Institute of Parallel and Distributed Systems
Distributed Systems Department
Universität Stuttgart
Universitätsstraße 38
DE–70569 Stuttgart

Diploma Thesis Nr. 2950

Geographical distribution schemes to support Publish/Subscribe in Mobile Ad Hoc Networks

Stephan Schnitzer

Course of Study: Computer Science

Examiner: Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel

Supervisors: Dr. Boris Koldehofe
Dr. Hugo Miranda

Commenced: July 13, 2009

Completed: January 12, 2010

CR-Classification: C.2.2, C.2.6

Abstract

Mobile Ad Hoc Networks (MANETs) allow communication between moving nodes which does not depend on infrastructure like access points, stationary routers or GPS satellites. This offers new communication opportunities, especially in challenging environments. To communicate in MANETs we often need routing functionality, which usually provides unicast-based functionality. Publish/Subscribe (Pub/Sub) is a well known and powerful paradigm that provides higher expressiveness than unicast routing. It decouples senders from receivers and allows information exchange between network nodes that offer certain data (called publishers) and nodes that declare their interest in data of some pattern (called subscribers). Especially in MANET applications, Pub/Sub provides functionality useful to support realistic scenarios and novel applications. In most cases Pub/Sub applications are used in wired networks. The system models of wired networks are different to MANETs, hence solutions for wired networks are not feasible in mobile context. This diploma thesis investigates which concepts and distribution schemes are feasible to support Pub/Sub in MANETs in an efficient way. A new algorithm (*TPSR*) is proposed, tailored to efficiently manage data distribution to support Pub/Sub in MANETs. TPSR contains two innovative principles. First it uses the dissemination of subscriptions to create source routes. Second it makes use of the signal strength messages are received with, to optimize routes in terms of striking a good balance between long routes and fragile routes. The simulator Ns-2 is used to evaluate the different solutions and to discuss problems and solutions, which heavily depend on the scenarios used.

Acknowledgements

I thank the DAAD (Deutscher Akademischer Austausch Dienst) for funding two research journeys to Lisbon, Portugal (Project 0813002).

I also thank my supervisor Prof. Kurt Rothermel for his support and help on choosing the topic of this thesis.

Special thanks goes to my two advisors, Dr. Boris Koldehofe and Dr. Hugo Miranda for teaching me how research works. They guided me during the whole 6 months. Especially during the last week they supported me a lot to get my thesis finished in time, providing valuable comments and suggestions.

For providing infrastructure, computing power and related work I also thank the Distributed Systems Department of the University of Stuttgart.

Beside human help, I thank God my father for his endless love and support during the past six months.

Contents

List of Figures

List of Tables

Listings

Algorithms

1 Introduction

Mobile Ad Hoc Networks provide great flexibility. In many scenarios wired data connections are not feasible. Using wireless data transmission often involves some kind of infrastructure like access points, base stations or routers. Installing infrastructure is often expensive and may be subject to failure. Therefore Mobile Ad Hoc Networks are a good alternative for different purposes. However, the lack of infrastructure requires the mobile nodes to organize themselves, for example to support message forwarding. Message delivery is often unicast based, which means that a node wants to send a message to a dedicated other node (identified by its unique node id or address). To deliver the message from one node to another is the job of a *routing protocol*.

Beside unicast based routing there are more expressive approaches. One important paradigm is called Publish/Subscribe. A node (called subscriber) declares its interest in data of some kind to the Publish/Subscribe. Another node (called publisher) may produce certain data and pass it to the Publish/Subscribe system. Now it is left to the Publish/Subscribe system to distribute this data to all interested subscribers. This decoupling allows optimized routing and makes it easy to create applications based on top of the Publish/Subscribe system as no knowledge about network topology is needed and the application does not have to care about which node produces which information or is interested in it.

This chapter explains the characteristics of a Mobile Ad Hoc Network and a Publish/Subscribe system. We explain why we think that using Publish/Subscribe in Mobile Ad Hoc Networks contributes to support interesting application scenarios.

1.1 Mobile Ad Hoc Networks

A Mobile Ad Hoc Network (*MANET*) consists of a number of nodes that are considered to be small devices with a mobile power source (e.g. a battery). They can communicate by 2-way radio with other nodes that are within their receive distance, which is mainly determined by the transmission power, the used antennas and the physical environment. The network is called *mobile* because all nodes are allowed (and eventually will) do movements. In realistic scenarios movements usually show some kind of pattern. For example if we think of people moving in a big city, certain regions will probably contain more people than others, the shopping mall will probably attract more people than the sewage plant. Sidewalks, streets, buildings or lakes will usually influence the way people move. Likewise locations of forests, lakes and trails probably bias the movements of members of a rescue team. Which nodes are able to communicate with which other nodes, therefore changes continuously due to their movement. The lack of stable data connections makes it a challenging job to efficiently provide message routing functionality.

Mobile nodes (including but not limited to MANETs) have become more and more important during the last decades and the number of mobile nodes will probably be subject to sustained growth. An alternative to MANETs to support mobile nodes is to provide an infrastructure that does not move and aims to provide a working radio connection at each location between a node and at least one node of the infrastructure (called access point, mesh router or base station). However an infrastructure entails some drawbacks:

Cost Installing stationary nodes like base stations or routers and connecting them to provide handover functionality and routing between different base stations is very expensive, especially if coverage of large areas is required

Failure tolerance Base stations are single points of failure (or we would have to spend even more money to get a certain level of failure tolerance)

Energy Nodes typically need much sending power, to reach distant access points (since carriers try to avoid a high density of expensive access points)

Deployment Installing infrastructure needs much time to deploy which might not be available. Additionally finding reasonable places to install routers is often not easy. Although many people want to use wireless communication only few would agree having a base station located next to them, even administrative consent could be required. For MESH routers this is even more complicated, since they are usually installed at a smaller sized grid, which might be impractical.

Infrastructure based communication is a well known and established principle, as famous examples like GSM, WIFI or DECT show. On the other hand, in many scenarios, MANETs might be better, for instance in terms of cost, smaller devices, less radiation or better failure tolerance. MANETs can be easily established on demand in unknown areas that do not provide any infrastructure. Users can benefit from smaller devices that are able to operate everywhere and at almost no cost. In the next chapter we propose some scenarios related to MANETs.

MANETs are a good option for networks composed of a large number of small nodes, for example in a shopping mall, where customers and shops can exchange relevant information. Another example would be rescue scenarios, where emergency services can gather information from many small sensors, monitoring e.g. health, temperature or building integrity.
If we want to provide packet routing from and to arbitrary nodes in MANETs, there are many different routing protocols that try to provide this functionality; examples are deferred to Chapter 2.2. MANETs have been subject to mature interest and research in the recent years. They offer new opportunities of communication without the need of cables or any kind of infrastructure. However their characteristics are different compared to wired networks.

In a MANET it may happen that two nodes send data at almost the same time. A node receiving both messages simultaneously might not be able to receive any of the two messages. As all nodes transmit on the same frequency, their packets may interfere with each other. Hence a receiver might be unable to distinguish the two signals and will not receive a valid message. This is a common issue and addressed by different techniques, for instance by [61802, TMS03] for the well-known IEEE 802.11 standard. If a messages cannot be received by a node, because two or more messages interfere with each other, we call this a *collision*. As a collision possible means that a message has to be transmitted again, thus needing more battery power, it is often an optimization goal to minimize the number of collisions.

1.2 Publish/Subscribe

Publish/Subscribe (Pub/Sub) is an asynchronous communication paradigm of message delivery in computer networks, first published in [BJ87]. An *event notification system* (ENS) [RW97] provides basically the same functionality. Sometimes an ENS may also address enhanced functionality like subscribing to composite events or even the possibility to define complex high-level events based on several low-level events.

A Pub/Sub system is based on the idea of having data producers and data consumers, where the producers (called publishers) do not have to be aware about interested consumers (called subscribers). This approach implies that the communication from a source node to a destination node is usually a limited set of combinations, since not every node is a publisher and not every a subscriber. It also tends to be asymmetric, which means that data needs to go from a producer to one or more consumers, but not the other way round. To implement a Pub/Sub system, we are typically talking about a middleware that provides methods like Publish() and Subscribe() to the application layer as depicted in Figure 1.1. An application can call $Subscribe(s)$ to declare its interest in information of a certain kind; this node is then called a *Subscriber*. Accordingly, a *Publisher* is a node that offers information to Subscribers by calling its $Publish(p)$ method. The Pub/Sub system aims to deliver publications to subscribers with matching subscriptions which triggers the $Deliver(p)$ function. A subscriber which is no longer interested in a subscription he previously subscribed to, he can tell this to the Pub/Sub middleware by invoking $Unsubscribe(s)$. Other system models also introduce a method like *Advertise(a)*. An advertisement tells the middleware which kind of information the publisher can produce. If present a publisher is not allowed to call $Publish(p)$ unless he previously has invoked $Advertise(a)$ with a being an advertisement that matches p. Advertisements provide no functionality to the application, they even make it more laborious to use the Pub/Sub system. For that reason they are only introduced if the underlying middleware

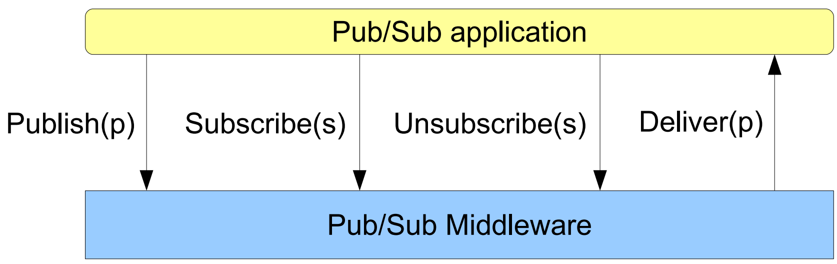

Figure 1.1: Publish/Subscribe system model

needs them. This is often used if the Pub/Sub system wants to create routing trees based on advertisements, which is not relevant for this thesis.

The two most import types of Pub/Sub systems are[1]:

1. Topic-based Pub/Sub, which uses a channel-like paradigm like IP Multicast. A limited set of channels is available and can be chosen by subscribers. Each Publication uses one of the channels, which causes the message to be delivered to all subscribers of this channel.
2. Content-based Pub/Sub, which uses an (theoretically) unlimited number of attributes and value ranges. A subscriber chooses a set of attributes and for each of them a value range. A publisher also uses a set of attributes and provides a specific value for each. A subscription matches a publication if and only if all of its attributes are used by the publication and additionally are within the given value range.

It is easy to see that content-based Pub/Sub is much more powerful than the topic-based one. We could easily emulate the second with the first one by using only 1 attribute and single-value ranges. This is the main drawback using the topic-based approach. Although its low expressiveness is sufficient for many scenarios, it is not suitable for other ones. Hence, in this thesis we only investigate content-based Pub/Sub.

In content-based Pub/Sub systems a publication is a mapping between arbitrary attributes and values. A subscription likewise is a mapping between arbitrary attributes and value ranges. This can be interpreted as arbitrary number of attribute/value pairs (or attribute/value-range pairs respectively):

$$\begin{aligned} \text{publication} &:= [\,\text{attribute; value}\,]* \\ \text{subscription} &:= [\,\text{attribute; value-range}\,]* \end{aligned}$$

Attributes can either be given by name or by a unique number. As we could define a bijective mapping between numbers and names, in this thesis we focus on numbers, which simplifies implementation and reduces the size of packets.

Some examples of possible MANET applications are [CCG+07, KSK06]:

Rescue team In case of emergency like earthquakes, conflagrations, storms or inundation existing infrastructure might have been partially or fully destroyed or is even absent. Rescue teams like police, fire fighters, medical service or volunteers can benefit from exchanging data. Nodes may provide for instance medical data of victims, building integrity or air quality. This allows rescue teams to subscribe to data relevant to them. Addionally also victims can subscribe to official announcements of some kind, for example the time when medical service arrives in a certain region, whether it is safe to leave their building or even access information about first aid before medical service arrives.

[1]Other types are for instance *type based*, *concept-based* and *XML* [GMR]

Shopping mall Shops can publish information related to the products they offer, their business hours and the location of the shop or the products. Customers are enabled to request information they are interested, for example they can subscribe to information related to trousers of their favorite brand and not exceeding a certain price. Or where to find something to eat at 1 am.

Airport Airlines, the airport operating company and shops could publish many information related to flights, products and security for example. Passengers may subscribe to information relevant for check-in, their journey or security. Without the need of any infrastructure this system could be easily deployed all over the world and provide useful information for airport passengers.

Sensors Wireless Sensor Networks contain nodes equipped with a sensor, such as for temperature, humidity, air pressure, speed or direction. They aim to monitor sensor data in regions that are hard to observe. In this scenario, sensor nodes will usually play the role of publishers and subscribers are humans or observation machines that want to be warned about critical conditions that require some counteraction for example. Additionally sensors might also act as subscriber, which makes it possible to produce aggregated publications (that are only issued if combinations of sensor events occur in some pattern).

Military One of the first applications for MANETs was to support military. This is especially useful since infrastructure might be destroyed or not be possible to use. In context of Pub/Sub many relevant events are created, for example to notify about injured people, observation data concerning relevant regions or commandments to arbitrary groups of participants.

1.3 Motivation

We argue that Publish/Subscribe has also big advantages in Mobile Ad Hoc Networks [ENP01]. Although MANETs and Publish/Subscribe have been subject of intensive research during the last years, there was only few research concerning Publish/Subscribe in context of Mobile Ad Hoc Networks. Existing solutions often restrict mobility in some way or maintain an overlay structure that produces lots of overhead if nodes move or subscriptions change. Design goals of Pub/Sub systems include speed, flexibility, decoupling (between subscribers and publishers) [EFGK03]. In MANETs we also have to address the delivery rate and the cost (i.e. the number of packets sent). Additionally the most challenging characteristic of MANETs is mobility. Nodes are allowed to do arbitrary movements. Although we can usually make some assumptions like estimating a maximum node speed, we cannot rely on any kind of stationary nodes (which is what some existing Pub/Sub systems do). Hence a good approach must cope with broken routes, lost packets, lack of position information and low energy consumption of mobile nodes. Providing all this at the same time is not easy and often a balancing act between opposed design goals.

In the previous chapter we mentioned several examples how Pub/Sub might be useful in context of MANETs. In the present days, Pub/Sub in MANETs is primarily used in context of research, which seems to be due to the fact that existing Pub/Sub systems often lack functionality or have been tailored to work in special scenarios only, like networks with few participating nodes, low node density or the requirement to install stationary routers.

We want to contribute to current research and improve this situation. This is our motivation to propose a new algorithm that supports Publish/Subscribe in MANETs.

The rest of this thesis is organized as follows. In the next chapter we provide an overview of existing approaches related to Pub/Sub in general and additionally focus on MANET context. We compare different algorithms and reason our decision the develop a new one. In Chapter 3 we provide our system model, design goals and possible design principles. Chapter 4 proposes *TPSR*, a new content routing algorithm to support Pub/Sub in MANETs. The evaluation of TPSR is explained in Chapter 5. The results are discussed in detail to reveal strengths and weaknesses of our algorithm. Reflecting the simulation results leads to Chapter 6 in which we propose future work to improve TPSR. The last chapter recapitulates the results and the contribution of our work.

2 Related Work

There are many scientific publications about Publish/Subscribe in wired networks and also many addressing different problems faced by MANETs. However there are only few publications dedicated to Publish/Subscribe in MANETs. This chapter provides a survey about existing literature and how it is related to the topic of this thesis. The first part provides a survey of broadcast techniques, because this is an important paradigm widely used for MANET applications.

2.1 Broadcast

Broadcast is a best effort mechanism to deliver a message to every participant of a network, in practice usually a *broadcast domain*. [NTCS99, WC02, Khe07, GMR] provide a good overview about different broadcast approaches and issues related to them. In this thesis we only focus on broadcast in Mobile Ad Hoc Networks. In MANET algorithms the number of transmitted messages is an important measure. Each transmission consumes battery power, which is expensive. Additionally while one node is transmitting, other nodes in transmission range have to delay their message to avoid collisions. For these reasons, the number of transmissions should be as low as possible. This chapter provides an overview of some existing broadcast algorithms and especially considers the number of transmissions they require.

2.1.1 Simple Flooding

Simple Flooding [WC02, JH01] is a straight-forward approach to perform a broadcast. The initiator or source node sends a message to each neighbor. Each node that receives a message also retransmits (rebroadcasts) it to all of its neighbors. However each node sends each message only once. The rebroadcast continues until each node that received the message also retransmitted it. Hopefully this affects a high percentage of nodes, which of course cannot be guaranteed since the network might be partitioned.
Using this algorithm we expect to send approximately as many packets as nodes are present in the network. To avoid collisions when multiple nodes retransmit immediately after having received a message, usually an improved version of flooding is used. Basically each node that receives a message applies a random delay before it rebroadcasts [NTCS99]. So different nodes receiving a message at the same time will probably choose a different delay and hence their messages will not collide.

The major disadvantage of flooding is that many retransmissions are completely unnecessary. Often nodes retransmit although all of its neighbors already received that message from other nodes. However flooding usually provides a good delivery rate. When reducing the number of nodes that retransmit, we might choose the wrong nodes, and hence some nodes might never receive the message due to that.

Disseminating data using flooding is a very expensive operation. When many or all nodes retransmit a message, a *broadcast storm* occurs, which means that due to the high network load, many collisions occur and other traffic has to compete with broadcast traffic [NTCS99]. However there are many cases, where flooding is a necessity. For example many MANET routing protocols make use of flooding to reach nodes at unknown locations (i.e. they issua a *route request*) and update its routing table. Flooding is often the last option, if better options did not succeed or are unavailable. In few cases it is even required to send certain information to all nodes. Therefore flooding is a key component and its efficiency important.

2.1.2 Probabilistic broadcast

To reduce the number of nodes that rebroadcast a message, a probabilistic approach was proposed [HHL02, NTCS99]. We define a probability p which means that each node decides to rebroadcast with probability p or to do nothing with probability $p - 1$. Hence, using $p = 1$ is the same approach as simple flooding. Choosing a good value for p is not trivial and is addressed by [SCS03]. According to [HHL06] a probability above 60 % is necessary to achieve a reasonable delivery rate.

2.1.3 Counter-based broadcast

An alternate approach to reduce the number of retransmissions is to use a retransmission-counter. Like the previous approaches, a node that receives a message applies a random timeout. But while waiting for the expiration it might hear other nodes retransmitting the same message. The node counts the number of retransmissions it receives. After the timeout expired, it will only rebroadcast the message, if the number of received retransmissions is lower than a certain threshold. Algorithm 2.1 shows the counter-based broadcast approach. As shown by [NTCS99] a high number of retransmits indicates that this node will not reach much additional area if it would decide to rebroadcast the message. This approach usually provides more benefit (in terms of less retransmissions) than the probabilistic approach. While the probabilistic version chooses the nodes that do not retransmit just by random, the counter-based approach makes nodes that provide too little benefit (in terms of additionally covered area) more likely to not retransmit. The key value to be evaluated and adapted to specific needs is the number of retransmissions. We encounter a trade-off between the percentage of nodes that actually receive the message and the number of retransmissions needed. Which nodes actually retransmit heavily depends on the randomly chosen delay. Since a node that chooses a high delay is more likely to receive enough retransmissions to prevent it from retransmission. Accordingly, a node choosing a small delay is likely to retransmit, just because during the short delay period not enough retransmissions were received.

Algorithm 2.1 Counter-based broadcast

```
procedure RECEIVEMESSAGE(msg)
    if msg is received for the first time then
        counter[msg] ← 0
        delay ← pick_random_delay_value()
        RESCHEDULEMESSAGE(msg, delay)      // after expiration, onTimeout(msg) is called
    else
        counter[msg] ← counter[msg] + 1
    end if
end procedure
procedure ONTIMEOUT(msg)
    if counter[msg] <= max_number_of_retransmissions then
        REBROADCAST(msg)
    end if
end procedure
```

2.1.4 PAMPA

The PAMPA algorithm [MLRR06, MLRR09, Mir07] is an enhanced version of the counter-based approach. Using the counter-based approach the nodes that retransmit heavily depends on random. We already saw that receiving lots of retransmissions often means that the additional area we reach is quite small. However, the number of retransmissions is not the only component indicating this. A second one is the distance to the previous node. If this distance is very long (i. e. close to the maximum receive distance), it is much more likely to reach a decent amount of additional area if such a node retransmits. Hence PAMPA does not choose the delay at random, but based it on the signal strength it receives the message with. Therefore lower signal strengths result in shorter delays which makes it more likely that this node will retransmit. Nodes being close to the node they received the message from, will probably never retransmit, since they choose a long delay and listen enough retransmissions. PAMPA enlarges the distances between hops that retransmit, while still trying to cover most of the region. This reduces the number of retransmissions for broadcast.

For that reason PAMPA is used as broadcast algorithm in this thesis. The only exceptions are the experiments with AODV and DSR which were not modified to use PAMPA, but use simple flooding. We assume that the signal strength value is provided by the network driver – functionality already present in many drivers. A more interesting question is how reliable this value is. For example due to hindrances like walls, buildings or even human beings, the signal strength can rapidly change, without requiring big changes at a node's location. As a preliminary answer we can argue that the worst case probably is (which has not been proofed, yet), if the signal strength value was completely random and unrelated to the nodes distance. In this case PAMPA would be as good as counter-based broadcast, which uses also a random value to determine the delay. If we assume that the signal strength is not perfect but also not completely random, we still expect PAMPA to perform better than the counter-based approach.

2.2 Routing in MANETs

Routing both unicast and multicast in MANETs has been subject to intensive research during the past decade and a number of different routing protocols have been published. We can distinguish between proactive and reactive approaches. Proactive means the protocol creates and maintains a routing overlay even if no user data is sent [NV09]. Therefore each node is required to periodically send advertisements, so that its neighbors are aware of its presence. If a node does not receive an advertisement from a neighbor for some time, it assumes that the link to this node is broken. In contrast, reactive routing protocols only find and maintain routes upon request. So when node A wants to send a packet to node B, it creates a route from A to B, but maybe not from A to C. Examples for proactive protocols are DSDV [PB94] and OLSR [CJ03]. Reactive protocols are for instance AODV [PR02], DSR [DBJB01, JM96] and DYMO [CBRP05]. Proactive protocols have some drawbacks. They produce traffic although no node wants to send messages, so the routes which are maintained with high effort, stay completely unused. Additionally, realistic scenarios often show asymmetric behavior, e.g. number of routes that are needed is often very low, compared to the total number of possible routes (which is proportional to the square of the number of nodes). Because of those drawbacks, we focus on reactive protocols, in particular on AODV and DSR.

DSR (Dynamic Source Routing, [DBJB01, JM96]) is a reactive routing protocol for MANETs. It uses two mechanisms, *route discovery* and *route maintenance*. If node A wants to send a message to node B, DSR will issue a route discovery. This basically means, that a message from A is flooded to all nodes of the network. Each node that receives a message attaches its own node id to the message before it is rebroadcast. Hence, a route request contains the issuer (A), the destination (B) and a path on which the node that receives such a route request is able to reach A. Node B eventually receives the route request. The route request provides him with a path to send a *route reply* to node A. This route reply contains the path that connects A with B. When this message is received by node A, it knows a path (i.e. a *source route*) to reach node B. To deliver a message, node A attaches the source route and sends the message to the next node as denoted in the source route. We call this a *source route* since the source (node A) knows the complete route to the destination and defines the path which is traversed to deliver the packet. As nodes can move, existing routes may break after some time because some node is no longer in transmission range of the remaining. A route which was once established is in the responsibility of the *route maintenance* procedure. Route maintenance tries to keep such a route working, for example by replacing nodes that have moved away by other ones. This is explained in detail in [JM96]

AODV (Ad-hoc On-demand Distance Vector, [PR02]) is a distance vector routing protocol with optimizations for MANETs. Like DSR it uses *route discovery* to find new routes and also uses similar techniques of *route maintenance*. The main difference is that AODV follows a hop-by-hop approach. So if node A sends a packet to B, AODV only addresses the next hop, but does not advice the next hop which path to follow. Each node receiving a payload message will decide, according to its routing table, to which node this message is passed next. Since an AODV node is not aware of the complete path a message will traverse, loops could occur. To avoid loops, AODV uses sequence numbers for each destination, so routes of different age can be distinguished. This also helps to replace stale routes in the routing table by new ones. For more details we refer to [PR02]. An interesting approach is published in [MM] and investigates whether AODV can be improved by using PAMPA for route discovery.

[DPR00] provides a performance comparison between AODV and DSR. They conclude that DSR is better for less challenging MANET scenarios (i.e. less nodes, traffic and movement). AODV performs better in networks with more participants and throughput and nodes moving faster. The main problem of DSR is the lack of removing stale routing table entries efficiently, so DSR spends lots of effort trying stale routes.

