path in the message and intermediate nodes will honor this by looking up the corresponding stored path and getting the next hop from it.

Assuming infinite cache size to store paths (so that no path replacement will appear due to memory space) and that every node uses the same path expiry times, this will work without a problem, as regular path update checks (by SPS messages as before) will ensure path freshness and will trigger repairs only if needed.

The current implementation also adds an optimized path to every message even when there is no optimized path available yet. For example, this occurs with 1-hop messages, or during the first few hops when the nodes will get no better path. However, this almost doubles the message size and could be easily avoided by including optimized path suggestions to a message on demand.

7.2.3 Routing and Limitations

Thus far, SONAR uses a greedy routing style. That is, shortest virtual paths are preferred. This works good in our simulation runs, however the network was rather dense. In more sparse networks this could lead to increased physical path lengths. It would be interesting to try different routing styles. SONAR could prefer shortest physical

paths with the dependency to still make some routing progress towards the destination in order to avoid routing loops. Whether this really results in shorter physical paths is an open question.

Additionally, we rely on bidirectional links. This is mainly assumed for sending NACK and ACK messages as they use the same path used by the SPS message. An extensions to ISPRP for finding a path back to the originally sender which does not rely on the path stored in the SPS message is needed to overcome this limitation.

7.2.4 Further Evaluation

We only tested SONAR in a limited number of scenarios. Further simulation runs in bigger networks and also in more sparse networks would be good to learn more about SONAR's behavior.

Furthermore, some of SONAR's parameters could be tuned in a better way than it was done here. One could think of a mechanism to adapt to different scenarios. Especially beneficial would be a mechanism to avoid or at least detect congestion. Thus, a node could limit its control traffic to a minimum. For example, by temporarily replacing the broadcast repair strategy with one that does not employ additional messages. We believe that especially broadcast traffic increases and forces congestion, thus skipping a HELLO message or occasionally choosing a bypass from the local routing cache could reduce congestion. However, as the repair quality then would be significantly lower, this would actually increase packet loss as a result of failed repairs due to stale bypass routes.

As described in Section 2.1, there is also a recursive design of ISPRP we chose not to use. Comparing the performance of the iterative and recursive implementation would be very interesting.

Another goal is it to replace the request generator by some more realistic P2P application. Voice services routed by SONAR will not work good as the message latency can grow to very high values if the mobility or network load is high. Other possible applications would be instant messaging or news services.

Bibliography

[1] Elizabeth M. Belding-Royer and Charles E. Perkins. Evolution and Future Directions of the Ad hoc On-Demand Distance Vector Routing Protocol. *Ad hoc Networks Journal*, 1(1):125–150, July 2003.

[2] Christian Bettstetter. Mobility Modeling in Wireless Networks: Categorization, Smooth Movement, and Border Effects. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(3):55–66, 2001.

[3] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, 1998.

[4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation*, December 2002.

[5] B. Chen and R. Morris. L+: Scalable Landmark Routing and Address Lookup for Multi-hop Wireless Networks. Technical report, MIT, 2002.

[6] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, June 2000.

[7] Curt Cramer and Thomas Fuhrmann. ISPRP: A Message-Efficient Protocol for Initializing Structured P2P Networks. In *Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 365–370, Phoenix, AZ, April 7–9 2005.

[8] Curt Cramer and Thomas Fuhrmann. Proximity Neighbor Selection for a DHT in Wireless Multi-Hop Networks. In *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing*, Konstanz, August 2005.

[9] S. R. Das, C. E. Perkins, and E. M. Royer. Performance Comparison of Two On-Demand Routing Protocols for Ad Hoc Networks. In *Proceedings of the IEEE Infocom 2000*, March 2000.

[10] E. W. Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerische Math.*, 1:269–271, 1959.

[11] S. Du, M. Khan, S. PalChaudhuri, A. Post, A. Saha, P. Druschel, D. B. Johnson, and R. Riedi. Safari: Self-Organizing Hierarchical Architecture for Scalable Ad Hoc Networking. Technical report, Rice, March 2004.

[12] Laura Marie Feeney and Martin Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In *INFOCOM*, pages 1548–1557, 2001.

[13] GloMoSim: Global Mobile Information Systems Simulation Library. `http://pcl.cs.ucla.edu/projects/glomosim/`.

[14] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of the ACM SIGCOMM '03 Conference*, pages 381–394, 2003.

[15] Y. Charlie Hu, Sumitra M. Das, and Himabindu Pucha. Exploiting the Synergy between Peer-to-Peer and Mobile Ad Hoc Networks. In *Proceedings of the 9th Workshop on Hot Topics in Operation Systems (HotOS IX)*, pages 37–42, 2003.

[16] Yih-Chun Hu and David B. Johnson. Caching Strategies in On-Demand Routing Protocols for Wireless Ad Hoc Networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 231–242, 2000.

[17] Yih-Chun Hu and David B. Johnson. Implicit Source Routes for On-Demand Ad Hoc Network Routing. In *Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing*, pages 1–10, 2001.

[18] David A. Maltz. *On-Demand Routing in Multi-hop Wireless Ad Hoc Networks*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 2001.

[19] Shree Murthy and J. J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mobile Networks and Applications*, 1(2):183–197, 1996.

[20] ns-2 network simulator. `http://www.isi.edu/nsnam/ns`.

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

[22] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244, 1994.

[23] Charles E. Perkins and Elizabeth M. Royer. Ad hoc On-Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, February 1999.

[24] H. Pucha, S. M. Das, and Y. C. Hu. Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks. In *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications*, December 2004.

[25] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM '01 conference*, pages 161–172, 2001.

[26] Sean C. Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, June 2004.

[27] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings IFIP/ACM Middleware 2001, Heidelberg, Germany*, November 2001.

[28] Elizabeth M. Royer, P. Michael Melliar-Smith, and Louise E. Moser. An Analysis of the Optimum Node Density for Ad hoc Mobile Networks. In *Proceedings of the IEEE International Conference on Communications*, June 2001.

[29] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, August 2001.