2.3 Pub/Sub without mobile context

2.3.1 Topic-based

Publish/Subscribe has become a major technology, which is more than 20 years old [BJ87]. Early Pub/Sub systems usually followed the topic-based subscription model. Many of them have emerged to industrial grade solutions [EFGK03] which are used for instance for data replication [Bir93]. Well known examples are iBus, SCRIBE [RKCD01] and the CORBA Notification Service, see [GMR] Chapter 10.3 for more examples. Since topic-based Pub/Sub systems have a lower expressiveness, their distribution strategies cannot be used for content-based Pub/Sub. However, we mentioned the topic-based approach because it can be called an historical foundation of content-based Pub/Sub.

2.3.2 Content-based

The list of available Pub/Sub systems for wired networks is relatively long. As those approaches are not suitable for MANETs, we provide only a short overview of two well-known systems. Some other examples are Elvin [SA97], Rebeca [FMG02], Gryphon [gry01] and LeSubscribe [FLP$^+$00].

Hermes

Hermes [PB02] was developed to provide a Pub/Sub system that is scalable (i.e. can be used for the internet) and generic enough to support custom applications build on top of it. Hermes establishes an overlay routing network, based on brokers which act as routers to forward Hermes related traffic. Publishers need to issue advertisements for the kind of data they offer before they are allowed to publish. Advertisements and subscriptions are flooded among the brokers, which sets up routing paths for each advertisement to the interested nodes. For each event type, Hermes maintains rendezvous nodes that manage advertisements and subscriptions are responsible to create appropriate routing paths. To avoid single points of failure, rendezvous nodes are replicated. To support content-based Pub/Sub, the principle of maintaining routing paths is extended by filters that allow flexible dissemination, avoiding unnecessary traffic.

Siena

SIENA [CRW01] is an event notification service which is especially optimized for large-scale applications. A network of event servers is responsible to receive and forward events. The network can be organized either as a peer-to-peer network or in an hierarchical manner. Clients may access this networks as providers (publishers) or subscribers. Publishers need to issue advertisements that tell the system which kind of data they are able to provide. Based on advertisements and subscriptions SIENA establishes a fine-grained tree within the network of event servers. Each server uses a set of filters that allows fast routing decisions for each publication, for example where to duplicate a message or in which branches the message must not be forwarded at all.

2.4 Pub/Sub with migration support

There are several approaches that provide some kind of migration functionality. Nodes are responsibly to tell when they want to move (check out) and at the new location they inform responsible nodes about their presence and subscriptions. Although literacy often calls this *mobility*, we prefer to call it *migration*. Examples of such systems are CEA [BMB+00], JEDI [CDNF01] or the mobility extension for SIENA [CIP02]. Some provide extended support for mobility, such as caching of messages while the node is moving and therefore temporarily unreachable. All those approaches share in common, that they are not intended to be used for MANET applications. Some are just extensions of the approaches mentioned in Chapter 2.3. If all nodes are moving broker based overlays for large-scale applications produce a non-negligible overhead which usually makes them inappropriate for MANETs. This also applies to extensions to support mobility.

2.5 Pub/Sub in special context

Several Pub/Sub systems apply only to specific environments and do not work for MANETs in general.

2.5.1 SPINE

SPINE [BKR08] requires a wireless MESH Network. MESH routers that do not move and are aligned in a grid to provide full radio coverage of the region are used to store subscriptions. The basic idea is to disseminate subscriptions in all routers located at the same horizontal row as the subscriber. Publications are disseminated to all routers located at the same vertical column as the publisher. Each router that receives a publication checks whether there are interested subscribers located in his row. If this is the case, the router disseminates the publication to the routers in its row, which is also received by the interested subscribers. Disseminating subscriptions only to one row reduces the number of sent packets. Simulations done by its authors did show that it provides good efficiency and is a reasonable approach to support Pub/Sub in Mobile MESH networks.

2.5.2 SSA

[DRWT96] proposes an algorithm that uses the signal strength to know which part of the MANET is stationary. Links between stationary nodes are expected to have low fluctuation at the signal strength. To know which nodes are stationary contributes to find good routes, since stationary routes can be used as routers that provide stable paths for long periods. Hence this algorithm assumes that not all nodes are moving or at least that nodes that currently do not move are very likely to keep their current position for some time. Whether this algorithm is suitable for certain situation therefore mainly depends on the assumption that a subset of nodes does not move and can act as routers (similar to routers in wireless MESH networks). This assumption however is the reason why this algorithm is not suitable for MANET applications in general. The simulation used to evaluate it, assumes that 20 % of the nodes are stationary. This scenario is more challenging than a MESH network, as the knowledge which nodes are stationary has to be created by the algorithm (in MESH networks each node is perfectly aware of the fact whether it is a router or not). Additionally nodes are usually not aligned in a grid nor cover the complete area.

2.5.3 STEAM

STEAM [MC02] provides techniques, similar to Pub/Sub. The communication model they follow is based on proximity group communication. This basically means that subscribers and publishers are required to stay in the same region. This approach addresses mainly scalability. MANET applications, especially routing protocols or higher level Pub/Sub protocols, usually do not scale well if the number of participating nodes and the network size increases. These scalability issues can be avoided, if routing paths do not exceed a certain length. Unfortunately this approach is only suitable for scenarios that accept those constraints, but not for generic MANET applications, probably even not for the use cases we propose in the next chapter.

2.6 Pub/Sub in MANETs

2.6.1 topic-based Pub/Sub

There are several publications that address topic-based Pub/Sub for MANETs. For example MOBCROSS [DSM09] uses the MAODV multicast routing protocol [KSK99] to create and maintain a broker-based overlay, connected by a multicast tree. All publishers are required to send advertisements before they publish something. An advertisement basically tells other nodes which kind of information (which topic) might be published by this node. Based on these advertisements, a broker can decide to send a subscription to the publisher. Then this broker will receive publications sent by the publisher. The broker is then responsible to deliver those publications to interested subscribers. Subscribers register their subscriptions at the responsible broker that will ensure that relevant messages are forwarded to them. As MOBCROSS only addresses topic-based Pub/Sub, we do not step into more detail.

2.6.2 content-based Pub/Sub

COMAN [LMP], a content-based Pub/Sub system for MANETs is implemented at the application layer and built on top of a modified version of MAODV. All nodes run a broker application and are part of an MAODV multicast tree, that enables each node to send messages to any other node. Subscriptions are sent to the brokers. Hence they know which publication needs to be forwarded to which branches of the tree. Topology changes are addresses by various tree repair mechanisms, many of them used by MAODV in the same or a similar way. Making each node part of a multicast tree is a very interesting approach. This is expected to provide good results in MANETs consisting of a limited number of nodes (their simulations were using up to 100 nodes) and a very sparse network. We expect COMAN to get problems due to high collision rates if the node density or the communication range is increased.
We think that COMAN is an innovative approach to support Pub/Sub in MANETs, however we think that further evaluations have to show how it performs in different scenarios. As we expect MAODV to produce a high overhead maintaining its routes [AMMMQ07], we do not expect COMAN to be a generic solution.

The algorithm proposed by [YB05] also uses a multicast ad hoc routing, in particular ODMRP. Multicast routing protocols are intended to be used according to topic-based systems. Using content-based Pub/Sub and having a set of subscribers S, means that there are $|\mathcal{P}(S)| - 1$ subsets of subscribers (we can ignore the case of 0 interested subscribers). As each subset can possibly be the set of interested subscribers, the publisher has to address, we would in a naive approach eventually create up to $|\mathcal{P}(S)| - 1 = 2^{|S|} - 1$ different multicast groups. As this does not scale, the mentioned algorithm uses Bloom filters to compress subscription data, empowering the algorithm to use a smaller set of multicast groups to deliver publications. This approach seams to have the same problems as COMAN and the results published at [YB04] seem to confirm that.

2.7 Overview

We found only limited research addressing content-based Pub/Sub in MANETs. Table 2.1 provides an overview of several approaches. It shows wether they work for wired networks (WN), support Migration (Migr.) or are suitable for MANETs.

Pub/Sub system	Model	WN	Migr.	MANET	Comments
SCRIBE [RKCD01]	Topic	✓			
CAN [RHKS01]	Topic	✓			
MOBCROSS [DSM09]	Topic			✓	
SIENA [CRW01]	Content	✓			
Elvin [SA97]	Content	✓			
Rebeca [FMG02]	Content	✓			
Gryphon [gry01]	Content	✓			
LeSubscribe [FLP+00]	Content	✓			
CEA [BMB+00]	Content		✓		
JEDI [CDNF01]	Content		✓		
Ext. SIENA [CIP02]	Content		✓		
SPINE [BKR08]	Content			(✓)	MESH network
SSA [DRWT96]	Content			(✓)	Stationary nodes required
STEAM [MC02]	Content			(✓)	Proximity required
COMAN [LMP]	Content			✓	Small, sparse networks?
[YB05]	Content			✓	Small, sparse networks?
TPSR (Chapter 4)	Content			✓	

Table 2.1: Overview of existing Pub/Sub middleware

Most Pub/Sub systems only work for wired networks or provide only limited support of mobility. The two solutions mentioned in the last chapter seem to address scenarios consisting of a small number of nodes distributed on relatively large areas. All other solutions we found work only if the mobile network obeys certain restrictions. For instance some nodes are not allowed to move at all. This confirms our argumentation that only few research aims to support Pub/Sub in MANETs and is supports our motivation to develop a more generic approach that provides useful contribution to current research. The next chapter discusses possible design principles to support Pub/Sub in MANETs.

3 Basic design principles for Pub/Sub in MANETs

There are many challenges in the design of a Pub/Sub content-based protocol for MANETs. Simply flooding all publications in the network seems to be a simple but probably not always efficient way. However, we choose this as reference algorithm to compare results since it showed the best simulation results according to preliminary tests explained in the last section of this chapter. We also take into account that the certain interest of subscribers depends on their locations, therefore subscribers that are located closer to each other are more likely to be interested in the same kind of information (publications), we call this assumption *GEO* in this chapter.
In this chapter we discuss the multiple trade-offs involved in the design of a Pub/Sub content-based protocol for MANETs and discuss some possible approaches.

The next chapter explains the system model and requirements that a Pub/Sub system has to address. Section 3.2 provides an overview of concepts which might be used to implement a Pub/Sub system. Additionally we discuss their feasibility and reason our decision to develop a new approach.

3.1 Requirements for a Pub/Sub system in a MANET

To support Pub/Sub in MANETs several optimizations and characteristics must be considered. Most important are (partially adapted from [DSM09], Chapter 3):

Mobility As we expect frequent topology changes due to node movements, the algorithm has to adapt to new situations without creating high overhead.

Scalability The algorithm should scale well of scalability factors. Examples are the number of participating nodes, the size of the network, the number of publishers, the number of subscribers, the publishing rate, the interval at which subscriptions change the node speed.

Asynchronous communication Due to node movement and the Pub/Sub paradigm, synchronous communication is not feasible in MANETs.

Energy consumption As battery is a valuable source on mobile nodes, the number of packets sent should be kept as small as possible.

Hardware constraints The memory and cpu requirements to process the algorithm should also be low.

Heterogeneity Realistic scenarios for MANETs often lack predictability. The number of participants may range from a few up to thousands or even millions of nodes. Additionally node distribution is often erratic. Certain regions have extremely high node densities, while others might contain no or few nodes.

Effectivity Provide a high delivery rate, i.e. aim to deliver each publication to all interested subscribers.

To address all of those design goals is challenging. A Pub/Sub system has to cope with different situations, produce few traffic and still provide reasonable functionality. In a MANET we cannot guarantee that we are able to reach a certain node. A node might be located in a region where no other node is able to communicate with it. Hence other nodes cannot send any traffic to it. For a unicast routing protocol it might be a feasible solution to tell the sender if the message was probably not delivered. Since a Pub/Sub system provides an abstraction layer that hides lower layer details from the application layer, a Pub/Sub system can be called a Middleware. Passing a notification to a publisher telling him, that its publication could not be delivered to some subscriber is hence considered useless. This would reduce the decoupling of publishers and subscribers and additionally the publisher is not supposed to benefit from negative acknowledgements. We even cannot provide this information reliably, since an interested subscriber might be unable to tell other nodes about his subscription because he is too distant from them. In contrast, a unicast routing protocol could provide a negative acknowledgement, as the sender explicitly tells which recipient this message is addressed to. In this case the routing protocol just assumes that the node exists, but is not reachable at the moment.

3.2 Some possible approaches

This section shows and discusses a set of different algorithms that might be possible solutions for Pub/Sub in MANETs.

3.2.1 Broadcasting publications

This simple approach, called *P-PUBS*, does not exchange subscription data between nodes. All publications are broadcast to all nodes, no matter to which node they are relevant. Each node that receives a publication locally knows whether he is a subscriber to that message and needs to pass it to the application layer. To make flooding efficient we could use Pampa (see Chapter 2.1.4).
This approach is expected to scale almost linearly with the number of publishers and the publication rate. However when traffic increases, more collisions occur, which may reduce the delivery rate and may also cause more packets to be sent (because retransmissions of other nodes are not received due to collisions). The node speed is expected to have almost no influence [Mir07]. Low node density however may reduce the delivery rate and increase the effort needed to successfully disseminate publications.

P-PUBS is located at the network layer, which is good because it cannot be affected by limitations of an underlying routing protocol.

3.2.2 Use unicast routing protocols

An alternative would be to propagate only subscriptions by flooding, so that each node knows which other nodes have subscribed to a certain publication. When a node wants to publish, it can use some routing protocol to deliver his publication only to interested nodes. In this approach routing is done by unicast routing protocols for MANETs like AODV [PR02] and DSR [DBJB01, JM96]. This approach generates (in contrast to P-PUBS) traffic, even if there are no publications sent at all, just by flooding subscriptions. On the other hand if there are only a few subscribers, the effort of sending a message only to few or even none recipients should be much lower than flooding the whole network. Hence we would expect to find some threshold number of subscribers that estimates in which cases P-PUBS and in which the approach based on unicast routing is more efficient. As the Pub/Sub system would be based on existing protocols, we expect drawbacks when trying to optimize the system, since some optimizations would require us to apply changes to AODV or DSR.
We present 3 different versions of this approach and assess them.

Non-optimized unicast routing

As described above without further optimizations. This approach scales almost linearly with the number of publishers and with the number of subscribers, if we ignore packet loss which occur more often if traffic increases. Due to packet losses or unlucky node positions, subscriptions might not get delivered, impacting delivery rate. All routing protocols share the principle that routes can be reused several times. If routes break very often, this impacts the number of sent packets, because this usually causes the routing protocol to flood the network with a route request in order to create a new route.

Unicast-routing with subscription polling

To get aware of subscriptions, a publisher also polls for updates, telling his neighbors the subscriptions he already knows. If a node receives such a polling message, and the set of subscriptions differs from those he already knows, he sends an incremental update message with the missing subscriptions. If it was the receiving node that wasn't aware of the latest subscriptions, it can also update its local cache. This modification is expected to increase the delivery rate, because the subscription caches of publishers are more up to date. Although we increase the number of sent messages, although this fact might be negligible if we consider the benefit. Another variant of this optimization is, to piggyback the set of subscriptions a publisher knows on a publication delivery message to a subscriber. Hence all nodes forwarding this message may update their own subscription cache if the publisher's cache was more up to date. Additionally they can determine if the publisher used an outdated set of subscriptions and send a notification to him. This increases the packet size but should anyway be better than sending this information in a different packet. Of course this works only if the publisher knows at least 1 interested subscriber, otherwise polling can still be used. Because only active publishers query subscription updates and usually at least 1 subscriber exists, the number of additional packets sent (using this modification) is expected to be very low.

Aggregated 1-hop-routing

If many interested subscribers exist, AODV and DSR are expected to not perform well, because we use routing based on unicast on a scenario which can be interpreted as a tree. For instance if we have to deliver a publication to 10 subscribers, the publishing node would send the same publication 10 times, each one with a different destination address. This might be unnecessary and produce additional traffic. Unfortunately AODV or DSR do not support sending to a set of recipients, which is what would fit our needs. However it is possible to circumvent this issue by sending messages only once to each node that is the next hop according to the routing table. Received there, this node inspects the packet to which recipients it must forward the publication to and creates a new packet and do the same again. This is not optimal, because a message is sent once for each node that is next in the tree, but it is much better than sending one message for each subscriber. To determine the next node we can simply access the existing routing table of AODV or DSR and also calculate an optimized delivery tree to reduce the number of hops as much as possible. If the routing table does not contain a route to a subscriber we still can tell to the routing protocol to route to that node. This will indirectly issue a route request which (hopefully) updates the routing table so that for other events we can do it the optimized way. If *GEO* applies, we can gain even more benefit compared to the previous algorithms, because it allows us to split routes later. DSR offers more opportunities to create an optimized routing tree, because it contains full source routes and also keeps several of them, so we can choose the best combination. However, the routing table of DSR often tends to contain stale routes [DPR00].

Poison routing table

One point that fazes at the previous solutions is the fact that flooding subscriptions already finds routes from the subscriber to all possible destinations. But because Pampa is used, the routing protocols cannot benefit from this information. Therefore we can extend the previous algorithm with the functionality of injecting source routes (collected by Pampa) into the routing tables of AODV or DSR. Doing this we can probably avoid many broadcasts caused by route requests. This advantage does not only exist for the first time a node issued a subscription, but also during normal operation, since subscriptions time out after some time, or a subscriber changes his subscription, which results in a new flood to disseminate the subscription in the network and hence updating the routing tables again.

PCache

Instead of Pampa we could also use PCache [MLRR09]. To use it for Pub/Sub was proposed in [MBGK+08], to disseminate subscriptions. PCache aims to store data only in a small percentage of geographically distributed nodes, maintaining a distance of a few hops between them. If a node needs to access data it can query his neighbors and the relevant message should be given to him by one of the nodes that store this data and is usually only one or two hops away. This saves memory on the nodes and also allows nodes to be temporarily unreachable or overhear packets during the dissemination phase and still be able to access the information when needed. However a publisher is required to ask his neigbors for relevant subscriptions by issuing a modified "query-all" message, which generates additional overhead but only provides small benefit. If the only advantage is that we are aware of more subscription we could do this probably with less effort, just by polling once periodically. That we need less memory is an advantage. In MANETs however we assume that memory is very cheap and needs very little power, whereas sending more messages is a bigger concern.

3.2.3 Multicast routing protocols

It might also be a question, whether multicast routing protocols can also be a solution. Multicast routing would offer a publisher to send a single publication to a set of recipients, while the routing protocol maintains a routing tree to get this done in an efficient way. An example is the MAODV protocol (see [KSK99]). However multicast routing provides a different paradigm than the traffic pattern we need. Multicasting allows a recipient to subscribe to a channel and senders to send messages for a certain channel, which is basically the concept of topic-based Pub/Sub. In content-based Pub/Sub, we cannot assume that a subscriber only chooses from a limited number of subscriptions. If every unique subscription is considered to represent a channel, we would almost simulate unicast routing with multicast routing, which does not make sense. Creating a dedicated multicast group for each publication does not make sense, either, as the overhead to create the tree is quite big.

An interesting approach using a single multicast tree to reach all nodes, was proposed in [LMP], we discussed this in Chapter 2.6. An alternative is to require the publisher to issue an advertisment before starting to publish. We could then create a multicast group for each advertisement, where all subscribers interest in publications that could match that advertisement will join. This solution has also some drawbacks. If an advertisement is too generic, many publications are delivered to subscribers that are actually not interested. Using stricter advertisements would (according to our system model) perhaps require a publisher to use several advertisements to adequately describe which data he can produce. Let us assume that a publisher is able to publishing data with attribute a_1 being either 0, 5 or 10. He has now two options:

- Issue a single advertisement with $a_1 \in [0, 10]$. This would also connect a subscriber who is only interested in $a_1 = 3$.

- Issue 3 different advertisements for each of the 3 possible values of a_1, thus reaching only interested subscribers, but also creating 3 mulitcast trees, which is lots of overhead.

The performance of such a solution heavily depends on the subscription and publication pattern.

A different approach was proposed in [JF08]. They use a self-implemented content routing protocol which uses Bloom-filters [Blo70] to decide in which branches of the routing tree a publication is forwarded. This mainly improves routing speed by accepting more traffic, since routing decisions might be wrong. This approach is interesting, but is intended to be used in wired networks, which is also supported by the fact that this approach is based on SIENA. If routes fail, the network of event routers would have to be reconfigured, which is expensive.

For those reasons we argue that multicast routing is not a feasible approach to implement a Pub/Sub system for Mobile Ad Hoc Networks.

3.2.4 Using a new content routing protocol

The proposed unicast-based algorithms share in common that a certain amount of traffic is unnecessary. At least sending a message multiple times, just to address multiple direct neighbors, is unneeded and only done to circumvent the limitations of unicast-routing as much as possible. However developing a new algorithm on the network layer enables us to build a system specifically tailored for Pub/Sub in MANETs. In Chapter 3.3 we explain why the approaches mentioned above are not suitable to support Pub/Sub for MANETs efficiently. As a consequence, in Chapter 4 this thesis proposes a new routing protocol.

3.3 Using unicast routing

3.3.1 Unicast-based algorithms

When we were developing the conceptual part of this thesis, the most obvious solution was to use existing routing protocols to support Pub/Sub in MANETs. In particular we investigated AODV and DSR which are well-known representatives of distance-vector and source-route routing protocols for MANETs. As a first step we wanted to send all publications with unicast messages using either of the two routing protocols, as described in Chapter 3.2.2. We call these protocols *P-UCR-AODV* and *P-UCR-DSR*. Thereupon we wanted to optimize routing later for example using aggregated 1-hop-routing.

3.3.2 Preliminary tests

To get a simple comparison and investigate the performance of AODV and DSR we created a simple set of test scenarios, which is described subsequently. For details about the evaluations we refer to Chapter 5, unless stated otherwise the setup explained there, also applies to the evaluation here. Results in this chapter reason our decisions to develop a new approach.

Attribute	**Value**	**Comments**
Number of nodes	200	
Node density	50 - 200	nodes/km^2, steps of 30
Max. node speed	3 m/s	
Publishing rate	0.5/s	rate per publisher
Number of publishers	10	
Number of subscribers	4	

Table 3.1: Parameters of preliminary evaluation

The simulation parameters are depicted in Table 3.1. Each subscriber has a unique, statically assigned subscription. Hence, we did not consider the effort to propagate subscriptions, yet. Additionally each publication always matches exactly one subscription. For further details we refer to Chapter 5. These simulations can be called unrealistic for Publish/Subscribe. In particular we do not even evaluate P-UCR since we do not use PAMPA to disseminate subscriptions. However, as a first step we only want to measure routing performance. In this scenario each of our 10 publishers chooses in average every 2 seconds one out of 4 subscribers he wants to send a message to. Hence in average each subscriber receives $1/8 \times 10$, which is 1.25, messages per second in total. Having 4 subscribers, 5 messages per second have to be delivered. We think this traffic pattern is low and should be handled by an 11 MBit/s MANET without any problems, especially because the message sizes are very small (see Appendix 5.4). We chose this pattern, because each publisher always sends his message to exactly one subscriber. If we had more than 1 recipient for a message, we could think about optimizations that try to aggregate routes in a way that saves some packets – having only 1 recipient this is not possible. In a useful Pub/Sub algorithm we would have to distribute subscriptions to all publishers (e.g. using PAMPA), this would obviously increase the traffic. This means that a useful algorithm involving AODV or DSR would be worse than their evaluation results shown in the next chapter.

3.3.3 Results

We compare the efficiency of AODV and DSR unicast routing with that of P-PUBS (introduced in Chapter 3.2.1). Besides AODV and DSR we simulated also P-PUBS(1) and P-PUBS(2). The number in braces at P-PUBS represents to number of retransmits used for PAMPA. So P-PUBS(x) uses PAMPA(x), we explained this already in Chapter 2.1.4. Intuitively we would expect unicast routing to be more efficient than broadcast. Since a more realistic scenario

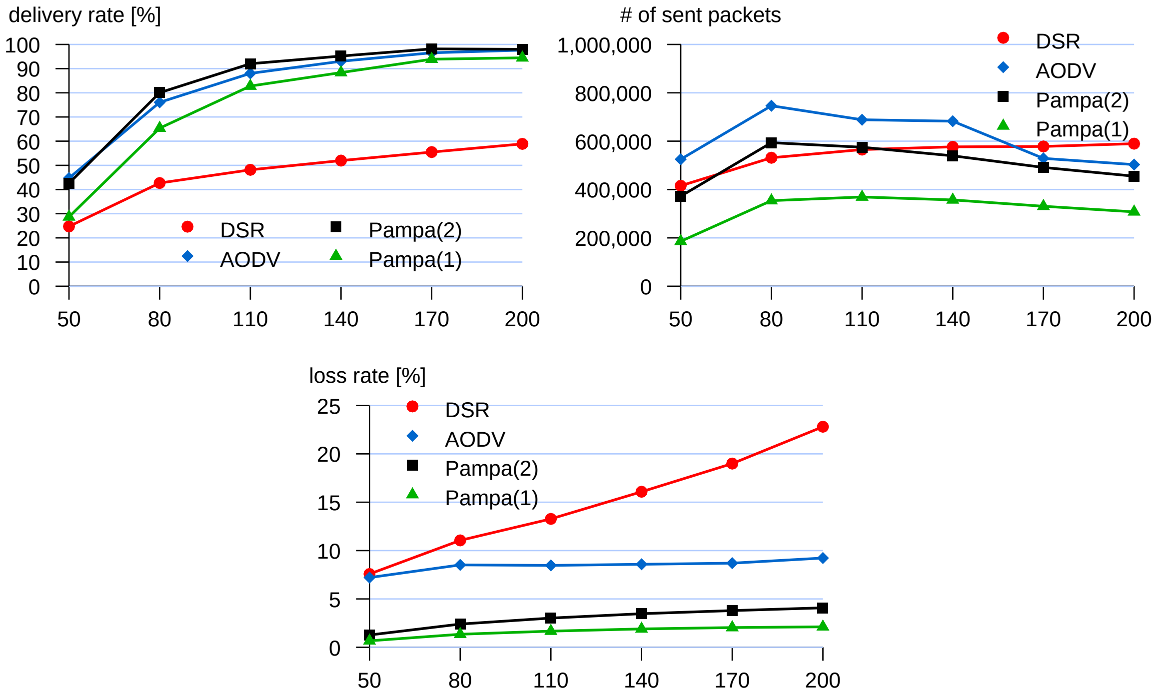

Figure 3.1: Comparison AODV/DSR/PAMPA with varied node density

would include several interested subscribers receiving a certain publication, P-UCR-AODV and P-UCR-DSR only make sense if they are much better than P-PUBS. Otherwise they get outperformed soon if subscribers issue similar subscriptions. Because then they would provide a reasonable solution only for very special scenarios. This is based on the reasonable assumption that sending a packet to one recipient is at least as much effort as to send it to more than one (although we cannot prove this). Figure 3.1 shows the results of our simulation. It is a very surprising result that flooding a message to all nodes is more efficient than sending it only to a single recipient using unicast routing. AODV achieves a similar delivery rate as PAMPA(2), but also sends much more packets. DSR sends fewer packets, at a density of 80 it is even clearly better. However, if we take into account that PAMPA(2) delivers about twice the numbers of packets DSR does, we always would prefer PAMPA(2). Even if such a low delivery rate would be sufficient we still could modify P-PUBS, so that the publisher just drops every second packet for example, which would reduce traffic by at least 50 % and delivery rate by not more than 50 % ("at least" and "more" since the collision rate drops, too). We send about 6000 publications in total, as PAMPA(2) needs in average about 500000 packets the effort of PAMPA(2) to disseminate a single subscription is about 83 messages. Looking at the cost per delivered packet, PAMPA(1) is the clear winner. Followed by PAMPA(2), AODV and DSR. Both unicast protocols suffer from high loss rates caused by the traffic they created. This results in frequent route errors. AODV and DSR are good if routes they found once do not change too soon. Otherwise we get broadcast storms that cause a high loss rate and also participate in making things worse. A more detailed comparison of AODV and DSR can be

found in [DPR00].
We want to provide an algorithm, which is able to connect multiple publishers with multiple subscribers, exchanging data in parallel and at decent rates. Having 10 publishers and each sending publications at a rate of 0.5 publications per second seem to be reasonable assumptions. As a result from this simulation we can say that already at these comparably low requirements on efficiency and delivery rate, AODV and DSR are not suitable to support Pub/Sub in MANETs.
Using PAMPA has an obvious advantage, which is that it is almost agnostic to the node speed. If nodes move very fast, routing protocols that take much effort to find stable routes, get not much benefit from them. The routes they create cannot be used often because node movement breaks them early. The conclusion that P-PUBS is much more efficient than existing unicast routing protocols, is our motivation to develop a new content routing algorithm whose design goal is to support Publish/Subscribe in Mobile Ad Hoc Networks in an efficient way, instead of building upon algorithms designed for different purposes.

4 Tree-based Power-aware Source Routing

In this chapter we introduce a new event routing protocol to support Pub/Sub in MANETs. Its short name is *TPSR* which stands for Tree-based Power-aware Source Routing protocol. The algorithm basically consists of two parts. The first one disseminates subscriptions to all nodes and the scecond delivers publications to relevant subscribers.

Subscriptions are pushed to every node, which includes in particular, the publishers of relevant publications. An early push of subscriptions contributes to an improved performance of the Pub/Sub system because:

1. For some traffic patterns, in particular when the ratio of publications per subscription is high, it reduces the total amount of traffic.
2. Reduces the latency for delivering publications, given that the list of subscribers is available to every node and to every publisher in particular.

TPSR requires an increased amount of memory, in comparison with more conservative approaches, possibly storing the subscriptions in a limited number of locations but exchanging an additional number of messages per publication. However, we have recently witnessed a considerable increase in the amount of memory made available to mobile devices which contrast with the not so significant gains achieved by batteries. These in turn are severely affected by the number of retransmissions performed by each node [Fee01].

Subscriptions are pushed using a variation of the PAMPA [MLRR06] broadcast protocol. PAMPA requires each node to retransmit at most once each message what ensures broadcast termination. Elsewhere [MLRR06, Mir07], it was also shown that PAMPA requires a number of retransmissions considerably lower in comparison with other broadcast algorithms.

In TPSR, each node that receives a subscription appends his own node id to the message before retransmitting, creating a list, or path, of hops traversed by the subscription. This path, which is actually a source route pointing to the subscriber, and the subscription are stored by every recipient. Therefore, when receiving a subscription message, each node becomes aware of the subscription and also of a route to reach its subscriber.

The broadcast nature of the wireless medium makes very likely that each node receives multiple copies of the same message, each retransmitted by a different node in proximity. Therefore, each node could learn different routes to the subscriber. However, these routes would perform differently, with some being longer or more likely to break due to node movement than others. Routes in MANETs are particularly vulnerable to node movement and therefore, it is important to choose routes with the lowest probability of becoming unusable. TPSR complements the original PAMPA algorithm by estimating the quality of each route.

PAMPA exclusively selects retransmitters from their distance to the sender. TPSR bias this basic algorithm to consider smaller routes and to select nodes that are neither too close nor too distant from the sender. The goal is to have TPSR creating source routes that are assumed to provide a working route between subscribers and publishers for a reasonable amount of time (before nodes moved too much and the route is destroyed).

The routes discovered by TPSR are periodically refreshed. Subscription messages contain a timeout value. When a subscription times out, all nodes purge it from their lists. To avoid, it, the subscriber must refresh the subscription following the algorithm described above, which will result in a regeneration of the routes.

Thanks to the subscription dissemination, a publisher is aware of all subscriptions and can easily check which subscriptions match each particular publication. We call to the subscribers whose subscriptions match this publication the list of recipients. For each recipient the publisher uses the source route stored when the subscription was received to determine the next hop that should be followed by the publication so that it is delivered to that recipient. Note that routes to several recipients may share the same next hop. Publisher creates an array of "route job" structures, indexed by the next hop. We call each "next hop" an addressee and each route job contains the list of recipients for the corresponding addressee.

A single "publication message" is broadcast by the publisher. In addition to the publication itself, the "publication message" contains the route job list. Because it was broadcast, the "publication message" is expected to be delivered to all neighbors of the publisher, which should include all addressees. The addresses receiving the message deliver it to the application if they are interested subscribers. In addition, they repeat the algorithm followed by the publisher. That is, they i) prepare a new array of route jobs, indexed by the next hop. Next hops are determined using their local knowledge on the routes to reach the subscribers that have been assigned to them; ii) prepare a new message containing both the publication and the route jobs array; and iii) broadcast this message.

The publication delivery algorithm of TPSR is depicted in Fig. 4.1. In this figure, the publisher (P) prepared a publication message containing the publication and two route jobs, whose addressees are nodes A1 and A2 and containing respectively subscribers S1 and {S2,S3}. Addressee A1 would then forward a publication message with a single route job for addressee S1. A2 in turn would create a new publication message with two route jobs, one for S2 and another for S3. The tree-based structures created during the delivery of publications that justify the name of the protocol are visible in the flow of data depicted in the figure.

Publishers and addresses monitor the network to ensure the continuation of the propagation of the publications. Due to node movement, some of the addresses may no longer be in transmission range, thus creating broken links in the propagation tree. A node that detects a broken link broadcasts the publication using Pampa. In addition to increasing the probability of successful delivery of the publication, this broadcast will equally serve to repair the source routes to the subscribers because in reply to receiving the publication, the subscriber will reissue a subscription.

This chapter details the operations performed by TPSR and discusses the options that have been taken in its design. The chapter is organized as follows. Section 4.1 describes the creation

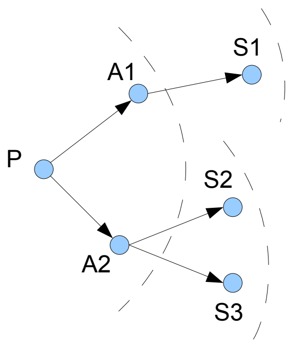

Figure 4.1: Tree-based publication delivery by TPSR

of source routes, addressing all the enhancements that have been applied to Pampa. The delivery of publications is the focus of Sec. 4.2 while the route repairing mechanisms are addressed in Sec. 4.3.

4.1 Creating source routes

When a subscriber invokes the Subscribe() method, a message containing his unique node id and the subscription, is broadcasted to all nodes. To do this in an efficient way we use Pampa [pam, Mir07]. Before a node retransmits such a message, it appends its own unique node id to the message. Hence, each node that receives a message also receives the path of nodes that sent or forwarded the message. It seems to be a promising approach, to use this information to deliver publications, since we can traverse the path in reverse to reach each subscriber. Figure 4.2 depicts the packet format of a subscription message. Given that the number of entries of the source route is $\#hops$ and $\#attr$ being the number of subscription attributes. The path we get is called *source route*, because this route is determined by the source node,

0	1	2	3	4	5	6	
subscription id		node id of subscriber		timeout value		$\#attr$	
attr. id Nr 1	from-val. Nr 1	to-val. Nr 1	...	attr. id Nr $\#attr$	from-val. Nr $\#attr$	to-val. Nr $\#attr$	} subscription
$\#hops$	node id Nr 1	dist. Nr 1	...	node id Nr $\#hops$	dist Nr $\#hops$		} source route

Figure 4.2: Packet format for subscription dissemination

which is the publisher. However, using Pampa to create source routes has two drawbacks. First we have to add the source route data to each packet, which marginally increases the message size and also means that we do not have a fixed-length header. Second the number of hops from subscriber to publisher is to be increased, which means we have to perform more retransmissions. Because it would not be reasonable to maximize hop distances, as Pampa does by default. This would produce very fragile and therefore almost unusable routes. If our route contains links that are close to the receive distance, it is very likely that these links break soon, due to small node movements. For that reason we need to modify Pampa in a way that tries to prefer hops of reasonable distance, which are not too close to the maximum receive distance (fragile routes) and not too close to the previous node (long routes). To create a buffer zone which prevents links from breaking too soon is also addressed by [DRWT96]. In other words we want to find a formula $Delay(distance)$ that uses the estimated distance from the previous node, to calculate a delay that results in more stable routes. Preferring hops of medium distance instead of maximizing it, in total we need more hops to reach each destination. However we expect to benefit from that investment, because we expect to get routes we can use for a much longer period of time before they break. A similar approach is proposed in [MM] to optimize AODV.

4.1.1 Delay function for Pampa

Finding a distance-based function

We assume that all nodes are equipped with omnidirectional antennas. Based on the implementation of wireless radio devices in Ns-2, using the *TwoRayGround* propagation model, we have the following theoretical background:
Signal strength power $(d) = \frac{F^2}{d^2}$, with d being the distance to the sender in meters, and $F = 0.01386645232$, a device-specific factor. To use this outside of Ns-2, the formula needs to be adapted to the used device. According to [WK97], a real formula is more complex than what Ns-2 emulates[1].
If we assume that nodes are uniformly distributed on the available simulation area, which of course is not exactly true using the random waypoint movement model, we conclude further that the number of nodes in a circle with a certain radius is proportional to the area of the circle and therefore to the square of the radius r:
Nodes $(r) \sim r^2$, the proportionality factor depends on the node density
One important point we have to consider is, how collisions can be avoided. Collisions occur if 2 nodes decide to forward a message at almost the same time. If we would create a function like delay $(d) \sim d$, the probability of collisions would also linearly increase, because the probability that other nodes in a critical area (in which nodes located there will retransmit with a too small time lag) causing a collision, also linearly increases with the distance to the sender. To avoid this we make delay $(d) \sim d^2$. As reasoned before, this makes the probability of collisions less dependant from the distance between sender and receiver. The aimed hop distance we want to make appropriate is called *target distance* and logically delay (d) must have its minimum value at the target distance. Therefore we propose this formula, which uses

[1]Namely they propose the formula $P_r = P_t \times \left(\frac{c_1}{d}\right)^n \times c_2$, device specific factors are the transmission power P_t, c_1 and c_2

the estimated distance d as parameter:

$$\text{delay}\,(d) = \begin{cases} F_1 \times (+1 - \left(\frac{d}{d_t}\right)^2) & \text{if } d < d_t \\ F_2 \times (-1 + \left(\frac{d}{d_t}\right)^2) & \text{if } d \geq d_t \end{cases}$$

The formula needs three constant values, d_t which is the target distance and the factors F_1 and F_2 which are used to get an appropriate range of the delay, making it also possible to balance whether hops smaller or instead bigger than the target distance shall be preferred. As appropriate values we chose $F_1 = 0.2$, $F_2 = 0.25$ and $d_t = 180$m. Those values were chosen based on some preliminary simulation results. Choosing F_1 and F_2 too low causes more collisions, high values increase the delay until the message got propagated to all nodes. We tried to choose a reasonable value in between the two extremities. Likewise choosing the target distance we want to avoid too long routes (small target distance, very stable, but many hops) and also too short routes (high target distance, few hops, but very fragile). However we did not investigate those parameters in more detail. The maximum transmission range used for the simulations was $d_{max} = 250$m, which is often used for simulations of 802.11 networks.

Considering the number of hops

The delay function mentioned in the last chapter may show unwanted behavior in some cases. For example it might happen that route A is extremely long, but was used because most of the hops are close to the target distance, and route B is quite short, but was delayed since some hops were smaller than the target distance. A possible modification of our formula to reduce wrong decisions, is to introduce a hop penalty. The assumption to do this is, that each node that becomes part of the route also increases the probability to fail sooner, because it is more likely that movement patterns occur that break the route. However this should not have too much influence on the delay to avoid long total delays on dissemination when encountering long routes. As a feasible solution we propose the following formula, based on the number of hops of the route:

$$\text{hop\_penalty}\,(h) = \frac{1.0 - (0.98^{h-1})}{5.0}$$

Assume the probability that a link breaks during a certain period of time is p_1. So the probability that it does not break is $1 - p_1$. The probability that a route with h links does not break is $p = (1 - p_1)^h$, since it consists of h links, each having the same probability to cause the combination of links (the route) to fail. We have not investigated the probability that a single link breaks, as it is highly dependant of the nodes' movement pattern. We used a value of 2 % (which explains the 0.98 in the formula), because the way, the resulting hop penalty increases with the number of hops, seems to be reasonable. The hop penalty is lower than 0.2 seconds, which is reasonable, compared to the power based part of the delay function. This upper bound can easily be changed by modifying the value 5.0 in the formula. We used $h - 1$ instead of h because the first receiver of a route job is already one hop away.
As we control the routes only by setting one delay value, we have to create a delay function which combines both components, a hop based one and a signal strength based one. As both

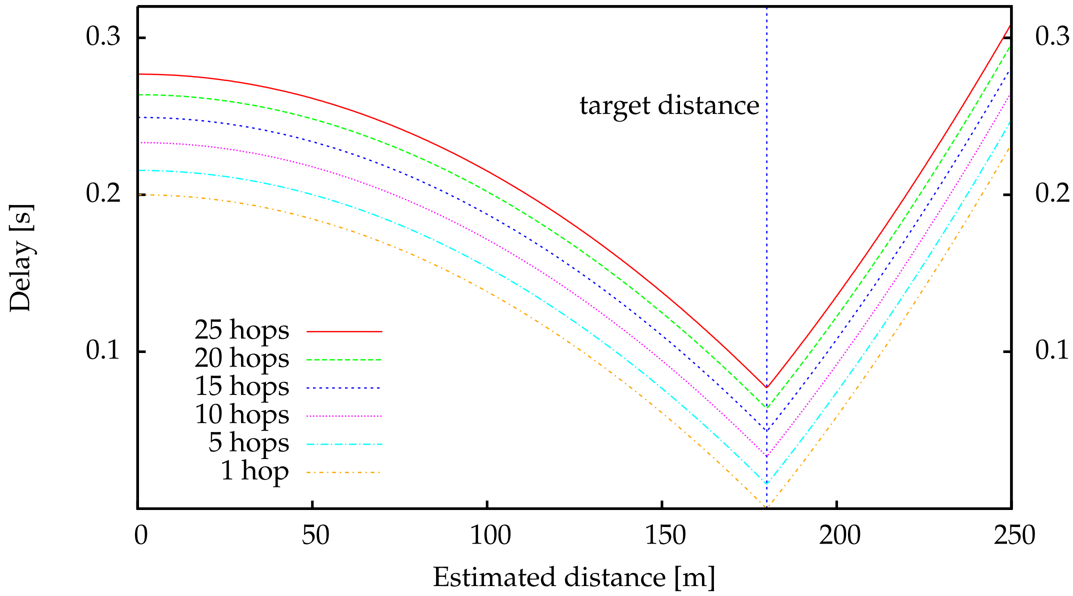

Figure 4.3: Pampa-Delay based on last-hop distance and number of hops

parameters are independent values, we just add both components to get the resulting delay function, depicted in Figure 4.3.

Transforming to a power based formula

Unfortunately a node does not know its distance to the previous transmitter, since we do not have location information (e.g. from a GPS receiver). Therefore we need to provide a formula, based on the signal strength power which is provided by the network driver of each node. Following our assumption and using the formula mentioned above, we get the following formula which is used by TPSR (p_t is the estimated receiving power at the target distance):

$$\text{delay}\,(p,h) = \begin{cases} F_1 \times (+1 - \left(\frac{p_t}{p}\right)) + \text{hop\_penalty}\,(h) & \text{if } p > p_t \\ F_2 \times (-1 + \left(\frac{p_t}{p}\right)) + \text{hop\_penalty}\,(h) & \text{if } p \leq p_t \end{cases}$$

Figure 4.4 shows this function. This formula is based on the assumption that each node sends with the same signal strength. This is required to provide p_t[2].

4.1.2 The number of retransmits

One key parameter of Pampa is the number of retransmits a node needs to receive, until it decides not to forward a message. Recall that, as explained in Chapter 2.1.4, Pampa(x)

[2] in Chapter 6.6 we explain how TPSR must be modified to support nodes using different radio equipment

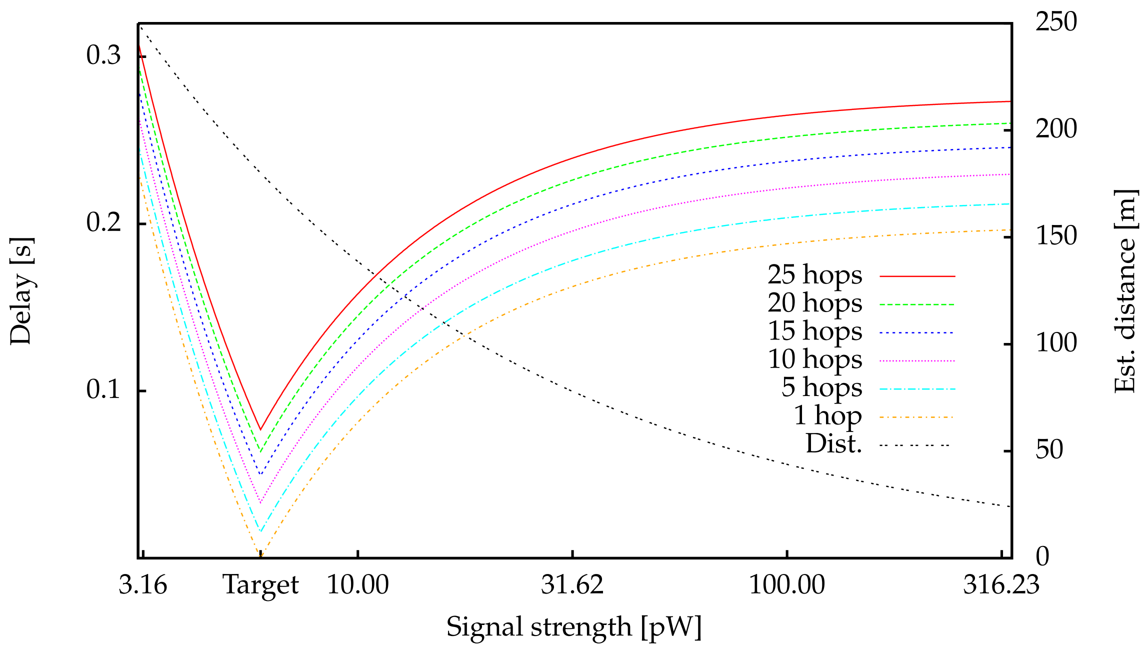

Figure 4.4: Pampa-Delay based on last-hop power and number of hops

means to use Pampa with nodes not retransmitting if "x" retransmissions are received. This parameter creates an obvious trade-off between delivery rate and the number of messages that are sent. Therefore we need to make a reasonable decision for this value. Simulations have shown that using Pampa(1), the delivery rate is much lower compared to Pampa(2) (presented in Figure 3.1). According to [Mir07], Chapter 3.3, Pampa(3) still provides a small benefit to the delivery rate compared to Pampa(1) and Pampa(2), which is assumed to be important. When subscriptions are not delivered this actually means that publishers might be unaware of existing subscriptions, which heavily impacts delivery rate. Additionally this offers more routes to each node, enabling it to choose the best of them. Using Pampa(2), each node must at least receive 3 messages (the first one plus 2 retransmits) to prevent his retransmission. Using Pampa(3) requires 4 copies to be received. This requires approximately 20 to 30 % more packets to be sent, compared to Pampa(2) [Mir07]. But because of the higher delivery rate we decided to use Pampa(3) to disseminate subscriptions and create routing trees.

4.1.3 Choosing source routes

Each node usually receives a subscription from several neighbors and can choose which of the routes he wants to use for packet delivery. An approach like AOMDV or DSR, that keeps different source routes in its routing table and enables to choose later the one which did not fail or is considered best, was not used for TPSR. The main reason for using only 1 route to each destination is that if a route fails, this means that the route was destroyed by the movement of some intermediate nodes, which is an expected behavior in MANETs. Because other routes that could have been used instead are of the same age as the failed route, we assume that they are already broken, too, or will break soon. However this was not investigated by simulations, so perhaps there could be some benefit keeping multiple routes. Chapter 6.5 addresses this problem. Each subscription contains a sequence number that allows us to decide whether a received route is newer than the one currently in the routing table. In that case, it will always replace the existing one. If it is older, it is always ignored. Being of the same age, TPSR uses this formula which is basically a modified version of the delay function shown in Figure 4.4:

$$\text{Priority}\,(p_{min}, h) = \begin{cases} \left(\frac{1-\frac{p_t}{p_{min}}}{10} + \frac{h}{5}\right)^{-1} & \text{if } p_{min} > p_t \\ \left(-1 + \frac{p_t}{p_{min}} + \frac{h}{5}\right)^{-1} & \text{if } p_{min} \leq p_t \end{cases}$$

Here we use the minimum power which occurred while creating the route, since this is the weakest point and therefore has the highest probability to break among all links our route consists of. If the minimum power is higher than the target power, we do not punish that (for good reason the delay function does, however), but divide it by 10 to prefer it not too much. This value is not more than a guess, whose influence needs to be investigated as part of future work.

4.1.4 Indirect updates

If a node receives a subscription and gets a new route, it can also check its routing table whether existing routes can be updated with the new information. The node not only received a source route to 1 subscriber, but also source routes to all intermediate nodes. It now inspects all entries of its routing table, to find all routes that contain nodes which are also part of the recently received source route. All entries it finds are replaced by the new version, because we assume that the new information represents a newer situation. This has 2 advantages. First of all this helps routes to persist longer, since the old part of the route is considered to be less stable than the new one. Second, it increases the probability that routes to different destinations use the same path, at least up to some intermediate node. This also decreases the number of sent packets, because crotches will probably occur later.

4.2 Delivery of publications

Each node maintains a routing table which contains the latest source route to each known subscriber. Based on this knowledge, it sends a single broadcast packet which contains one or more *route jobs*. The packet format is given in Figure 4.5. The number of route jobs is given in $\#rj$, $\#rec_x$ gives the number of of recipients of route job number x and $\#attr$ provides the number of publication attributes. Those numbers are necessary only to encode variable-sized sets of values. They enable the parser of the packet header to know how to interpret received data and depend on the implementation. The publisher chooses the addressees based on the source routes in his routing table. So actually the source route is only used to determine the next hops (addressees), which is a similar approach as used in DYMO [CBRP05]. In rare cases it may happen, that an addressee decides to use a different path than the one stored in the publishers routing table. This might happen for example if the delivery of a publication overlaps with a subscription disseminated or when the publisher did not receive a subscription renewal (which also means using an older path). Addressees receiving such

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th></th></tr>
<tr><td>$\#rj$</td><td>path quality</td><td colspan="6"></td><td></td></tr>
<tr><td colspan="2">addressee 1</td><td>$\#rec_1$</td><td colspan="2">recipient 1,1</td><td>...</td><td colspan="2">recipient 1,$\#rec_1$</td><td>} route job Nr. 1</td></tr>
<tr><td colspan="8">...</td><td></td></tr>
<tr><td colspan="2">addressee $\#rj$</td><td>$\#rec_{\#rj}$</td><td colspan="2">recipient $\#rj/1$</td><td>...</td><td colspan="2">recipient num_rj,$\#rec_{\#rj}$</td><td>} route job Nr. $\#rj$</td></tr>
<tr><td colspan="2">publication id</td><td>$\#attr$</td><td>attr. id Nr 1</td><td>value Nr 1</td><td>...</td><td>attr. id Nr $\#attr$</td><td>value Nr $\#attr$</td><td>} publication</td></tr>
</table>

Figure 4.5: Packet format for publication delivery

a message are responsible to forward it to the set of subscribers in their route job. To do this he also needs to calculate a new set of route jobs and broadcast the publication again. We assume that the previous node is also able to hear that broadcast. If that happens, the previous node knows that this addressee actually received the route job assigned to him. Hence it is interpreted as an acknowledgement. If the previous node does not receive such an acknowledgement before a timeout value, it assumes that the message was lost (perhaps a collision occurred) and will retransmit. If it does not receive an acknowledgement even after several retries, it assumes that the route is broken and will use Pampa to get the publication delivered anyway. The following chapters explain in details which routes are used and how nodes act to get publications delivered.

4.2.1 Creating route jobs

When a publisher decides to publish certain data, it checks his local cache of known subscriptions and creates a list of recipients which are (according to his knowledge) relevant subscribers. After that the publisher creates a set of what we call *route jobs* to be attached to the packet containing the publication. Each route job contains an addressee, which was (and hopefully still is) a direct neighbor of the packet sender. Additionally a route job contains a list of relevant subscribers, this addressee shall deliver the publication to. Basically the sender passes the responsibility to the addressees. Algorithm 4.1 shows how route jobs are created.

Algorithm 4.1 Create route jobs

```
procedure CREATERTJOBS(Recipients)
    DELIVERPUBLICATIONTOME()            // Method checks if we are subscribed, first
    routeJobs ← EmptyMap()
    delayIndex ← 0
    for all r ∈ Recipients do
        if r ≠ ThisNode() then
            sourceRoute ← (ThisNode() → routingTable[r])
            nextHop ← (sourceRoute → first() → next())
            if r ∉ routeJobs then
                routeJobs[nextHop] ← NewRouteJobs(ThisNode(), delayIndex)
                delayIndex ← delayIndex + 1
            end if
            routeJobs[nextHop] → insert(r)
        end if
    end for
    return routeJobs
end procedure
```

4.2.2 Delay

When we send a message with multiple route jobs, each addressee will (except if he is the only addressee of that route job) forward this message. To avoid collisions between those forwarded messages, each route job is assigned a delay index. This index tells the addressee at which delay he shall forward the message (index value 0 means immediately). That way we can contribute to avoid collisions and also get less delay, compared to if all addressees would select a random-only delay. The calculation depends on several constant parameters, shown in Table 4.1.

Name	Value	Description
DELAY	0.025	Steps of time to calculate the delay and timeout of route jobs, should be at least big enough to ensure that if one node sends half that delay after another node, it is impossible to get a collision between them. 'Half' because of some jitter (see JITTER).
TIMEOUT_JITTER	3	How much jitter is added to timeout values, between 0 and the provided value, in multiples of DELAY.
TIME-OUT_FACTOR	4	By what factor is the timeout increased (multiple of DELAY) to calculate the timeout value
JITTER	$\pm 25\%$	How much jitter is applied to every delay calculation, percentage of DELAY

Table 4.1: Parameters for delay calculation

Using these variables, we propose this formula to calculate the delay:

$$\text{Delay}\,(index) = \text{DELAY} \times \max\left(0, index + \text{RandomNumberBetween}\,(-0.25, 0.25)\right)$$

Ns-2 is completely deterministic, which may lead to unrealistic results. For instance, Ns-2 assumes that calculations need no time (infinite cpu speed) and that the packet delay only depends on distance, but in reality we also have delays produced by the network link layer or hardware drivers. To avoid side effects due to these problems, we use a short random part of the delay calculation.

4.2.3 Timeout values

We also need a timeout, after which we expect to have received an acknowledgement from each addressee we sent a route job to. If we did not hear an addressee rebroadcast the message (which is interpreted as acknowledgement) before that timeout expires, we assume that the addressee did not receive it. This may happen either because of collisions or because the node moved too far away. If that happens, we rebroadcast our initial message again, containing all route jobs we did not receive an acknowledgement for. Used parameters are listed in Table 4.1. The delay depends on the total number of recipients of all route jobs we sent, which is called $num_{rec.}$:

$$\begin{aligned}\text{Timeout}\,(num_{rec.}) &= \text{DELAY} \times (num_{rec.} \times \text{TIMEOUT\_FACTOR} + 1.0) \\ &\quad + \text{DELAY} \times \text{RandomNumberBetween}\,(0, \text{TIMEOUT\_JITTER})\end{aligned}$$

Using this formula, timeout values have the highest value at the publisher and decrease on their way to the subscriber. This is expected to provide a marginal advantage if a message, an intermediate node sends causes a collision and neither the previous node, nor the next node are able to receive it. In that case it is better if the intermediate node times out before the previous one, preventing the latter from retransmitting (because he hears the retransmission of the intermediate node just in time). The value of 4 however is only an estimation that is greater than the jitter, which cannot exceed 3. Algorithm 4.2 shows how the timeout formula is used by *TPSR*.

Using broadcast messages to deliver publications affects the performance of TPSR. 802.11 networks provide collision avoidance and detection functionality [61802, TMS03]. However this is only available if messages have a single destinatination. Using this functionality for TPSR would partially replace our solution that uses timeout values and that retries if no acknowledgement was received after the timeout expired. However the 802.11 colission avoidance has some drawbacks if used for TPSR:

- Additional messages are sent, in particular RTS, CTS and ACK
- For each addressee a dedicated message must be sent, which increases traffic
- Most of the traffic is related to flooding and therefore must use broadcast messages. However broadcast messages do not care about collision avoidance of unicast messages. So the benefit would be rather small, as collisions are only avoided between publication messages.

Algorithm 4.2 Delay queue timeout

```
procedure QUEUETIMEOUT(queueEntry)
    routeJobs ← (queueEntry → routeJobs())
    needAck ← (queueEntry → needAck())
    if not needAck||routeJobs ≠ ∅ then
        sendMe ← true
        cachedPacket ← (queueEntry → packet)
        if not needAck then
            if queueEntry → numRetries > maxRetries then
                                // for maxRetries, see Table 5.2, "Max. # of retries at routejobs"
                sendMe ← false              // Give up existing route, use Pampa as fallback
                publication ← (cachedPacket → publication)
                FLOODBYPAMPA(cachedPacket, publication → issuer, publication → uid)
            else
                timeout ← Timeout (routeJobs → NumRecipients())
                RESCHEDULEQUEUEENTRY(queueEntry, timeout)
            end if
        end if
    end if
end procedure
```

Since most addressee are not subscribers but only forward the message, the forwarded message can be interpreted as an acknowledgement by the previous node. When using 802.11 unicast messages, we were not able to benefit from this specific traffic pattern. We can conclude that our decision to use broadcasts improves the efficiency of TPSR.

4.2.4 Fallback using Pampa

If a node retried several times, until it reached the maximum number of retries, it gives up trying the known route. After the timeout value has expired once again, the publication is flooded using Pampa(2). One reason is of course to deliver that message anyway, although with huge effort compared to the route-based approach. The more important reason is, however, to notify the subscriber that the route is broken and needs to be renewed. All subscribers that see this publication for the first time will immediately resubscribe. This is explained in the next section in detail.

Algorithm 4.3 Receive Routejobs

```
procedure RECVROUTEJOBPACKET(receivedPacket)
    routeJobMap ← (receivedPacket → routeJobMap)
    sourceRoute ← (receivedPacket → sourceRoute)
    if routeJobMap[ThisNode()] ≠ NUL then                    // This node is an addressee
        delayIndex ← (routeJobMap[ThisNode()] → delayIndex)
        recipients ← (routeJobMap[ThisNode()] → recipients)
        newRouteJobs ← createRtJobs (recipients)
        needAck ← (newRouteJobs ≠ ∅)     // This is a leaf node and only ack's to prev. node
        newPacket ← newPacket (newRouteJobs, sourceRoute)
        INSERTINTODELAYQUEUE(newPacket, delay (delayIndex) , needAck)
    else if ThisNode() ∈ sourceRoute then
        lastHop ← (sourceRoute → first() → next())
        cachedPacket ← (delayQueue → GetEntry (lastHop))
        if cachedPacket ≠ NUL & cachedPacket → routeJobMap[lastHop] ≠ NUL then
            DELETE(cachedPacket → routeJobs[lastHop])
        end if
    end if
end procedure
```

4.3 Maintaining source routes

4.3.1 Reactive route repair

As stated in the last section and shown in Algorithm 4.2, when a route job cannot be delivered after several tries, it uses as its last option Pampa to get the publication delivered. Pampa uses 2 values to distinguish different broadcasts, the unique node id from the issuer and a sequence number created by the issuer. In normal operation, these values are set by the issuer (the first node that broadcasts the content). Using this mechanism in combination with route repair may however lead to high traffic. Because several nodes may encounter a broken route and give up, the same publication could get flooded many times, which is unnecessary. Instead of using the real issuer, we use the originator of the publication instead. And instead of a newly created id, the publications uid is used. This way, when more than one node decides to give up, the different flooding waves usually do not overlap much. Pampa recognizes packets as duplicates, although they were issued by different nodes that gave up routing. If the publication was disseminated over the whole network, and a node then decides to give up, this decision does not lead to a new dissemination, which effectively reduces traffic a lot. According to Chapter 3.3.2 each dissemination with Pampa(2) causes about 40 % of the nodes to retransmit. If we look at the evaluation results, the number of give ups is always much higher than the subscriptions actually sent. This means that for a single broken route often more than one node decides to give up and to use Pampa(2). So for each node that wants to use Pampa(2) to deliver publications, beside the first node that decided to do that, we save many packets. In our simulations this easily reached a magnitude of tens of thousands, which is a significant amount of traffic we can save. If a subscriber receives a publication from Pampa, and it did not send a subscription during the last 8 seconds[3], it will immediately resubscribe. The block time is necessary to avoid frequent resubscriptions, just because several different publications – either from different publishers, or from the same publisher, sending at a high rate – could not be delivered. The decision to immediately resubscribe is based on the assumption that at least 1 route from a publisher is permanently broken. Of course this assumption is not always true. For example a high collision rate may also be interpreted as a broken route. However it seems that in most cases our assumption holds. We call this "reactive route repair", because we react by trying to repair already broken routes.

4.3.2 Proactive route repair

The second concept to keep routes stable is a proactive approach. If publications are delivered using route jobs, we always keep track of the worst connection link (i.e. the route quality), too. Each node that forwards a publication message also checks whether the estimated distance (based on the signal power), it received the message with, is greater than the route quality provided in the message header. If that is the case, the existing value will be replaced by the value of the distance to the previous hop. Hence a subscriber that receives a publication also knows the worst link that occurred on the publishers known source route to him. The subscriber is now able to decide whether he shall immediately resubscribe or not. To evaluate the quality of a route is not trivial. If we resubscribe too early, we also decrease the intervals we subscribe at, which produces a lot of additional traffic. On the other hand, resubscribing too late, gives no benefit any more. A reasonable threshold probably also depends on the expected node speed (because speed is expected to destroy routes faster), the publishing rate (due to increased sampling rate at which we can check the route) and the movement model. We propose the following formula, that depends on the estimated maximum distance that occurred on a source route $d_{route,max}$, using the target distance d_t and the maximum receive distance d_{max}:

$$\text{routeIsBad}\left(d_{route,max}\right) \quad = \quad \left(d_{route,max} > \text{RECOVERY\_LIMIT} \times \left(d_{max} - d_{target}\right)\right)$$

For the recovery limit we used a value of 15% which seems to be reasonable. It is left for further research to investigate what values are best, based on the various scenario parameters.

[3]see Table 5.2, "Resubscribe blocking time"

5 Evaluation

In this chapter we present and discuss the evaluation of TPSR. The contributions of the evaluation are two-fold:

1. To understand how TPSR performs under different networking conditions, finding its limitations and most suitable execution environments.
2. To compare TPSR performance with another approach, in particular with P-PUBS that, although being a brute-force presented the best performance in the preliminary evaluation.

Evaluation is performed by varying some parameters affecting the networking environment. Each parameter was varied individually, while keeping the remaining in some predefined default value, thus generating a multitude of network conditions. For each point depicted in a figure, 10 equivalent but independent scenarios where defined. The results presented in this chapter average these 10 independent runs, thus attenuating possible bias in the evaluation which could be introduced by a single run exhibiting particularly favorable (or unfavorable) conditions.

The chapter starts with the presentation of the metrics used by the evaluation and of the parameters that have been set for the simulations. It then proceeds by presenting and discussing the evaluation results for each of the parameters that has been addressed.

5.1 Metrics

To satisfy the goals of the evaluation, the 6 metrics presented below were used:

Delivery Rate The delivery ratio is defined as the proportion of subscriptions that were successfully delivered to interested subscribers. That is, let s_p be the number of subscribers known to be interested in some publication p and r_p be the number of interested subscribers receiving publication p. The delivery ratio dr is therefore defined by:

$$dr = \frac{\sum\limits_{\forall p \in P} r_p}{\sum\limits_{\forall p \in P} s_p}$$

Where P are all publications issued during a simulation. A precise definition of interested subscribers, that takes into account the delay in subscription propagation, is deferred to Section 5.3.3.

Sent packets The number of packets transmitted is a factor that significantly affects the lifetime of the devices [Fee01] and therefore, the network performance. Results for this metric show the accumulated number of transmissions performed by all nodes in the network and therefore, provide an intuition on the lifetime of the devices and on the traffic produced.

Loss Rate For each transmission t, this metric accounts the number of nodes that, although in transmission range of the sender of t, have not received the transmission. This metric serves mostly as an indication of congestion in the network, given that the most common reason for some node not receiving a message are collisions. To formally express this metric, we consider functions *neigh(n)* which returns the number of nodes in transmission range of node n and *receiv(m)* that returns the number of nodes that effectively received some message m.

$$lr = 1 - \frac{\sum\limits_{\forall m \in M} receiv(m)}{\sum\limits_{\forall m \in M} neigh(S_m)}$$

with S_m being the transmitter of message m and M being all Messages sent during a simulation.

Route give-up's/retries These metrics, which are only relevant for TPSR, account the number of broken links found in the simulation. As described in the previous chapter, a retry will occur if after some predefined timeout, some node was not able to confirm the reception of a publication message by an addressee. If the established number of retries is exceeded, a node broadcasts the publication and this event is accounted as a give-up. In all simulations a give-up will occur after two unsuccessful retries. Therefore, a constant to be observed in the evaluation is that the number of give-ups must be lower or equal to half the number of retries.

Knowledge of existing subscriptions This metric evaluates the success of TPSR in propagating subscription messages to relevant publishers. If publications are not delivered the reason might be either that TPSR was not able to deliver them or that it was not aware of all interested subscribers and therefore did not even try to deliver it. Our metric allows us to distinguish between those two cases. It accounts the proportion of subscribers that each publisher is aware of at the moment of the publication[1]. Defining the function *aware(p)* which returns the number of interested subscribers that the publisher of publication p is aware when the publication is issued and s_p as the number of effective subscribers interested in p, the "knowledge of existing subscriptions" (ks) metric would be given by:

$$ks = \frac{\sum\limits_{\forall p \in P} aware(p)}{\sum\limits_{\forall p \in P} s_p}$$

[1]Taking into account the delay in subscription propagation described in Section 4.1 by considering only interested subscribers, as explained in Section 5.3.3

Where P is the set of publications issued during a simulation. It should be noted that in P-PUBS, the delivery rate is not affected by a lack of subscription knowledge, since subscriptions are not distributed at all. In the plots, this is represented by assuming that in P-PUBS nodes have a perfect knowledge (100 %) of the relevant subscriptions.

Issued subscriptions This metric evaluates the total number of calls of the Subscribe() function invoked by all participants in a simulation. In the case of P-PUBS, it only depends of the number of times each subscription is refreshed to prevent its expiration and of the number of changes in subscriptions. This serves as a control value for comparison with TPSR where in addition to the periodic subscription refreshments and changes, subscriptions may be initiated by the proactive and reactive route repair functionalities described in the previous chapter.

This set of metrics is expected to provide a good overview of TPSR performance. The impact of the protocol in the network can be observed by the *Sent packets* and *Loss rate* metric which evaluate the number of sent messages and collisions. These are two standard metrics frequently used for evaluating routing protocols. The efficiency of the protocol is better evaluated by the *Delivery Rate* metric. Finally, the *Route give-up's/retries*, *Knowledge of existing subscriptions* and *Issued subscriptions* give a better understanding of the impact on the design choices.

5.2 Network Simulator 2

For the simulations Ns-2 version 2.34 [ns2] was used as platform. Ns-2 is a simulator for network research and published under the BSD license. It is written in C++ and uses OTcl as command and configuration interface. Its manual is published at [FV]. The Ns-2 codebase was extended with:

- PAMPA-Agent [pam], using a modified delay function
- NOAH [noa]
- Self-implemented Publish/Subscribe system.

Ns-2 needs to use exactly one routing protocol. Using a protocol which is distributed as part of Ns-2 (like AODV or DSR) has some disadvantages. For example those routing protocols attach their own header to broadcast packets, increasing packet sizes. This in turn makes collisions more likely. According to some preliminary tests, the collision rate using P-PUBS increases up to about 50 %. Therefore the only option to avoid this unnecessary overhead was to use NOAH. NOAH is basically a dummy routing protocol, which just sends packets without modification, leaving it to the receiving nodes to interpret them. Simulating Mobile Ad Hoc Networks, there are many parameters we have to provide to Ns-2 or the Pub/Sub system we implemented. These parameters are shown in two tables to provide an overview to the reader. Most of them were explained in Chapter 4. The remaining ones only apply to the evaluation and are explained in the current chapter.
Table 5.1 depicts the parameters that deserved further investigation as they were expected

to mostly influence simulation results. For each of them we show and explain the results in Chapter 5.5. The default values apply, if that parameter was not varied and we did not state otherwise.

Attribute	**Min.**	**Max.**	**Default**	**Comments**
Node density	50	200	140	nodes/km^2, steps of 30
Max. node speed	0 m/s	10 m/s	3 m/s	steps of 1
Publishing rate	0.1/s	1.5/s	0.5/s	steps of 0.2/s, rate per publisher
RWP pause time	0 s	600 s	200 s	steps of 200 s
Number of publishers	0	30	10	steps of 10
Number of subscribers	0	30	10	steps of 10
Changing subscriptions after	15 s	3840 s	60 s	log. steps, redeciding may choose same sub. again
Location dependency of subscriptions	0 %	100 %	100 %	steps of 25 %
The following simulations vary also the node density, but at changed scenarios				
Hotspots randomly placed	50	200	140	
Manhattan Grid Model	50	200	140	
Stuttgart City Model	50	200	140	

Table 5.1: Evaluation parameters with variable values

For parameters we think are less important or have been discussed earlier (for example in Chapter 3.2.2), we always used a constant value. They are shown in Table 5.2.

5.3 Choosing Subscriptions

5.3.1 Hotspots

In our system model we assume that the interest of subscribers depends on their location (see *GEO* in Chapter 3). We introduce the concept of a *Hotspot*. Each hotspot is assigned to a certain point on the simulation area. The hotspot is associated to a subscription template. In our simulations templates do not overlap, so a publication will never match the subscription template of more than 1 Hotspot. To ensure this, we use as many attributes as hotspots. We can then assign hotspot number i at attribute number i a value range which is disjoint with that of all other hotspots. In Alg. 5.1 we show this algorithm for the grid-based placement of hotspots, which is used by default. The algorithm used for random placement, used in Chapter 5.5.10, is not explicitly provided, but quite similar. Instead of the number of rows and columns it just gets the number of hotspots that shall be created. The values pos_x and pos_y are chosen by random within the simulation area.

Attribute	Default value	Description
Number of nodes	200	
Timeout of subscriptions	200 s	
Resubscribe after	45 % of timeout	
Payload size of a publication	16 Bytes	
Max. # of retries at routejobs	2	
Subscribers 1^{st} subscription (avg.)	80 s	Only applies if "Changing Subscriptions after" is bigger
Delay statistics after changing subscription	5 s	
Size of an attribute id	1 Byte	
Size of an attribute value	4 Bytes	
Publications without Hotspots	0 %	
Pampa retransmits to create routes	3	
Pampa retransmits otherwise	2	
Target distance of TPSR	180 m	
Max. receive distance	250 m	
Network links	802.11 @11 MBit/s	
Ns-2 propagation model	TwoRayGround	
Subscription overlap of Hotspots	None	
Number of hotspots	4	
Resubscribe blocking time	8 s	Explained in Chapter 4.3.1
Route recovery limit	15 %	Explained in Chapter 4.3.2
Target grace factor	0.9	

Table 5.2: Evaluation parameters with fixed values

5.3.2 Hotspot-based choosing

To simplify evalution each subscriber is never subscribed to more than 1 subscription. This allows us to abandon the $Unsubscribe(s)$ method mentioned in Section 1.2. Therefore if a subscriber is subscribed to s_1 and calls $Subscribe(s_2)$, he is indirectly unsubscribes from s_1 before. That is the $Subscribe()$ method replaces an existing subscription (if present).

When a subscriber wants to decide which subscription to use, this decision defaults to be location dependent. More precisely, each subscriber checks its current position on the location area and calculates the distance to each hotspot. Then the reciprocal inverse provides the probability that a certain hotspot is chosen. This is shown in more detail in Algorithm 5.2. Probabilities are used to define a more realistic model, avoiding the use of a strict border where interest switches, just because a node moved a few meters. Our approach considers both, subscribers are more likely to use a subscription which is located close to them, but are

Algorithm 5.1 Create grid-aligned hotspots

```
procedure ADDHOTSPOTGRIDSET(cols, rows)
    hotspotId ← 1
    iMin ← MIN_VALUE
    iMax ← MAX_VALUE − 1
    aMin ← MAX_VALUE
    aMax ← MAX_VALUE
    for row = 1 to rows do
        for cols = 1 to cols do
            sub ← newSubscriptionTemplate()
            for i = 1 to rows * cols do
                if i = hotspotId then
                    sub → addAttribute (i, aMin, aMax)
                else
                    sub → addAttribute (i, iMin, iMax)
                end if
                pos_x ← (AreaSizeX()/rows) × (row − 0.5)
                pos_y ← (AreaSizeY()/cols) × (col − 0.5)
                SAVENEWHOTSPOT(pos_x, pos_y, sub)
            end for
            hotspotId ← hotspotId + 1
        end for
    end for
end procedure
```

also free (although at less probability) to choose a hotspot which is more distant. We believe that this mode is close to scenarios like a rescue scenario, for example many fire workers are supposed to be interested in similar information and tend to concentrate in certain regions like command bases or rescue teams. A medical emergency team will probably also use a mobile rescue center, causing many MANET participants there being interested in information relevant to emergency doctors. Those scenarios share in common that proximity of subscribers influences their interest, while publishers are likely to be more spread.

By default we always use the location-based approach. However to investigate how this affects the performance of TPSR, we added the input parameter "location dependency of subscriptions" (*locdep*), ranging from 0 to 100.

Each subscription has a timeout value, currently set to 200 seconds. A publishers that does not receive an updated subscription during that time, will cease to send publications to this subscriber after the timeout value expired. As it may happen that a subscription update is lost or not delivered, it seems a good idea to make the interval in which subscriptions are issued lower than the timeout. We decided to use 45 % of the timeout value as a threshold after which the subscriber will reissue his subscription. If subscriptions change not very often, this allows a publisher to miss a subscription update and still sending to the subscriber, assuming that subscriptions did not change. As it is very unlikely to miss 2 consecutive subscription updates, it should happen rarely that subscriptions time out at the publisher.

When a subscriber chooses a subscription different to his current one, he will immediately issue this new subscription to make all publishers aware of this change. When a subscriber redecides his subscription it may happen that he chooses the same subscription again. In that case it does not reissue a subscription to avoid unnecessary traffic. A lower $locdep$ value biases the algorithm by reducing the possibility to combine routes, since it becomes less likely that paths to different subscribers go in similar directions.

5.3.3 Statistics on changed subscriptions

When a subscriber issues his first subscription or changes his subscription later, it takes some time (usually a few seconds), until it is propagated to all publishers. During this period a publisher is not aware of the current subscription of this subscriber. Hence it would send the wrong publications, or even no publications at all, to him. Additionally even if the publisher would know immediately about the updated subscription (which is of course impossible), we also would have had to consider the delay that applies to the publication delivery. Even with this perfect knowledge at the publisher, the subscriber would have to wait a few seconds after he issued his subscription before he is able to receive the first publication. So the delivery of the first publication will be delayed by the time to disseminate the subscription to all nodes and additionally the time to deliver publications. In our simulations this is only a few seconds, but may noticeably impact the delivery rate. Concerning this, P-PUBS is in advantage. It sends all publications to all nodes. When one of them changes his subscription he can immediately start to pick the relevant publications that pass by.
It is a common principle that a node that registers to receive certain traffic notices a short delay until he receives the desired data (a well-known example is IP Multicast). So we think that this delay is expected behavior that should not impact delivery rate. To handle these delay issues of TPSR, we use global knowledge to decide which publication we expect to get delivered and which not. As this only affects statistics and not the algorithm itself, this is feasible. When a publication is issued, we check all existing subscriptions (using global knowledge). For each subscription we check whether the publication matches its attributes and also whether this subscription is older than 5 seconds (using a global time source). If both are the case, this publication is registered that it is expected to get delivered to this subscriber. If a subscriber receives a publication, this is counted as delivered if and only if it has also been registered for this subscriber. We follow 2 basic ideas. First, we assume that the point of time, a publication is issued, decides which subscriptions might be relevant. This means that a subscriber changing his subscription from A to B is still interested in publications for A issued before he switched to B. Likewise he is not interested in publications for B, issued before he actually changed his subscription to B. Second, we allow a delay of 5 seconds to get all publishers aware of a changed subscription, although the delay is usually much lower.

We call a subscriber s *interested* in some publication p if p is registered as a relevant publication of s according to the criteria's mentioned before.

Algorithm 5.2 Choose subscription

procedure GETNEWSUBSCRIPTION($pos_x, pos_y, locdep$)
  if RandomNumberBetween(0,100) $< locdep$ **then**
                // Choose subscription based on node location
    $\mathsf{nnprob_{sum}} \leftarrow 1$ // Non-normalized total sum of probabilities
    for $\mathsf{id} = 1$ to $num_{hotspots}$ **do**
      $\mathsf{hs} \leftarrow \text{getHotspotById}(\mathsf{id})$
      $\mathsf{distance} \leftarrow \sqrt{((\mathsf{hs} \rightarrow pos_x) - pos_x) + ((\mathsf{hs} \rightarrow pos_y) - pos_y)}$
      $\mathsf{nnprob[id]} \leftarrow \frac{1}{\mathsf{distance}}$
      $\mathsf{nnprob_{sum}} \leftarrow \mathsf{nnprob_{sum}} + \mathsf{nnprob[id]}$
    end for
    $\mathsf{nnval} \leftarrow \text{RandomNumberBetween}(0, \mathsf{nnprob_{sum}})$
    for $\mathsf{id} = 1$ to $num_{hotspots}$ **do**
      $\mathsf{nnval} \leftarrow \mathsf{nnval} - \mathsf{nnprob[id]}$
      if $\mathsf{nnval} < 0$ **then**
        LEAVEFORLOOP
      end if
    end for
  else
    $\mathsf{id} \leftarrow \text{RandomNumberBetween}(0, num_{hotspots})$ // Choose subscription randomly
  end if
  $\mathsf{hs} \leftarrow \text{getHotspotById}(\mathsf{id})$
  return newSubscriptionForHotspot($\mathsf{hs}, ThisNode()$)
end procedure

5.4 Calculation of packet sizes

In Ns-2 it is a common principle that packet data is not sent in a format that would be usable for an implementation on real devices. This allows developers to use data structures that are easy to access (e.g. C++ class objects) instead of converting everything to a compact byte array representation. To allow accurate simulations however, Ns-2 needs to know the size of the packet as if it was converted to a compact format. This size value must be stored in the hdr_cmn::access (Packet*) ->size () variable. In this section we explain, how the size of a packet was calculated and provided to Ns-2. Table 5.3 shows the fixed values used, table 5.4 shows what values are part of which elements. To calculate the size of the packet we simply sum up all element sizes which are part of the packet.

The following data can be emulated by or extracted from the existing encoding scheme:

- Ns-2 port number: Could also be encoded within the packet type
- Route job delay: indirectly given by the order of the route job list
- Subscription id: We could use the UID instead
- Publication id: We could use the UID instead

Variable	Value	Comments
SIZE_PUB_DATA	16	Payload data of publications
SIZE_SUB_ATTRID	1	attribute id, the next value belongs to
SIZE_SUB_TIMEOUT	2	Timeout value (1-second steps)
SIZE_PUB_VALUE	1	Attribute value
SIZE_NUM_ATTRS	1	Number of pub/sub attributes
SIZE_NODEADDR	2	Unique node id
SIZE_SUBID	2	subscription sequence id
SIZE_UID	4	unique id
SIZE_PTYPE	1	packet type
SIZE_DISTANCE	1	estimated distance to a neighbor in meters
SIZE_HOPS	1	Number of hops
SIZE_SOURCEROUTE	1	Number of source route entries
SIZE_NUMROUTEJOB	1	Number of route jobs
SIZE_NUMROUTEJOBE	1	Number of route jobs entries

Table 5.3: Sizes of elements that are part of packets

Element	Used variable	Comments
Common	SIZE_NODEADDR	
	SIZE_UID	
	SIZE_PTYPE	
	SIZE_NUMROUTEJOB	Number of route jobs, if applicable
Publication	SIZE_NODEADDR	Originator node
	SIZE_PUB_DATA	
	SIZE_NUM_ATTRS	
	SIZE_SUB_ATTRID +SIZE_PUB_VALUE	For each specified attribute
Subscripion	SIZE_NODEADDR	Originator node
	SIZE_SUB_TIMEOUT	
	SIZE_NUM_ATTRS	
	SIZE_SUB_ATTRID +2×SIZE_PUB_VALUE	For each specified attribute
Sourceroute	SIZE_SOURCEROUTE	Number of entries
	SIZE_NODEADDR +SIZE_DISTANCE	For each entry
Routejob (occurence-dep.)	SIZE_NODEADDR	Originator node
	SIZE_NUMROUTEJOBE	Number of destination nodes
	SIZE_NODEADDR	For each destination node

Table 5.4: Components of packets

- Many other things which are only used for statistics purposes or can easily be derived from other values.

5.5 Results

5.5.1 Varying node density

Average node density (given in km^2) was changed by varying the simulation area while keeping the number of nodes constant at 200. The edge sizes of the simulation area where set by the formula $edge\_size = 1000 \times \sqrt{\frac{200}{density}}$
The average node density we used assumes that all nodes are uniformly distributed, which is usually not true. However this gives us a good estimation about how sparse the network is. The results are shown in Figure 5.1. We can see that the delivery rate is almost the same, with small advantages for TPSR at sparse networks. The fact that the delivery rates at density 50 are similar is interesting if we take the subscription knowledge into account. Although TPSR knows only about 66 % of the subscribers, it delivers about 44 %, which gives an effective delivery rate[2] of about 67 %. This is much higher than P-PUBS and confirms that even in sparse networks it is possible to create useful routes, because TPSR uses Pampa with 3 retransmits to create routes, while P-PUBS only uses 2 retransmits.
If we look at the number of sent packets, we see a clear benefit of TPSR at least between a density of 110 and 200. One interesting point is, that at a density of 50, both algorithms send fewer packets than having a density of 80. At P-PUBS we attribute this result to the small delivery rate, which indicates that many disseminations stop prematurely because of the sparse network. Using TPSR similar situations occur if Pampa is used to deliver publications after a route failed. Additionally the knowledge of existing subscriptions is about 30 % lower, which actually means that TPSR does not even try to deliver to one third of the interested subscribers, so we can assume that the publication delivery of TPSR is much better than that of P-PUBS, but the little knowledge compensates this benefit. The loss rate seems to be almost the same.
At higher densities TPSR is slightly better, because it sends fewer packets. To give up we always retry twice. This explains why the number of retries must always be at least twice the number of give ups. The plot shows that in the majority of cases, if we retry once, we are very likely to fail the second time, too. The number of retries that gets an acknowledgement is about 1000, agnostic to density, which can be derived from the shown graphic. A low delivery rate impacts also the number of subscriptions sent, because we currently have no recovery mechanism if subscription message got lost. If a publisher is not aware of a subscription, he will not try to send any packets to that, neither using routing nor using Pampa. So the subscriber is not able to detect that a required route does not exist. The publisher just waits until the subscriber recreates routes after his subscription expired. This effect reduces the number of sent subscriptions at very low node densities, in the plot visible at a density of 50. Increased density also makes routes smaller and better (because there are more nodes

[2]*effective delivery rate* means the delivery rate in relation to the subscription knowledge

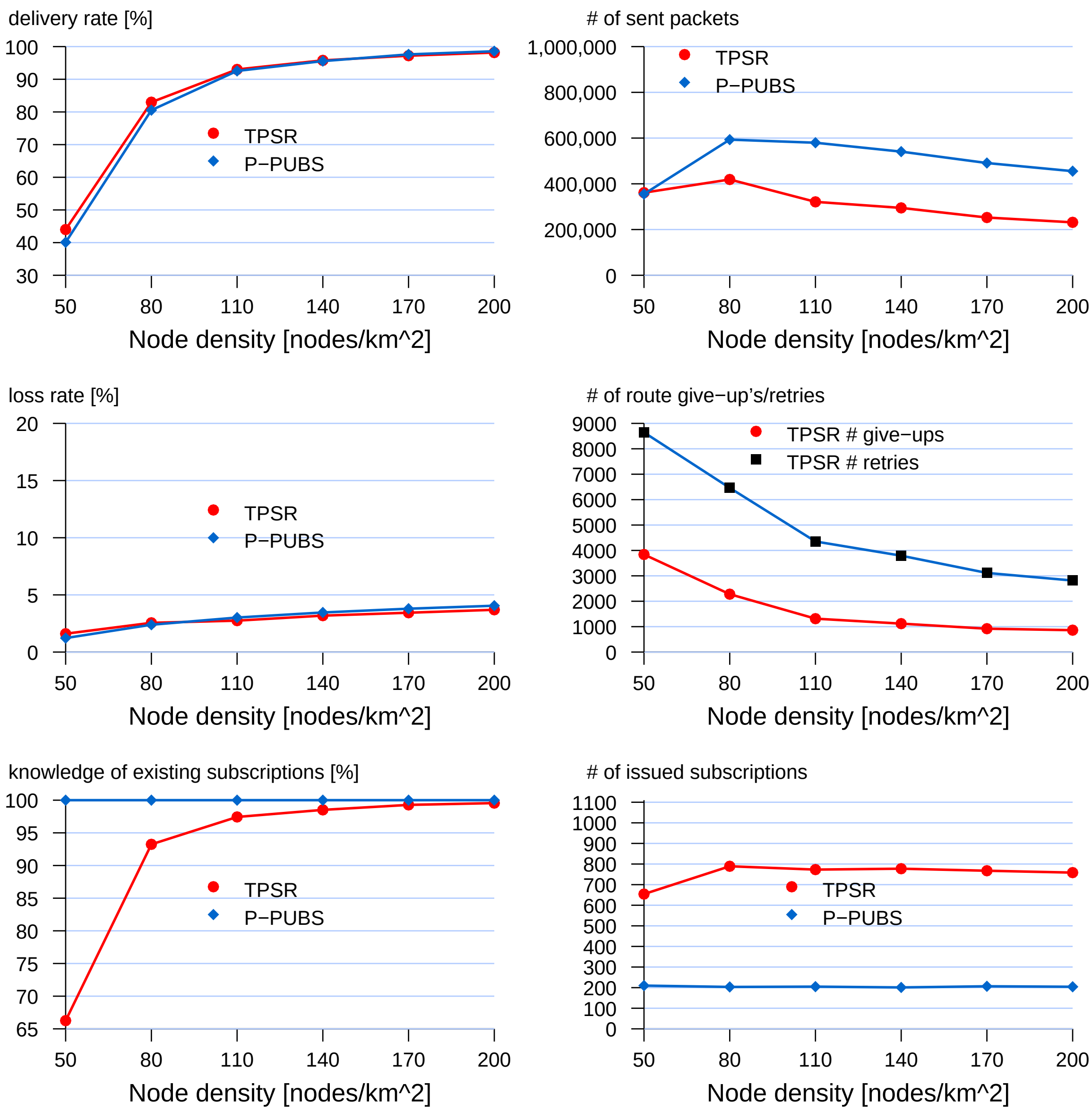

Figure 5.1: Comparison Pampa/TPSR with varied node density

which allows to choose a better one), which decreases the number of sent subscriptions a bit, because in average routes persist longer.
To sum up we can say that at node densities above about 100 nodes per square kilometer, TPSR provides good benefit compared to P-PUBS. We chose 140 as default density, because we think this is a reasonable value for real scenarios that provides good potential of optimization. A second conclusion of this simulation is that TPSR performs not very good, if the network is too sparse. At least it seems we does not waste packets compared to T-PUBS. Checking densities lower than 50 probably does not make much sense, because this would decrease the delivery rate even more, which makes this scenario almost unusable.

5.5.2 Varying maximum node speed

Using the random waypoint movement model we have to provide a minimum and a maximum speed. For each movement operation, the speed is randomly chosen between these two boundaries. The minimum speed is necessary to avoid a continuous slow-down of all nodes [BRS03]. If the minimum node speed was set to 0, any node choosing a speed very close to 0 will perhaps need hours until he reaches the given destination. Because this is not what we want, a minimum speed of 0.5 m/s was used. The only exception is of course when using a maximum node speed of 0. In that case the minimum speed was also set to 0. We have to mention that a speed of 0 is not really random waypoint any more. Random waypoint starts with uniformly distributed node locations. But because of the characteristics of RWP, nodes tend to concentrate in the center of the simulation area [BRS03]. If we would run a simulation starting with uniformly distributed nodes, we would get different results, since the node distribution at the beginning of the simulation is different to the one we get after nodes were moving for some time. We would get different results just by varying the simulation time, as the influence of different node distribution on startup would be different. To avoid such problems, it is a common practice to specify a time interval that cuts off the first part of the scenario. We used 3600 seconds for this. To generate a scenario, nodes are placed uniformly, then move for 3600 seconds according to the given speeds and pause time values. After that the current position of each node is calculated and written to the scenario file as its initial position. Henceforward movements are written to the scenario file for another period given as the desired simulation time. Using this warm-up phase works fine, with one exception. When a node speed of 0 m/s is used, 3600 seconds later all nodes are still at their initial position, uniformly distributed. This explains why the values shown in the plots of Fig. 5.2 differ at that speed. Nevertheless this value provides useful information. First we can see that this improves the delivery rate, because we do not have sparse boundary areas any more. Due to that, Pampa covers a larger area, which increases the traffic. It seems obvious that TPSR can gain most if nodes do not move. In case of non-moving nodes P-PUBS needs 365 % more packets. Even at a maximum speed of 10 m/s, TPSR is better, although it is expected to become worse at higher speeds.

5.5.3 Varying publishing rate

The results for different publishing rates are depicted in Fig. 5.3. The publishing rate is given in the average number of issued publications of each publisher per second. This is an important value to see how algorithms scale. We see that P-PUBS causes a higher loss rate which reduces delivery rates if publishing rates increase. TPSR has some kind of constant overhead, caused by distributing subscriptions, that does not depend on the publishing rate. For that reason, P-PUBS is slightly better at a rate of 0.1 packets per second. Although loss

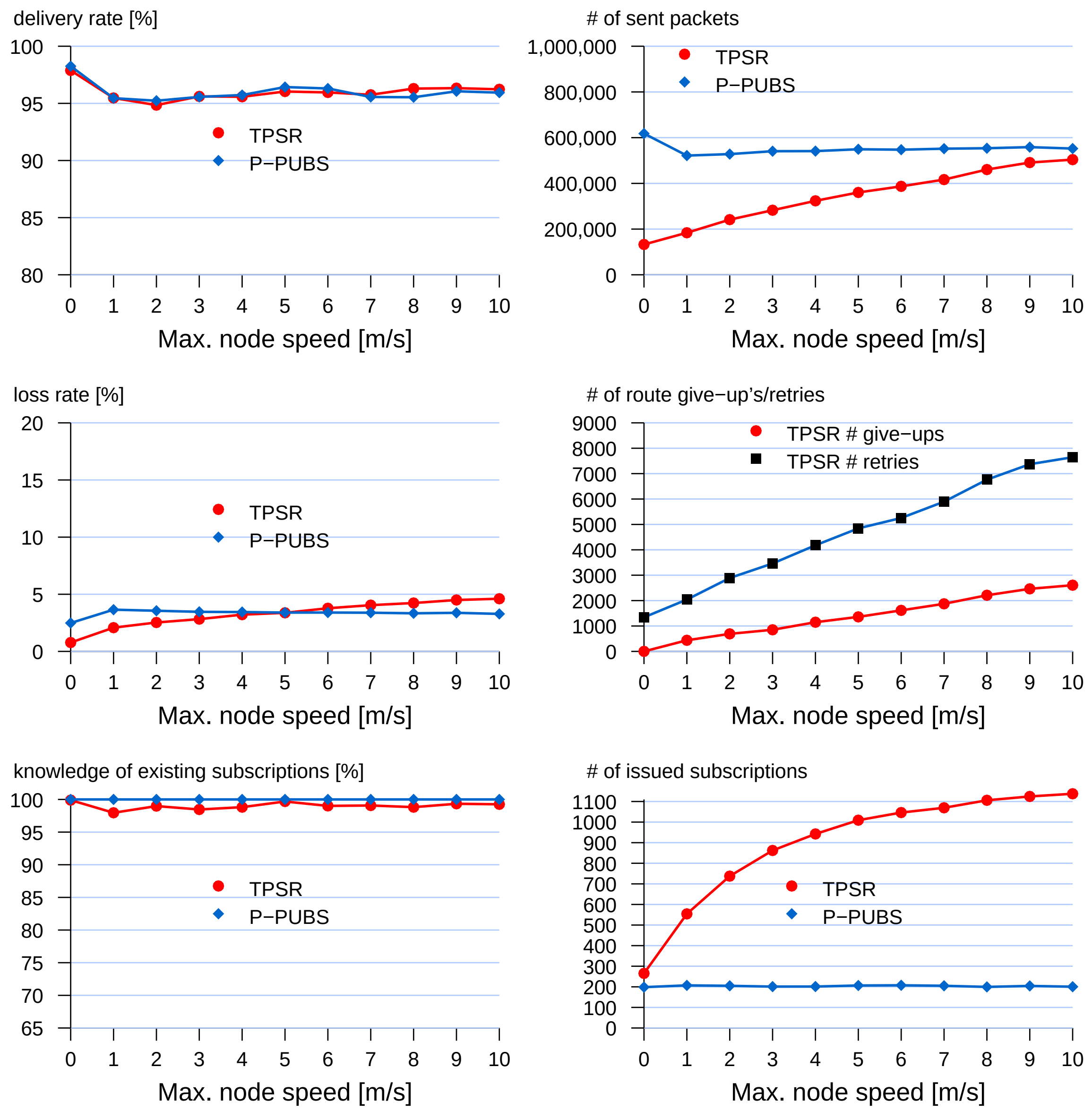

Figure 5.2: Comparison Pampa/TPSR with varied node speed

rates and the number of sent packets seem to look like linear functions, the growth seems to be greater than linear. When loss rate increases, we also send more packets to compensate that (even using P-PUBS), which again increases the loss rate. The number of sent subscriptions seems to converge to a certain upper boundary. At low publishing rates, broken routes are detected later, hence delaying some subscriptions and in total reducing their number. However it is also clear that no "hard" upper boundary for the number of sent subscriptions exists, because if the loss rate increases, the number of routes which are considered to be

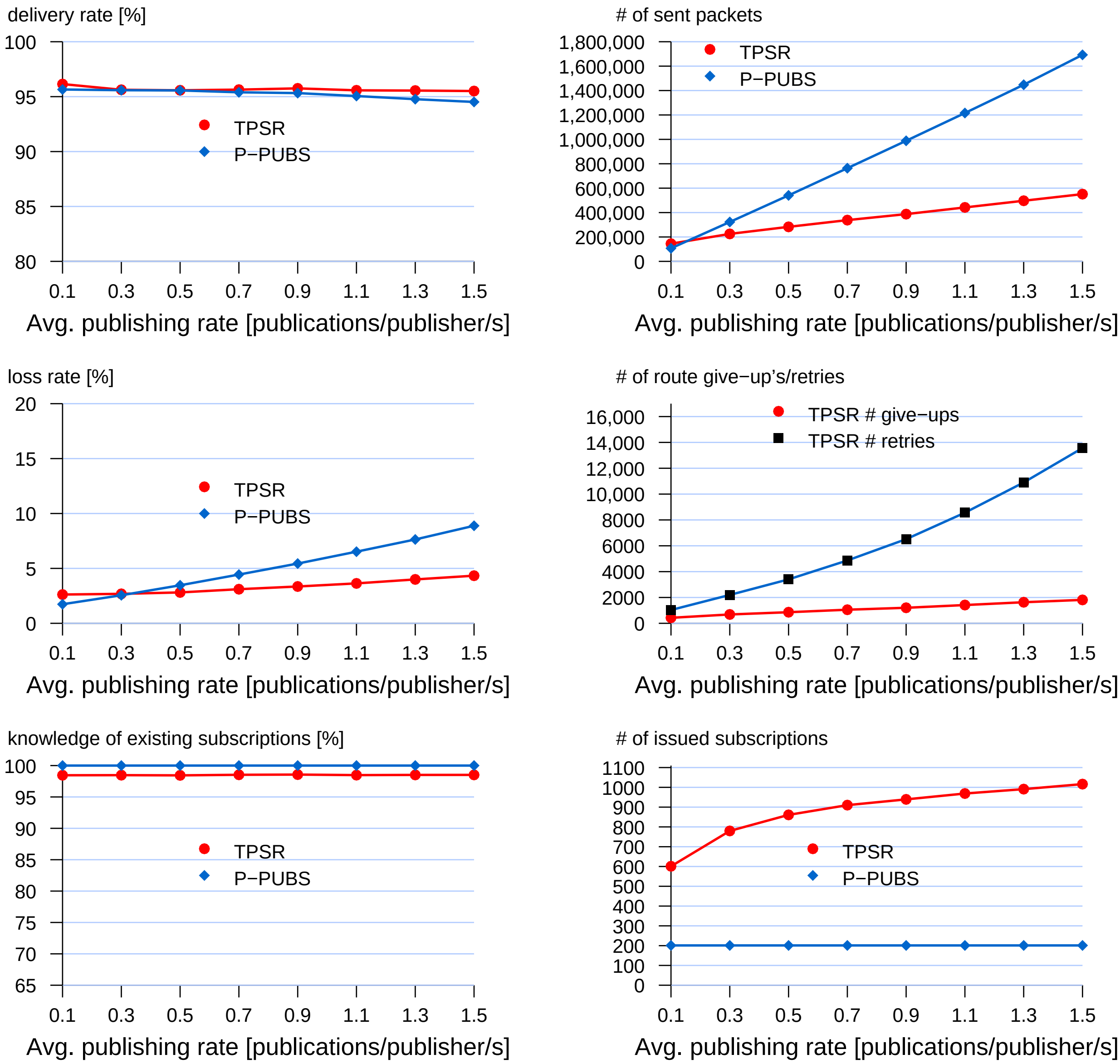

Figure 5.3: Comparison Pampa/TPSR with varied publishing rate

broken by mistake, also increases. Especially at very high publishing rates we would expect different results.

5.5.4 Varying RWP pause time

In Fig. 5.4 we see that the influence of the maximum pause time we use for the random waypoint model is relatively small. Values at 400 seconds do not look as smooth as the other

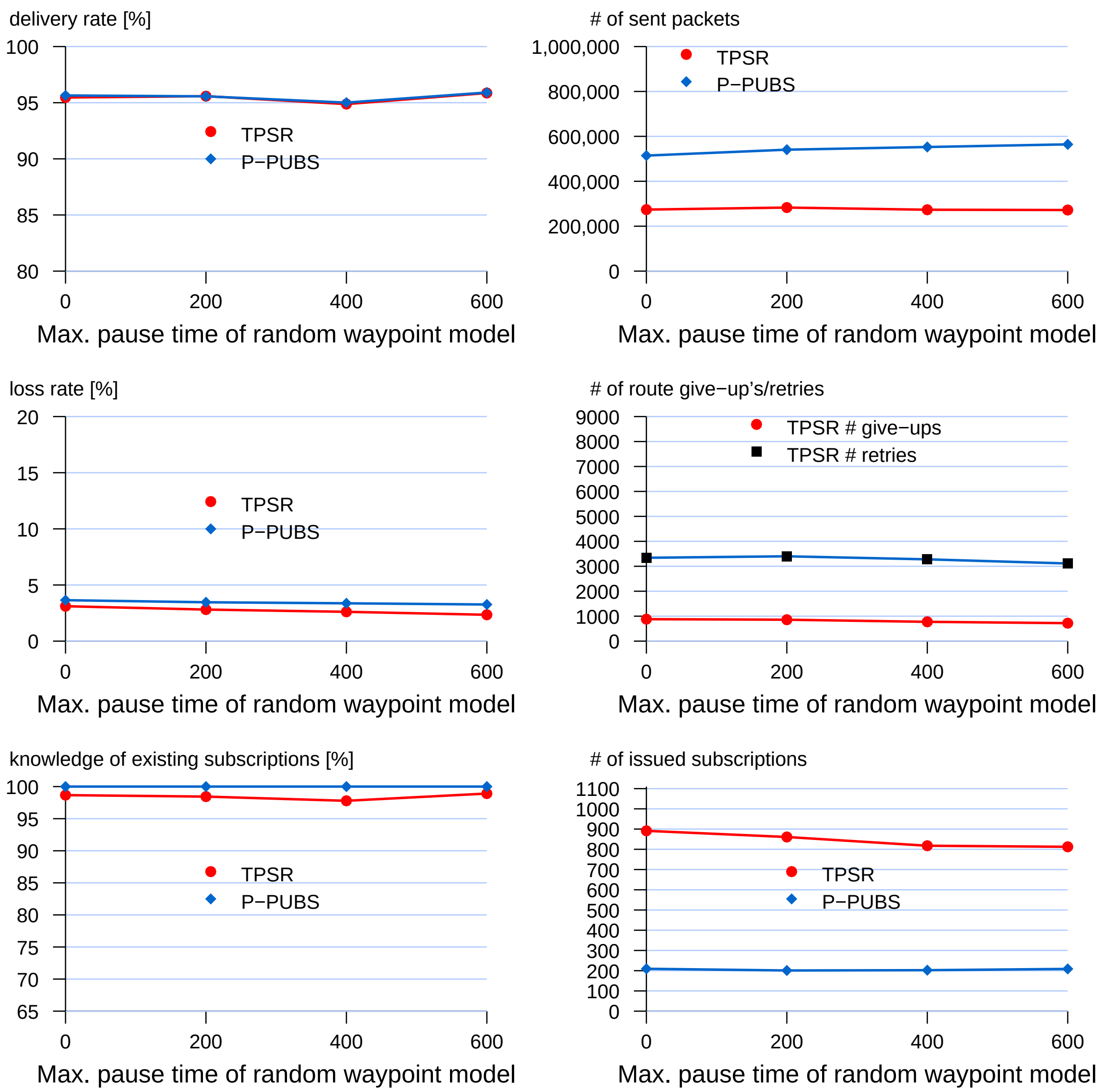

Figure 5.4: Comparison Pampa/TPSR with varied RWP pause time

ones, but after investigating the samples, we find, that that this is just caused by random. Once a network gets partitioned, it is more probable that this situation persists for some period, if the pause time increases. We simulate 10 samples for each value provided in the plot. For example at a pause time of 400 seconds the delivery rate ranges between 93.18 % and 97.48 %. This indicates that the drop of 0.5 % between 200 and 400 seconds is due to random. Looking at the number of sent packets, we see that P-PUBS marginally increases with increased pause time, while TPSR decreases. Although these deviances are small, we can explain this. Random Waypoint chooses the destinations, nodes move to, uniformly distributed. If nodes always move, they concentrate quite much in the central region of the simulation are. All nodes that at a certain time do not move (take a pause), are uniformly distributed over the area. If we increase the pause time, this also increases the number of nodes which are uniformly distributed, hence reducing the concentration of nodes in the center of the simulation area [BRS03]. As already explained in Chapter 5.5.2, this leads to higher traffic using P-PUBS and lower traffic using TPSR. In contrast to P-PUBS, TPSR benefits from stable routes. As nodes that do not move cannot destroy routes, higher pause times reduce the number of sent subscriptions, which is the reason for the small improvements at the number of sent packets.

5.5.5 Varying number of publishers

Figure 5.5 shows the results for different numbers of publishers. Using TPSR, even when there is no publisher present, we encounter a threshold we cannot go below. We send at least about 30,000 packets, which corresponds to about 200 subscriptions sent in total. This threshold applies to P-PUBS independant from number of publishers and also to PTSR without any publishers. This also means that in scenarios where subscribers are present, but no publishers, TPSR produces useless traffic where P-PUBS sends not a single packet. On the other hand, even with only 1 publisher, TPSR is expected to be better. Let us take a closer look at the number of sent packets. P-PUBS scales almost linearly, because if the number of publishers increases this also augments the number of publication disseminations accordingly. This also increases the number of collisions, which results in even more traffic. So actually P-PUBS is in fact worse than linear. TPSR needs to maintain the needed routes. This effort however is not linearly compared to the number of publishers. Newer routes can indirectly update older ones, no matter which subscriber initiated the subscription message. Due to this fact the number of issued subscriptions increases a lot with the first 10 publishers, but grows slowly between 10 and 20. Hence the same applies to the number of sent packets, which includes also subscription message. The number of retries grows faster than linear because not only the traffic increases, but also the collision rate. This effect also influences the number of sent packets to grow faster than linear (expected result having even more publishers). We see that TPSR does not scale better, but needs about one third the number of packets which P-PUBS needs, also having a better delivery rate.

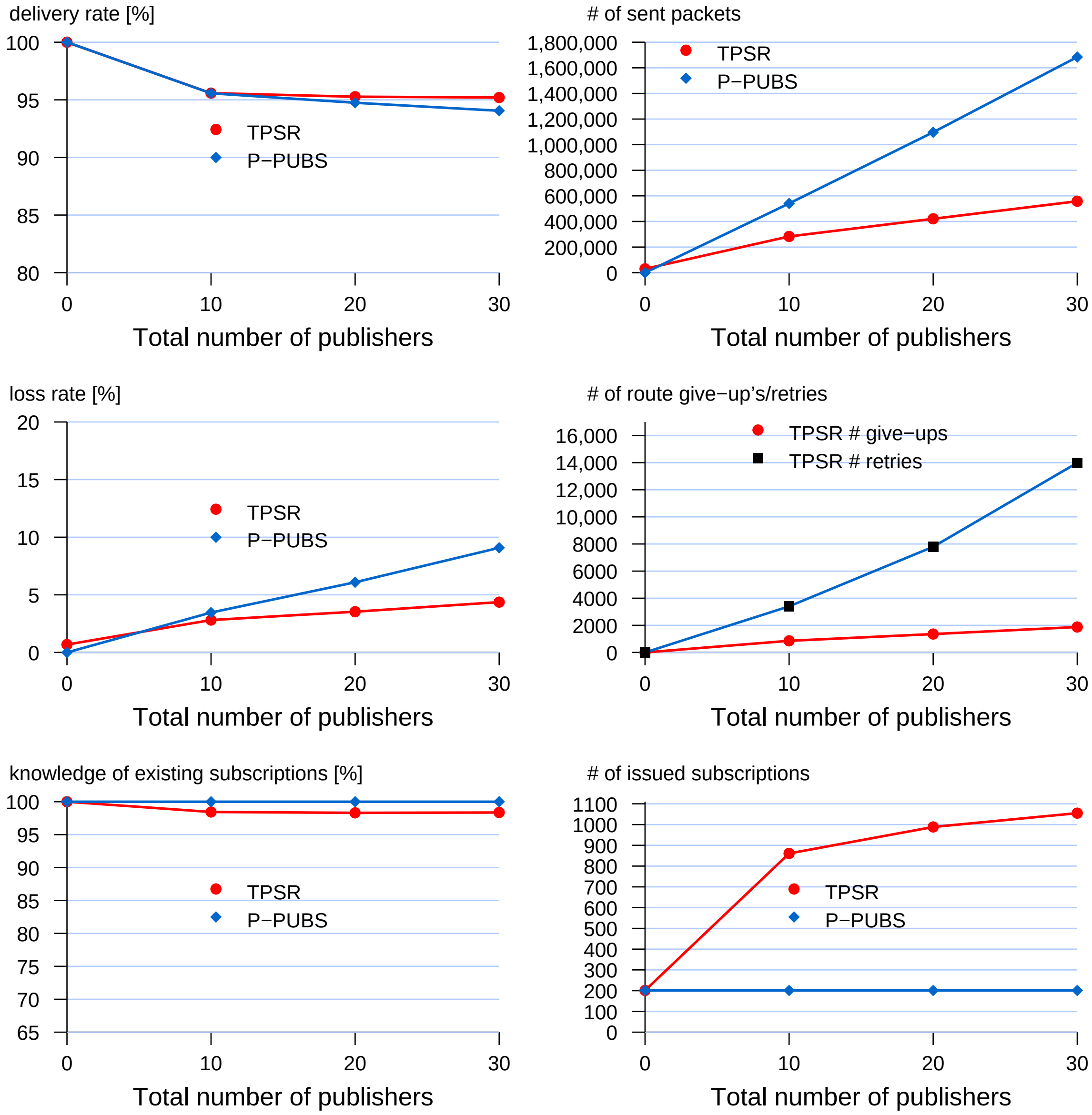

Figure 5.5: Comparison Pampa/TPSR with varied number of publishers

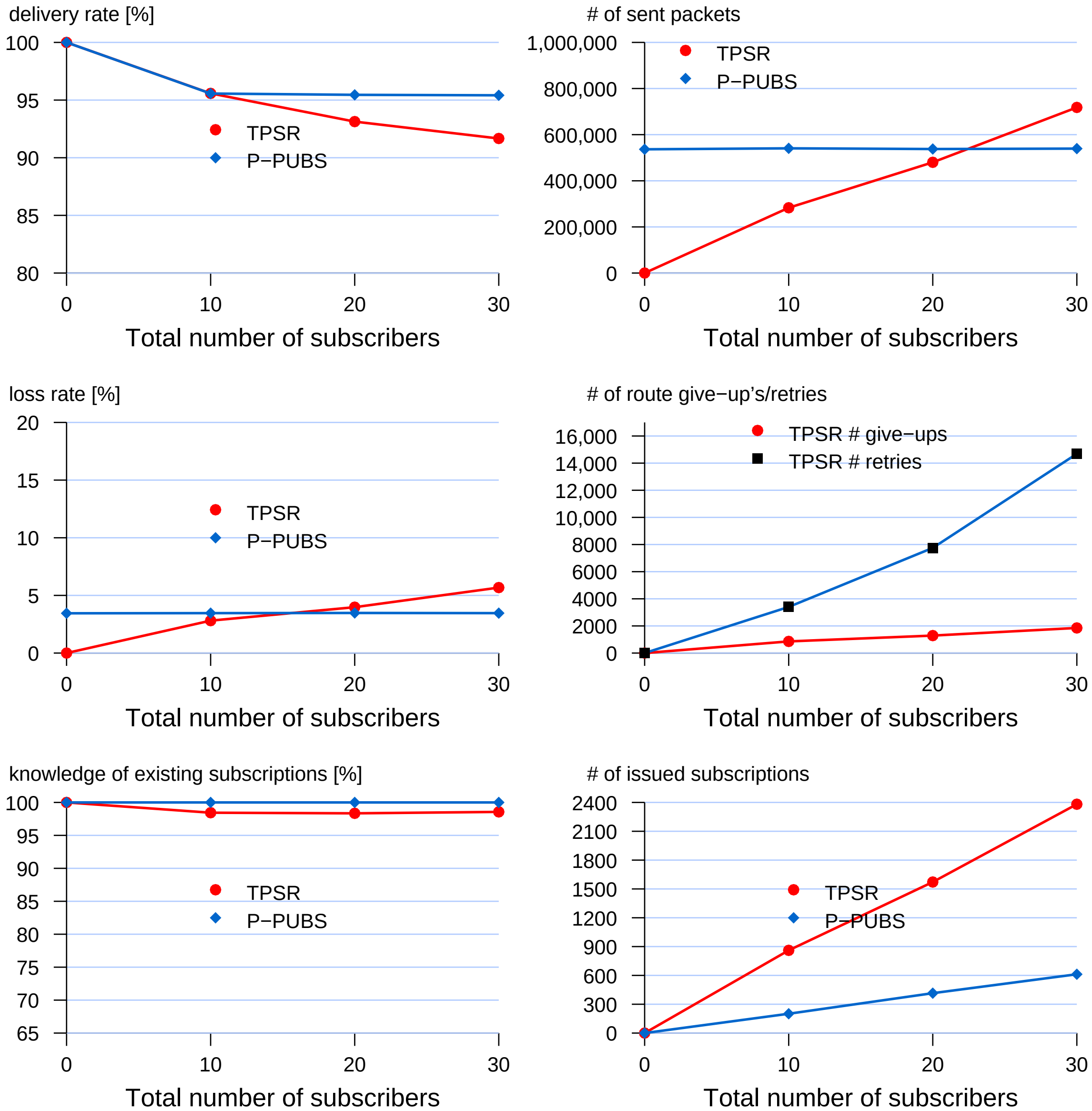

Figure 5.6: Comparison Pampa/TPSR with varied number of subscribers

5.5.6 Varying the number of subscribers

In Figure 5.6 we varied the number of subscribers between 0 and 30. In these plots we clearly see some limitations of TPSR. Using 20 subscribers we see that the delivery rate declines below that of P-PUBS. The number of sent packets is still a bit lower, so there is no obvious winner. Having more than 20 subscribers however, TPSR is clearly set aside. When having few or even no subscribers, TPSR is the protocol of choice. It scales down to no traffic at all if no subscribers are present. If TPSR is required to deliver a publication to several subscribers,

we would expect it to scale better than linearly, as the delivery is tree-based, which reduces the number of sent packets, compared to an approach based on unicasts. For example we would expect that sending a publication to 5 subscribers is (a little) more than half the effort of sending it to 10 subscribers. However this gain seems to get almost compensated by an increasing loss rate, which increases the number of retries.
Figure 5.6 leaves some open questions for future work:

1. At 20 subscribers the loss rate of TPSR is higher although the number of sent packets is lower compared to P-PUBS. We can argue that using TPSR traffic concentrates on the chosen routes, while other regions encounter less traffic. The traffic occurring with P-PUBS is more equally distributed concerning this matter. Anyway this tells us that there seems to be some space left to improve TPSR. This needs to get analyzed.
2. The savings of route aggregation seem to be not very big right now. This is clearly something which can be improved (see Chapter 6.2 for suggestions).
3. Although we can improve the protocol there will always exist some threshold, beyond which P-PUBS is better. We have to check if it makes sense to integrate P-PUBS into TPSR as fallback solution in case of many interested subscribers (which can be decided for each publication individually by the issuer). The size of the network can be estimated based on the sizes of the source routes, the network density additionally based on the route quality (in sparse networks routes tend to become bad, because its less likely to find nodes close to the target distance).

5.5.7 Varying the subscription recheck interval

The subscription recheck interval determines how often each node redecides which subscription he wants to use. The results are shown in Figure 5.7. This interval indirectly also influences how often subscriptions are changed. As subscribers may always decide to choose the same subscription they previously used, this not always happens. Nodes that are very close to a certain hotspots are more likely to choose his subscription and hence are less likely to change it later (assuming they are staying close to that hotspot). There we see the increased number of issued subscriptions at P-PUBS. As subscriptions are renewed at least each 90 seconds (45 % of 200 seconds), we see a threshold of about 133 subscriptions we cannot go below (1200 seconds simulation time divided by 90 seconds, times 10 subscribers). If more subscriptions are sent, these are due to the fact the subscribers chose a new subscription (choosing the same again would not issue a subscription). We see that the influence on TPSR is not heavy. At small intervals we see that the subscription knowledge is slightly lower, at the delivery rate we see the consequence, in particular that TPSR is marginally better at higher intervals. The number of sent packets is a little bit higher at higher recheck frequency, but we still stay significantly below P-PUBS. When subscribers issue subscriptions more frequently on their own initiative, this also reduces the number of resubscriptions caused by TPSR to recreate routes. We see this at the number of issued subscriptions, for example by comparing the values at 15 seconds and 30 seconds. TPSR declines about 150 subscriptions, while the value for P-PUBS declines by more than 200 subscriptions. So the increased number of renewed subscriptions also reduced the number of subscriptions necessary to maintain the

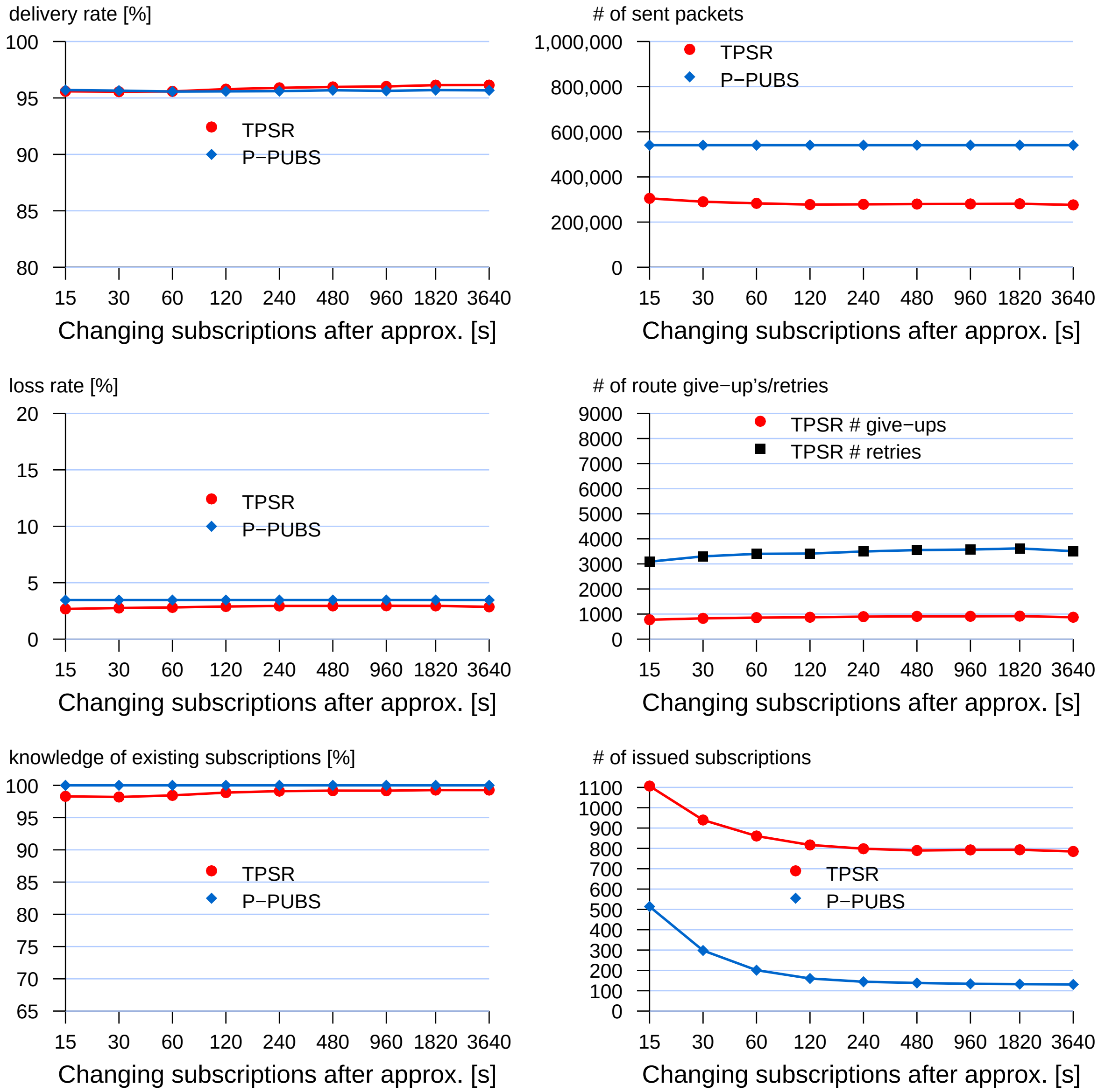

Figure 5.7: Comparison Pampa/TPSR with varied subscription recheck interval

source routes. For that reason the number of retries and give ups is also slightly increases with greater recheck intervals.

5.5.8 Influence of location-based subscribing

We assumed that an event routing protocol should benefit if the location of a subscriber influences the subscription he uses. The algorithm allows to specify the probability to choose

subscriptions based on subscribers locations or just by random, as already explained in Chapter 5.3.2. We investigated how big this benefit is, the results are shown in Figure 5.8. We can conclude that those benefits are negligible for the current version of TPSR. In particular we can give two reasons for that. First the current implementation of TPSR combines routes to multiple subscribers only in few cases. Second we have 10 subscribers choosing from 4 hotpots. In average each subscription is chosen by 2.5 subscribers. As this value increases if the number of subscribers increases, we would expect to see this benefit in Figure 5.6. However we have to consider that more subscribers also increase the loss rate and how often we have to rebuild the routing tree. Both issues increase the number of sent packets, so Figure 5.6 does not contribute to a better understanding of the influence of location-based subscribing.
As many subscribers are located somewhere in between all hotspots, the difference between choosing subscriptions based on location and based on random is not very significant. For example we could assume that the location of a node decides whether it is a subscriber or not. When it leaves a certain area it will just unsubscribe. In all of our simulations we assumed that the movement pattern of subscribers (and likewise that of publishers) is exactly the same as that of any other nodes. This seems to be an unrealistic assumption that also removes lots of possible benefit we could get. It is left to future work to optimize the hotspot model to be more realistic.

5.5.9 Benefit of proactive route recovery

It is not obvious whether proactive route recovery (explained in Chapter 4.3.2) just causes additional traffic or really provides an improvement. In Figure 5.9 we compared TPSR with and without proactive route recovery. While the delivery rate and the subscription knowledge are almost identical, other values show some differences. Using proactive route recovery we basically reduce the number of give ups (since we try to recreate the route before it breaks), but on the other hand we increase the number of sent subscriptions. This is obvious, because if we disable this feature, we only recreate routes that are supposed to be broken already. The proactive approach resubscribes often earlier which means that we do this more often. The benefit we get is not really big, but present. Currently we did not check how TPSR performs when we vary the route recovery limit. This is subject of future work.

5.5.10 Influence of Hotspot locations

By default hotspots are aligned on a grid (see Chapter 5.3.1 for details). However it is not obvious why this should be a good idea. To see whether our results depend on this evaluation detail, we also ran a set of simulations that places hotspots on random locations before the simulations starts. Figure 5.10 shows the results. As we can see there, it seems that the difference is close to not present. Hence we can conclude that aligning hotspots in a grid does not really influence our results. The main reason is, that the influence of the hotspot model we used is very low, we saw this already in Figure 5.8. So it does not really matter where the hotspots are located.

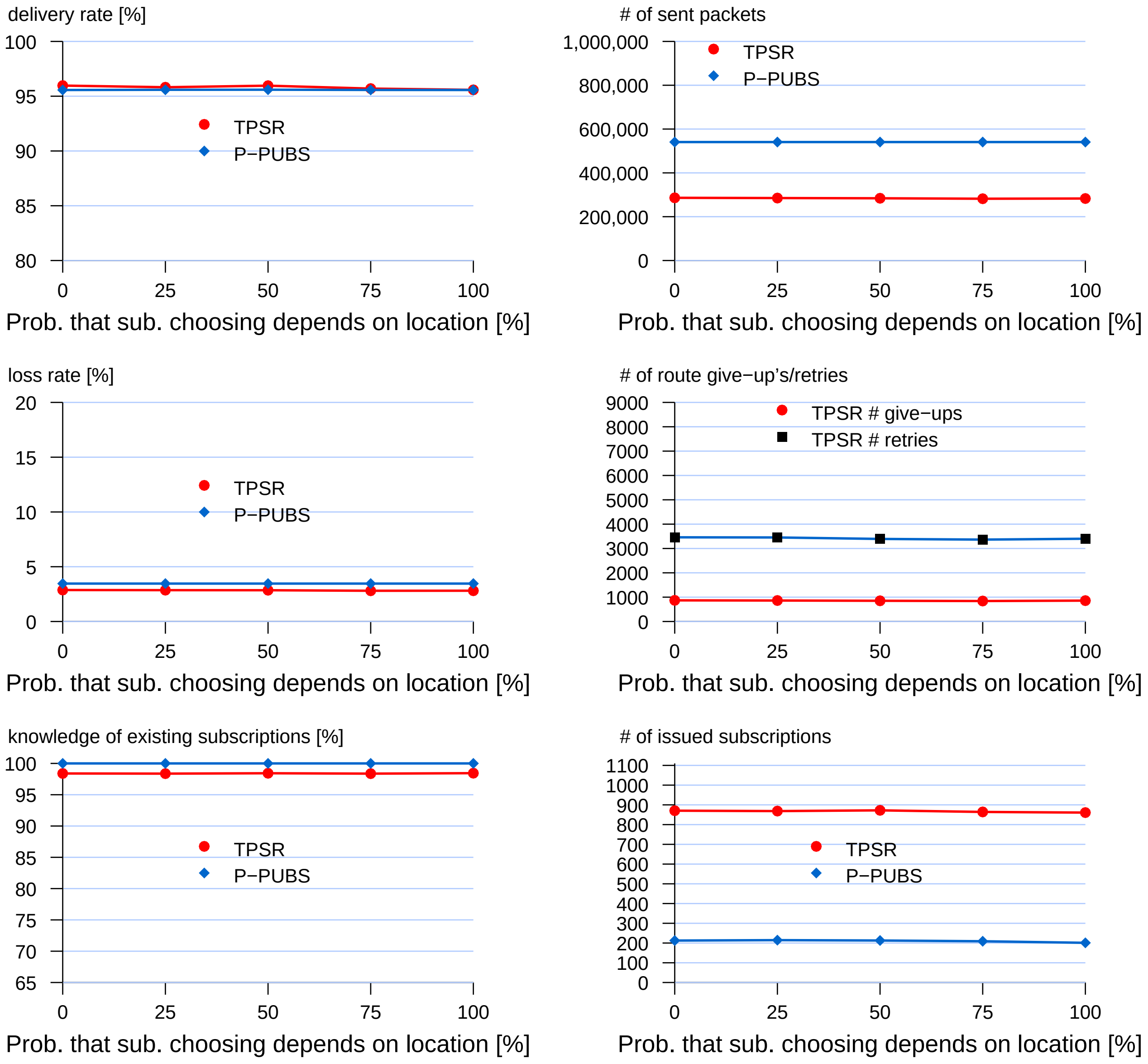

Figure 5.8: Comparison Pampa/TPSR with varied probability that choosing subscriptions depends on their distance to hotspots

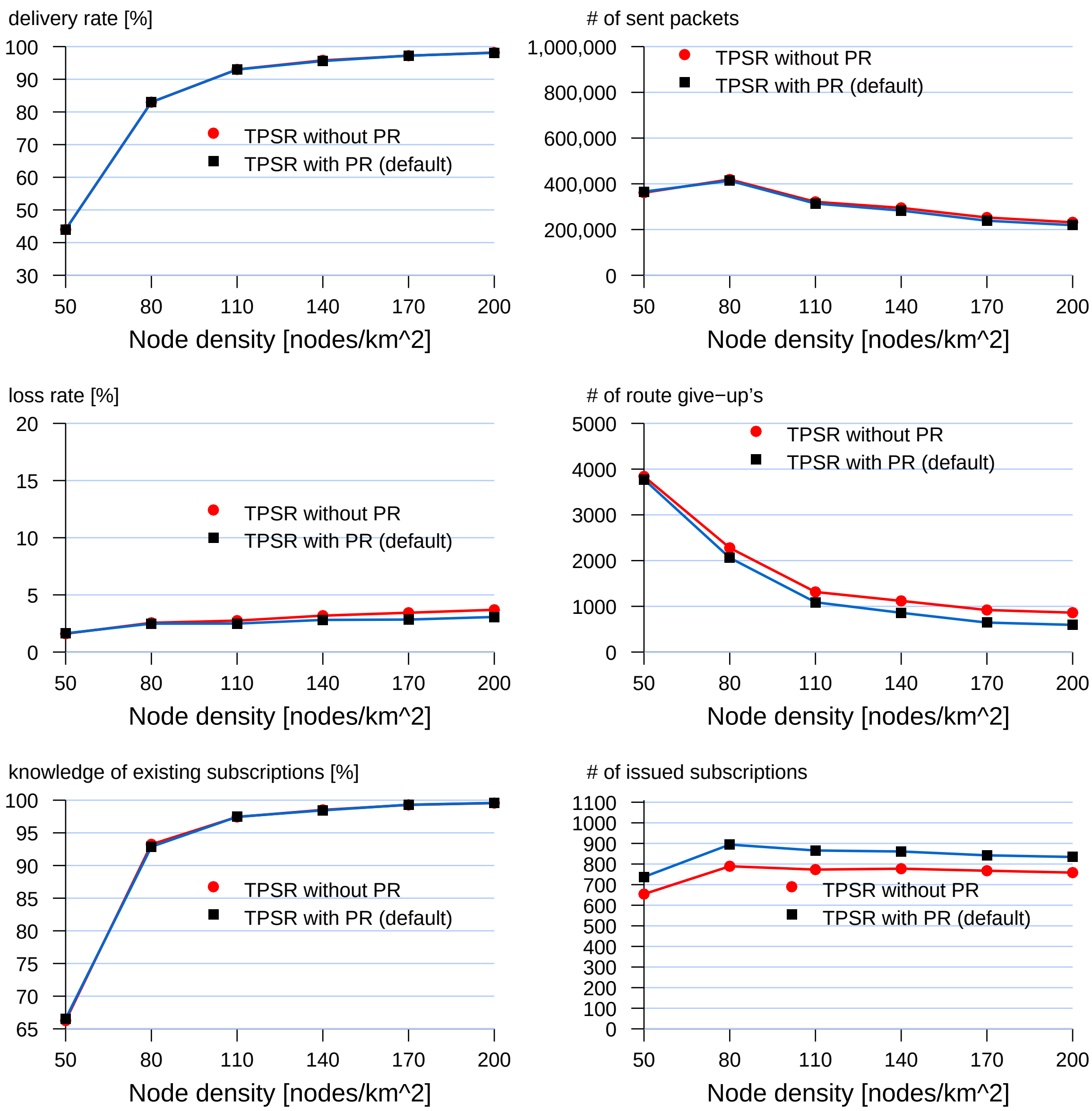

Figure 5.9: Comparison TPSR with and without proactive route recovery

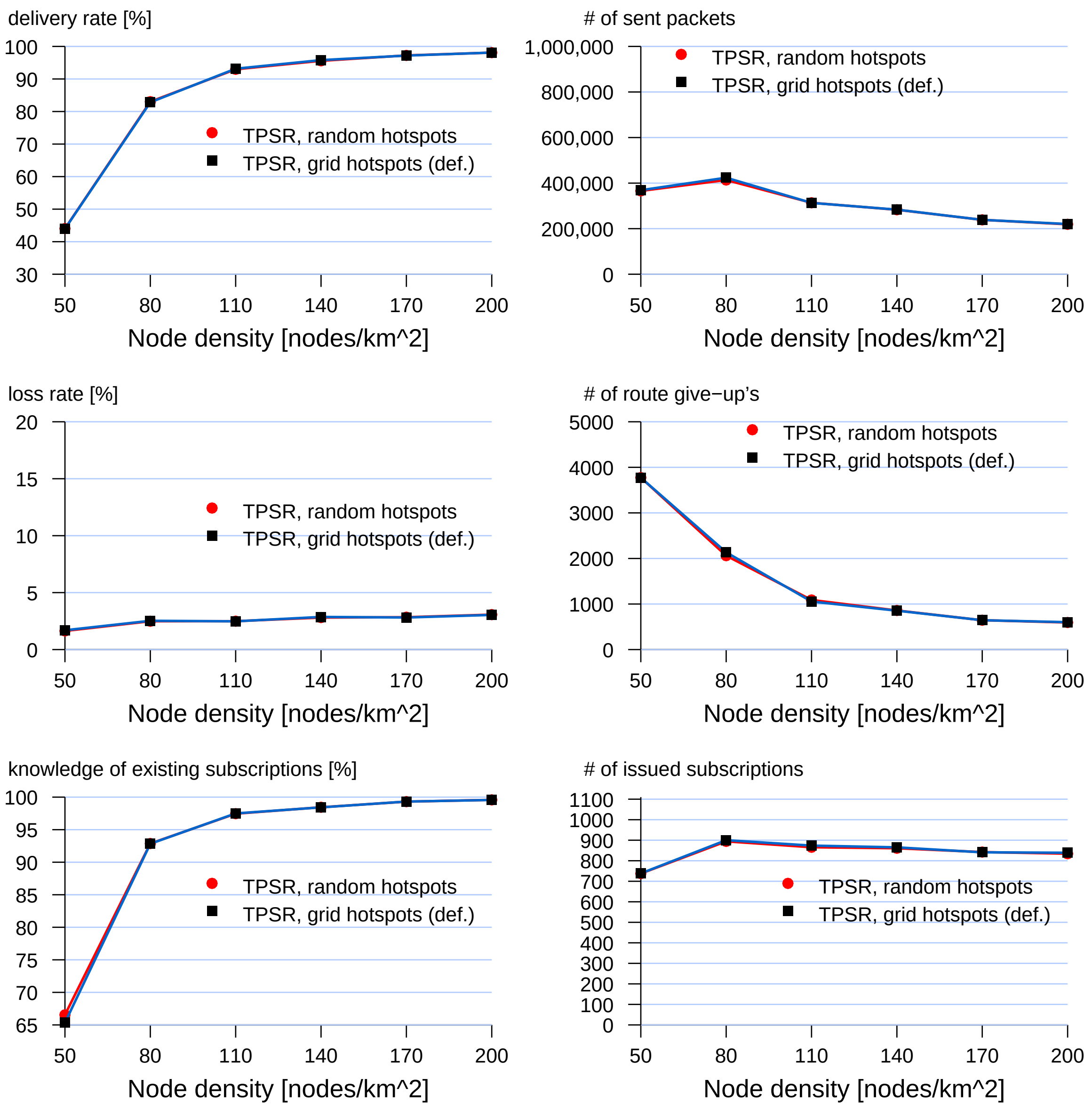

Figure 5.10: Comparison TPSR with randomly placed and grid-aligned hotspots

5.6 Other Movement Models

Beside Random Waypoint [JM96], which has well-known limitations [YLN03, BRS03], we also evaluated the Manhattan Grid Model [SB] and a Stuttgart City Model [can][3]. Both models simulate areas of a certain size. Changing the size of the areas would impact the nodes movement, for example by virtue of a different number of streets or (in case of the latter model) even different kinds of urban infrastructure. Hence, we decided to use the same street maps for all simulations. Because we vary the node density this of course requires also to vary the number of nodes. The steps of the node densities are the same as in the RWP simulations, ranging from 50 to 200 nodes per square kilometer. The mapping between node density, simulation area size and number of nodes is shown in Table 5.5, comparing Random Waypoint *RWP*, Manhattan Grid Model *MGM* and Stuttgart City Model *STGT*.

	Number of nodes			Area size [m^2]		
Node density	**RWP**	**MGM**	**STGT**	**RWP**	**MGM**	**STGT**
50	200	112	128	2000x2000	1500x1500	1600x1600
80	200	180	204	1581x1581	1500x1500	1600x1600
110	200	247	281	1348x1348	1500x1500	1600x1600
140	200	315	358	1195x1195	1500x1500	1600x1600
170	200	382	435	1084x1084	1500x1500	1600x1600
200	200	450	512	1000x1000	1500x1500	1600x1600

Table 5.5: Overview of scenario environments for RWP, MGM and STGT

5.6.1 Manhattan Grid Movement Model

The Manhattan Grid Model was proposed by [umt98]. The basic idea is that people usually move in an urban environment containing streets and crossing, without being able to move through buildings for example. A fixed number of horizontal and vertical streets are defined. Nodes always move along streets and at each crossing they decide where to move next. With 25 % probability they move left, with 25 % right, and with the remaining 50 % they keep their direction. In this simulation we used a simulation area of 1500 m by 1500 m, with streets at intervals of 250 meters. The speed is chosen between 0.5 and 3.0 m/s, pauses are chosen between 0 and 30 seconds. To make the simulation more realistic, additionally the speed is changed in average every 10 seconds. The scenarios were created using Bonnmotion [bon][4]. Figure 5.11 shows the results.

When interpreting these results, it is interesting to compare with the RWP simulation, shown in Figure 3.1. At low node density, MGM is worse than RWP. Nodes moving in different

[3] The citation only refers to the scenario generator (Canumobisim) we used. To Canumobisim we pass an AWML (Augmented World Modeling Language) file containing the streets of the city of Stuttgart. This AWML file was created by the Distributed Systems Department of the University of Stuttgart. However this file is not publicly available, yet.

[4] command line options used: "ManhattanGrid -i 36000 -d 18000 -n $numnodes -x 1500 -y 1500 -h 3 -p 200 -l 0.5"

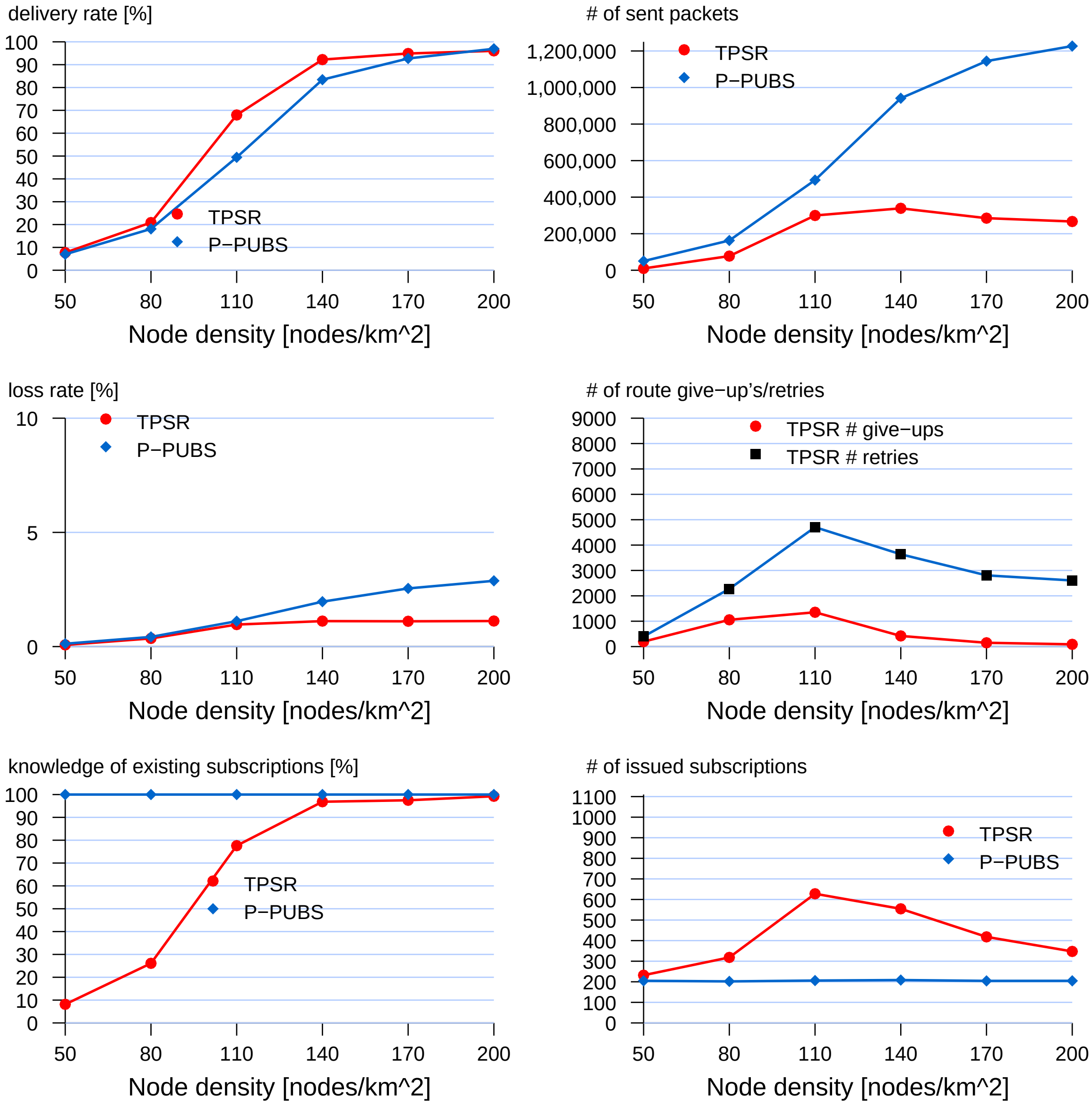

Figure 5.11: Comparison Pampa/TPSR with varied node density, Manhattan Grid Model

parallel streets are unable to communicate with each other[5]. This means that traffic has to follow the grid. In sparse networks, it happens often that there is no neighbor close enough. Using RWP helps bypassing areas of low density, with MGM this fails more often, since nodes only move on streets. If a few streets provide no link, dissemination just stops. Another advantage of RWP is the non-uniform node distribution. If most nodes concentrate in a certain area (i.e. in the center of the simulation region), it is likely that both, publisher and subscriber are located within this area of higher node concentration. In the MGM nodes are more evenly distributed compared to RWP, which also makes RWP perform better. Using MGM we need almost twice the density to achieve a similar delivery rate as RWP. At a density of 110 and 140 TPSR has a higher delivery rate, because it uses Pampa with 3 retransmits to find routes, instead of 2 retransmits, used for P-PUBS. A significant difference can be seen at the number of sent packets. At P-PUBS this grows a lot at densities higher than 110. We cannot explain this with the loss rate, which is actually lower than at RWP. The reason seems to be, that the network is not saturated, yet. Almost all nodes retransmit at P-PUBS, it seems to happen not very often that a node receives enough retransmits that he decides to not retransmit. At node densities higher than 200 we expect this growth to stop. If at every possible location in the area three copies are received, additional nodes will not have to retransmit, too – or if they do, other nodes will no longer. TPSR has this problem only to create source routes, which is few traffic compared to the traffic required to deliver publications. Additionally TPSR has another big advantage. As mentioned, all routes have to follow the streets. This makes it much more likely that routes that exist in a publishers route cache can be update by source routes from other subscribers (as explained in Chapter 4.1.4). This improves the existing routes a lot and hence reduces route failures. While at RWP the number of issued subscriptions stays almost constant while node density increases, using MGM almost no publication needs to get delivered by Pampa as explained in Chapter 4.3.2. For instance at a density of 200, the number of route repairs is also about 3 times higher using RWP. We can see that by comparing the delta between P-PUBS and TPSR at the number of issued subscriptions of the two plots. The MGM which is considered to be more realistic than RWP shows clear supremacy of TPSR over P-PUBS. It shows that TPSR scales very well with the number of participating nodes. Hence especially in urban areas with higher node densities we expect TPSR to provide good performance.

5.6.2 Stuttgart City Model

The Manhattan Grid Model tries to model an urban environment and hence might be more adequate than Random Waypoint. However the simulation is still very artificial. However even for Manhattan this might be somehow unrealistic, for example because of parks or streets that do not match the grid alignment. In most other cities of the world we find almost no grid alignment. To model a more realistic scenario we used a street map of the city of Stuttgart using an area of 1600 by 1600 meters. We used Canumobisim, a movement generator

[5]Actually it is theoretically possible, because the receive distance is 250 meters, which is the same as the distance of two parallel streets. However this is almost impossible to happen.

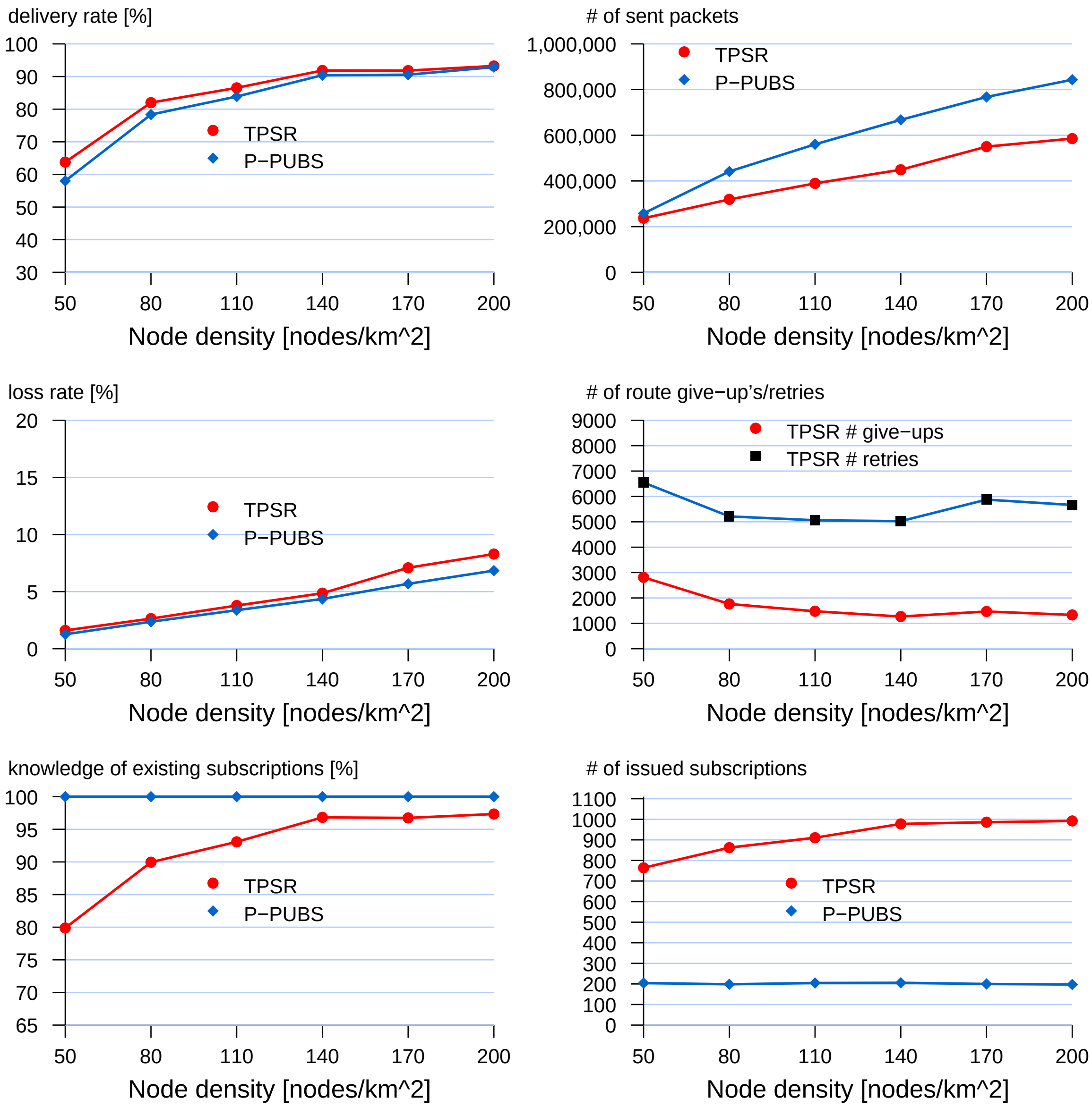

Figure 5.12: Comparison Pampa/TPSR with varied node density, Stuttgart City Model

developed at the University of Stuttgart [can][6], to create the scenario files. Compared to P-PUBS the delivery rate at a density of 80 is almost the same. At lower density Stuttgart City Model (STGT) is better than RWP, at higher densities RWP is better. At RWP the node distribution is much more homogeneous than at STGT. In particular RWP has a higher node concentration in the center [BRS03], but using STGT the differences between regions having a high node density and regions with low density are much more extreme. At a density of

[6]The code was patched to include a border of size 10 (instead of 0.0000001) which prevents Ns-2 from crashing. The parameters we used are printed in Appendix A.3

50, this high node concentrations causes a high delivery rate. On the other hand, if density increases, we get many collisions while many nodes are still unreachable, that we never see delivery rates of more than approximately 93 % – using RWP we got more than 98 %. The simulation area is printed in Appendix A.3.

Comparing node distributions

Using the program "nam" (part of Ns-2) we were able to see graphically how nodes move and how they concentrate on the simulation area. To get an intuition about node distribution, Figure 5.13 provides an example of node locations before the simulation starts (left picture) and after the simulation ran about 600 seconds (right side). Additionally Figure 5.14 provides examples for Manhattan Grid (left picture) and Random Waypoint (right picture) for reference. For RWP and MGM we cut off the first part of the scenario to avoid using a different node distribution on startup (see Chapter 5.5.2 for more detail). Hence the initial node distributions would look similar to mature ones. However at Stuttgart the first part has not been cut off (since Canumobisum does not support that, yet). That explains the need to provide two pictures for STGT.

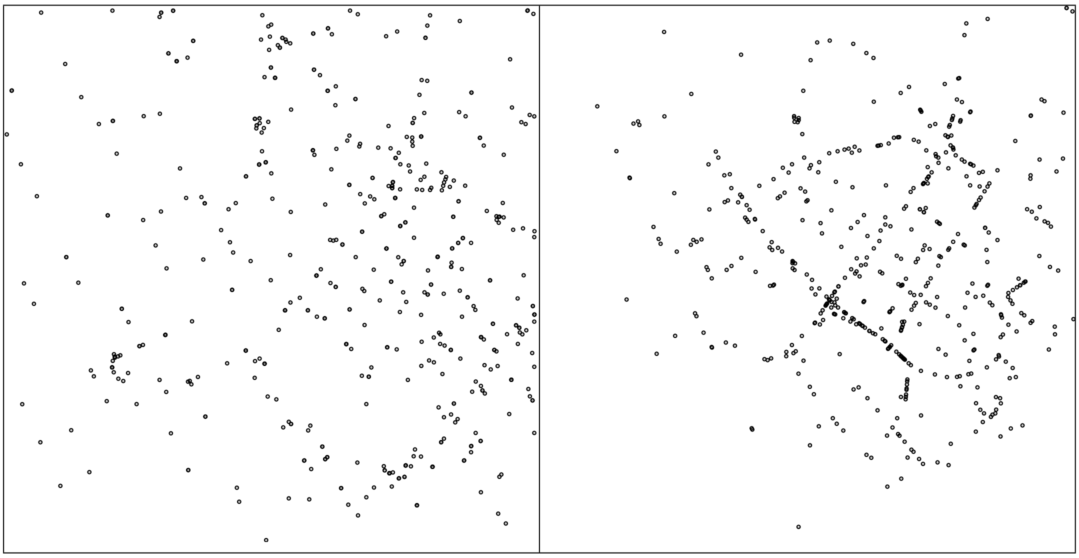

Figure 5.13: Stuttgart City Model, node location example (left is initial, right after about 600 seconds)

Especially on the big streets in the center of map nodes travel very frequently. This leads to high collision rates for both algorithms. Compared to MGM, TPSR gets worse, while P-PUBS gets better. However TPSR is still significantly better than P-PUBS. But it poses the question whether and how TPSR can be improved to handle such situations better.

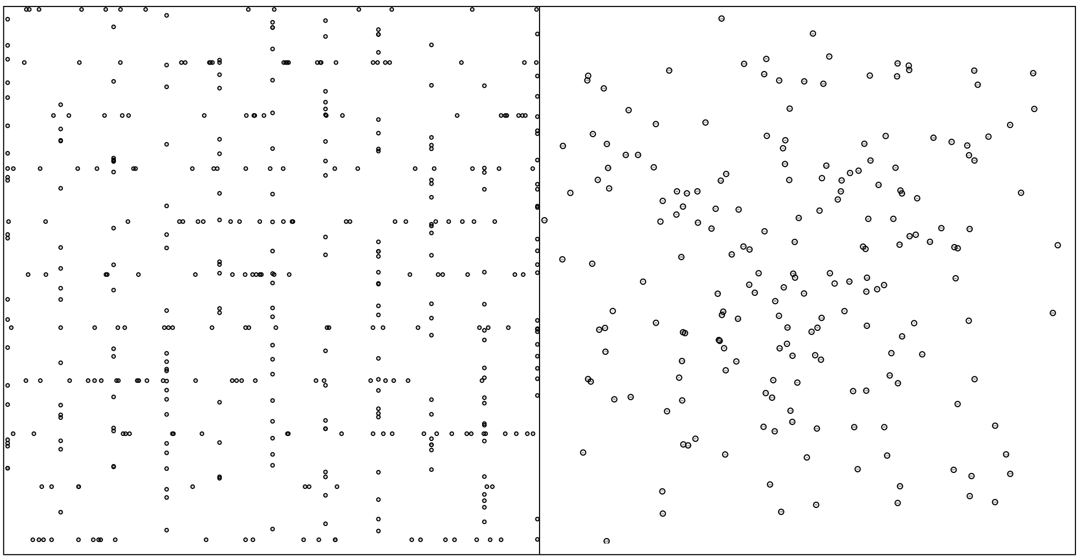

Figure 5.14: Node location example (left is MGM, right is RWP)

6 Suggested improvements for TPSR

Our simulations revealed strengths and weaknesses of TPSR. For example based on our scenarios we found out that if more than 10 % of the nodes are subscribers, TPSR becomes worse than P-PUBS. Investigating the different questions arising from the evaluation chapter, we think that although TPSR is a promising approach, there is also much which can be improved. For example we should try to combine routes to subscribers located in the same region. To make TPSR more scalable it is also a promising approach to avoid recreating the whole routing tree each time a single route got broken. Although AODV [PR02] and DSR [JM96, DBJB01] did not provide satisfying results in our simulations, they propose good improvements which might also improve TPSR. In this chapter we propose future work that was not tested yet, but deserves further investigation. In addition we show how these improvements are related to existing work of other authors.

6.1 Single route refresh

When publications get delivered by PAMPA (because the route probably got broken), this causes the recipients to resubscribe, which is a second flooding of the network. However if the other routes are still stable, there is no need to recreate all routes new. It would be sufficient to send an updated route just to the node that encountered the broken link. As this node issued the flooding, the subscriber received also a source route which can be used to send the reply back (similar as AODV does). If the subscriber keeps track of the quality of the routes used by publishers to send packets to him, he can make a reasonable decision whether to use a uni-cast route reply message or whether to recreate all routes instead. However we have to consider that creating stable routes with PAMPA requires more effort that just flooding the network, since PAMPA(3) requires more packets than PAMPA(2) and additionally using smaller hop distances is also expected to cause more packets to be sent (see also Chapter 4.1). According to preliminary simulations creating a source route tree takes about twice the effort of disseminating a message with PAMPA(2). Hence we would not use PAMPA(2) as fallback in case of broken routes any more, if we want to use the traversed path as source route, which was also explained in Chapter 4.1. This however increases the traffic needed to disseminate publications if a route fails. Optionally we could use PAMPA(2) to create source routes in this particular case, since the delivery rate is not as important as when issuing a subscription. Because if a subscriber does not receive a publication disseminated by PAMPA, this affects only a single publication.
In some cases we might encounter another issue. If more than one addressee is unable to deliver a publication, it may happen that a subscriber only receives the publication via PAMPA

from addressee A, but not from addressee B. For example because B already received the dissemination and suppressed his own dissemination. The consequence is that all subscribers that did not receive the publication using routing, only know a path to addressee A. However this path can be much longer than a path created if the subscriber would just resubscribe. It is left to further investigation how this issue influences efficiency. However we expect this influence to be low, since it probably happens rarely that more than one route is assumed to be broken at a certain publication. Additionally even if we would use longer routes, this situation exists only temporarily until the route tree is completely rebuilt (e.g. if the subscription times out).

6.2 Improved route merging

Especially for GEO, route aggregation is a good possibility to optimize routing. The current version only combines that parts of the routes, where they use the same nodes, which is determined by the nodes' locations which is almost based on random. Additionally, if an updated route has (by chance) passed nodes of an existing route, this part is also replaced by the new one. For further improvement, we suggest to attach a distance vector to each source route, which is also cached in each node's routing table. If a node wants to create route jobs to a set of subscribers, it can also look at the distance vectors. Perhaps it is more efficient to use an existing route more than once and assign its route job also to nodes which usually would have gone another route. If the number of hops plus the number of additional hops (read from the distance vector) is smaller than the sum of hops of both source routes, we found a better route. This optimization can also be used to combine more than 2 routes. Although the message size will increase due to the added distance vector, we expect this optimization to provide some benefit (primarily in sending fewer packets).

6.3 Local route repair

Instead of recreating all routes (or a single one) new, we could also implement route repair functionality. A node that is not part of the delivery tree, lets call him Z, might also hear the message containing the route job from a node X and also the message from the next node (Y), the route job is assigned to. If that happens, Z knows the distance to both, X and Y, and (because this is part of the source route submitted by Y) the signal strength at which Y received the message from X. Having this knowledge Z may conclude that the existing route is not that stable any more and it would be a good idea if he would act as an intermediate node between X and Y. Then Z can issue a route optimization offer to X, which can update his routing table accordingly. In a different situation it might happen that a node A who sends a route job to B, which forwards it to C, hears C forwarding the message (e.g. to D) at reasonable power. Node A hence can decide to update its routing table and skip node B from the next message on. A similar approach is used by DSR [JM96], however without making use of the receive power and hence needing more traffic. Both optimizations need to be more sensitive than proactive route recovery at the subscriber, so that using these local repair features really avoids the restoration of the whole routing tree. Similar ideas where also published in [GM08, JM96, LMP], however they do not rely on the signal strength.

6.4 Improved route tree creation

In Chapter 4.1.1 we proposed a formula to calculate the delay that applies to the forwarding of subscriptions. Our simulations suggested this formula to be adequate. However it contains 5 parameters we did not exactly investigate. So it is left to further research to analyze the influence of each of them, aiming to find a better solution than the current one. It may also be an option to use a function similar to the one proposed for PAMPA2 [MM]. Doing this we can decrease the average delay and speed up dissemination, but it would be much more likely that nodes choose a similar delay and cause a collision (see Chapter 4.1.1 for reasons). The probability of collisions increases with the average node density. We only simulated node densities up to 200 nodes per km^2, using higher node densities is expected to increase the number of collisions even more, making TPSR perhaps a better option for sparse networks.

6.5 Keep multiple source routes

Instead of keeping only the best received source route for each destination, TPSR could keep all routes instead. This offers some advantages which could result in improved efficiency. This is a well-known approach, used by a set of routing protocols like DSR [JM96], AOMDV [PR02] or the protocol proposed in [CE95]. If a publication delivery fails permanently we currently would use PAMPA to get a publication still delivered and also indirectly inform the subscriber that the route got broken. If an alternate route still works, we could avoid flooding and therefore reduce traffic. The probability that an alternate route still works, after the primary one failed, determines whether the number of sent packets is actually reduced or even increased. If that probability is very low, we send packets, trying alternate routes, only to find out that they do not work either. If we finally have to use PAMPA anyway, we would actually send more packets than the current version of TPSR that uses only one route for each destination. We need to achieve a high probability, to benefit from this improvement. Selecting and creating those routes is addressed in several publications, for instance [LG01], [KS93] or [AT04].

6.6 Support of asymmetric connection links

The current version of TPSR depends on symmetric connection links. If node A has received a source route from node B, then it is expected that node A is able to deliver publications using the link to B. However if node B has a higher sending power, a better antenna or more sensitive radio hardware, this assumption is no longer true. In networks composed of heterogeneous nodes the current algorithm can become worse than P-PUBS. TPSR might never be able to deliver a single packet using source routes. Hence all publications get delivered by PAMPA. Since TPSR always tries to use its routes, we produce much overhead, which makes it worse than P-PUBS. TPSR tries to avoid links close to the receiving threshold, aiming for a target distance (see Chapter 4.1.1). However this formula, based only on the receiving power, can make things even worse. For example if a node that has only quarter sending power (sufficient for 125 meters), receives a route from a node that can send up to 250 meters, the algorithm prefers links at 180 meters distance which is clearly out of the range of the "weak" node. Therefore the presence of such nodes would destroy the benefit of TPSR.

To enable TPSR for such situations we would have to apply some changes. We suggest that every node appends its send power (including antenna gain) to all messages. Receiving nodes are then able to calculate a reasonable estimate of the distance to the sender, and – knowing their own sending power – they are also able to estimate up to what distance they are able to send packets. This is important, because a route which is created starting at the subscriber going to each publisher, is intended to be used in reverse direction later. We propose that the target power (see algorithm from Chapter 4.1.1) used in the delay formula depends on the smaller of the 2 send power values. This ensures that we create a route that can be traversed in both directions. If node B forwards a publication message from node A, this is interpreted as acknowledgements by node A. It would not be sufficient if node B can hear node A, it is also required that node A can hear the retransmission of node A. Otherwise it would always assume that the route is broken and use PAMPA to deliver the publication. Using different kinds of nodes may also produce longer routes, since using the smaller of the two send powers tends eventually aims for smaller target distances. To reduce the sizes of routes it makes sense to not just scale the delay function, but to increase the minimum delay if the maximum receiving distance is lower than the default. Doing so we can ensure that routes using nodes with a higher transmission range are preferred. Explaining graphically we would just cut off an intermediate part the current delay formula.

A more difficult problem to solve is, to get on with strange connectivity issues that occur in real life. For example entering or leaving buildings may change the connectivity very rapidly. Although it does not really matter whether the estimated distance is accurate or not (in other words: whether low receiving power is a result of the distance or a result of the environment), it does matter if changes occur quickly. This may result in frequent route tree rebuilds and therefore increased traffic. Unfortunately it is challenging to find reasonable simulations. Perhaps it might be even cheaper to test such scenarios in real life. To evaluate this is left for future work.

6.7 Update delay for route creation

If a subscription is sent, we use PAMPA to disseminate the message to all nodes. The delay, which is used to get stable routes, is currently calculated based on the delay and the number of hops of the first message a node received. Although this seems to produce good results, it can probably be improved. If a node receives a message which is close to its maximum receiving distance, it is put in PAMPA's send queue with a very high delay assigned. However soon after that it may receive a second packet containing the same subscription on a much better route. This better route would in the current implementation already be used to update the existing one. However the high delay is not being updated, at present. It seems to be an improvement that has no disadvantages at all, to reschedule the existing entry in the send queue if the delay that would apply to the second message would time out earlier than the entry currently present in the routing table. This way we expect slightly improved source routes.

6.8 Intermediate nodes route updates

The current algorithm is based on the knowledge of the publisher to determine which subscribers are interested. If this knowledge is not up to date, which happens if subscription updates got lost, it might happen that an intermediate node has better knowledge. One optimization is to append a list of all subscribers to each message containing route jobs. To avoid too much traffic, only the first addressee (having delay index 0) compares the subscriber list with what he thinks are interested subscribers. If it then detects that the previous nodes forgot subscribers, it will add them to the list of recipients he is responsible for and also to the list of all recipients (to avoid later nodes from doing the same). Additionally it attaches source route update information to the message it forwards. This update contains source routes and subscriptions of the added subscribers. This propagates the update backwards to the publisher and allows him to update its routing table. Each time this route is used, the knowledge moves one more step back towards the publisher, until it reaches it. Because most nodes have the most up to date knowledge of subscriptions, it should take only very few publications sent until the publisher receives the most up to date version. Although only addressees with delay index 0 are allowed to do that, it may happen that several nodes, located on different route paths, deliver to the same subscriber. However it is not expected to happen often and that situation is not permanent, due to the backward propagation. Optionally we could prevent multiple nodes doing the same by adding a note to the packet header telling whether on this route all route job delays were equal to 00.
We made some preliminary simulations to investigate how much benefit we can expect from this approach. As a result, we can say that the influence on the delivery rate is almost negligible (usually between 0.1 and 0.3%). However the traffic increases noticeable, because if any intermediate node has outdated subscription knowledge, we probably deliver to subscribers, who are no longer interested. So the problem is that it is much more likely that one of the intermediate nodes has outdated knowledge than that the publisher has. To resolve this problem, we could attach the subscription id of all subscribers known to the publisher (no matter whether they are relevant for the publication to send, or not). Doing so, every intermediate node is able to check whose knowledge was more up to date and we can avoid adding irrelevant subscribers to route jobs. Of course this increases also the packet sizes. It should be sufficient to use only 4 Bits for each subscription id, which gives us 16 values, representing the real subscription id modulo 16. Values between 1 and 7 greater are considered to be newer, values between 1 and 7 smaller are considered to be older. This system assumes that it is very unlikely that a single node does not receive 8 or more consecutive subscriptions from one subscriber, which seems to be reasonable.

6.9 Improve delivery rate of route creation

It may happen that a node does not receive a subscription. Preliminary simulations have shown that in the majority of these cases many other nodes also did not receive the subscription. The reason is that either the network was partitioned or (more likely) that the dissemination ended very soon. For example the initial message, sent by the subscriber was dropped because of a collision. To avoid this we suggest to add some startup-mechanism to PAMPA. For example the initiator could send each message 3 times instead of once (with a short random delay) and all direct neighbors could send it twice. A similar problem is addressed by [HHL02]. This requires also modifications to the counter of retransmits, so that messages sent by the same node multiple times, are counted only once. This improvement should also benefit from the combination with the previous suggestion ("Intermediate nodes route updates"), because then we can probably reduce the number of retransmits from 3 to 2, reducing traffic. Also the subscription timeout can probably be reduced. The "Resubscribe after" value (see Table 5.2) can probably also be increased, the main reason of setting it below 50% was to avoid longer periods a publisher has no valid subscription any more, just because he did not receive one. An interesting situation concerning this optimization happens when publishers do not know a single interested subscriber and therefore would not send any packet at all. If we want intermediate nodes to participate in providing subscription knowledge, we could also introduce route jobs with zero recipients choosing just one of our known neighbors randomly as addressee. If this node does not know more, he just sends an empty acknowledge back, otherwise it can forward the packet. That way we can make usage of the knowledge of other nodes, even if no known interested subscriber exists.

7 Summary/Outlook

To provide Publish/Subscribe functionality in Mobile Ad Hoc Networks is challenging. However this combination offers precious contribution to support different kinds of applications. Existing Pub/Sub systems usually require wired networks. Solutions that work for mobile networks are often assumed to obey certain constraints, especially concerning node movement patterns. Algorithms tailored for special environments usually do not work for some scenarios or provide non-satisfying functionality. Additionally measures assumed to be essential like the number of sent packets are sometimes not addressed by evaluations of MANET protocols. According to our simulations an optimized broadcast algorithm (Pampa) sent less packets to deliver messages to all participants of the MANET, whereas AODV and DSR sent more packets to deliver a message only to a single destination. Comparing broadcast protocols with unicast protocols seems to be an uncommon but not unreasonable approach. As a result we decided to develop a new approach, called *TPSR*. TPSR is not built on top of existing routing functionality, but tailored to the data distribution characteristics of Pub/Sub. An important characteristic of Pub/Sub is the fact, that disseminating subscriptions to all nodes already can be used to create source routes. Hence after having received a subscription message, each node is able to reach the subscriber, just by following the known path. The contributions of this approach are two-fold: We distribute subscriptions and at the same time create routes which can be used to deliver publications. A second innovative idea is to use the signal strength messages are received with, to optimize routes in terms of finding a good compromise between long routes and fragile routes to support content-based routing in MANETs. This helps that routes exist for a longer time. Since existing routes can only be destroyed because of node movement, establishing a buffer zone at each link effectively helps that routes persist for a longer period of time.

Our simulations have shown that TPSR presents a reasonable efficiency which superseded our expectations. But the simulations also revealed some limitations. Additionally we were able to see plenty of possible improvements that are expected to make TPSR use less traffic. An important question, which was not addressed by this thesis, is how good the estimation of the distance based on the transmission power is. Performing computer based simulations to evaluate this, seems to be impossible using todays hardware equipment. Therefore this should be evaluated using real devices in realistic scenarios. In real life signal strength can vary much between two consecutive messages. Since wave reflections may occur or buildings function like shields, small movements can mean a big impact on the transmission range and the signal strength.

This work provides a good approach to support Pub/Sub in MANETs. Publish/Subscribe is not only useful for wired networks, but also for Mobile Ad Hoc Networks. We were able to show that it is feasible to implement an efficient algorithm which even offers lots of possible optimizations mainly addressing efficiency. We hope this algorithm contributes to research and will be subject to further investigation building on and extending TPSR.

A Appendix

A.1 Simulation file (TCL)

```
# Stephan Schnitzer 2009

Class TestSuite
Class Test/pubsub -superclass TestSuite

proc usage {} {
  global argv
  puts stderr "usage: ns <tcl-file> -alg <alg-name> -sc <movement-file>..."
  puts stderr "<alg-name> must be one of those: pampa-pubs, pampa-ucr, pampa-tree"
  puts stderr "<movement-file> should be: scen_<num-nodes>_<node-density>_<node-speed>_<movement-model>_<pause-time>.<idx>"
  puts stderr "Parameters were: $argv"
  exit 1
}

proc default_options {} {
  global opt
  set opt(chan)       Channel/WirelessChannel
  set opt(prop)       Propagation/TwoRayGround
  #set opt(prop)       Propagation/FreeSpace
  set opt(netif)      Phy/WirelessPhy
  set opt(mac)        Mac/802_11
  set opt(ll)         LL
  set opt(ant)        Antenna/OmniAntenna
  set opt(density)    0      ;# How many nodes per square km
  set opt(ifqtype)    Queue/DropTail/PriQueue
  set opt(ifqlen)     50     ;# max packet in ifq
  set opt(out)        "-"    ;# trace file
  set opt(lm)         "off"  ;# log movement
  set opt(nn)         200    ;# number of nodes used
  set opt(x)          2500   ;# Max. X dimension of the topography
  set opt(y)          2500   ;# Max. Y dimension of the topography
  set opt(maxborders) 20     ;# Max. borders, sufficient for a border of 10 on each edge
  set opt(alg)        "pampa-pubs"
  set opt(usenam)     1      ;# Also create a NAM logfile?
  set opt(rp)         NOAH
  set opt(numsub)     10
  set opt(numpub)     10
  set opt(stop)       1200.0
  set opt(range)      250    ;# default RX range in meters
```

```
  set opt(targetdist) 180
  set opt(matching-subs) 100.0
  set opt(sub-locdep) 100.0
  set opt(hrl)       0   ;# enable random hotspot locations instead of grid
  set opt(pr)        1   ;# enable proactive route recovery
  set opt(ir)        0   ;# enable intermediate route updates?
  set opt(ss)        0   ;# enable static subscriptions?
  set opt(rrl)       15.0 ;# route recovery limit in percent
  set opt(sub-coverage) 0.0
  set opt(pub-rate) 0.5
  set opt(rj-max-retries) 2
  set opt(numhotspots) 4
  set opt(infotext) " " ;# additional text printed in the result summary
  set opt(subscription-checkinterval) 60.0
  set opt(subscription-timeout) 200.0
}

proc RXT {distance} {
  set v [expr 0.01386645232 / $distance]
  return [expr $v * $v]
}

Mac/802_11 set dataRate_ 11Mb
LL set mindelay_ 50us
LL set delay_    25us
LL set bandwidth_ 0 ;# not used
Agent/Null set sport_ 0
Agent/Null set dport_ 0
Agent/CBR set sport_ 0
Agent/CBR set dport_ 0
Queue/DropTail/PriQueue set Prefer_Routing_Protocols 1
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set Rb_ 2*1e6
Phy/WirelessPhy set Pt_ 0.28183815
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0

# ======================================================================

TestSuite instproc init {} {
  global opt tracefd topo chan prop RouterTrace
  global node_ god_
  $self instvar ns_ gconf node_ pampa_ pampa2_ pubsub_
  set ns_     [new Simulator]
  set topo [new Topography]
```

```
  if {$opt(out)=="-"} {
    set tracefd stdout
  } else {
    set tracefd [open $opt(out).tr w]
  }
  $ns_ use-newtrace
  $ns_ trace-all $tracefd
  if {$opt(usenam)==1} {
    set namtrace [open $opt(out).nam w]
    $ns_ namtrace-all-wireless $namtrace [expr $opt(x) + $opt(maxborders)] [expr $opt(y) + $opt(maxborders)]
  }
   set chan    [new $opt(chan)]
   set prop    [new $opt(prop)]
   set RouterTrace "ON"
  $topo load_flatgrid [expr $opt(x) + $opt(maxborders)] [expr $opt(y) + $opt(maxborders)]
  puts $tracefd "M 0.0 nn:$opt(nn) x:$opt(x) y:$opt(y) adhocRouting:$opt(rp) sc:$opt(sc)"
  puts $tracefd "M 0.0 matching-subs:$opt(matching-subs) sub-coverage:$opt(sub-coverage) pub-rate:$opt(pub-rate) subscription-checkinterval:$opt(subscription-checkinterval) subscription-timeout:$opt(subscription-timeout)"
  puts $tracefd "M 0.0 prop:$opt(prop) ant:$opt(ant) numsub:$opt(numsub) numpub:$opt(numpub)"
}

Test/pubsub instproc init {} {
  global opt node_ god_ chan topo pampa pampa2 tracefd
  $self instvar ns_ gconf node_ pampa_ pampa2_ pubsub_
  $self next
  if { $opt(rp) == "DSR" } {
    # DSR requires CMUPriQueue, otherwise ns2 will segfault
    set opt(ifqtype) CMUPriQueue
    set opt(ifq) $opt(ifqtype)
  }
  Simulator set mobile_ip_ 0
  set routingAgent_ $opt(rp)
  Simulator set routingAgent_ $opt(rp)
  set god_ [create-god $opt(nn)]
  $ns_ node-config -adhocRouting $opt(rp) \
               -llType $opt(ll) \
               -macType $opt(mac) \
               -antType $opt(ant) \
               -propType $opt(prop) \
               -ifqType $opt(ifqtype) \
               -ifqLen $opt(ifqlen) \
               -phyType $opt(netif) \
               -channel $chan \
               -topoInstance $topo \
               -wiredRouting OFF \
               -agentTrace ON \
               -routerTrace OFF \
               -macTrace ON \
               -phyTrace OFF \
```

```
                -toraDebug OFF \
                -movementTrace OFF
  if { $opt(rp) == "DSR" } {
#$ns_ set-address-format hierarchical
  }
  set gconf [new PubsubGconf]
  puts "Setting GCONF options"
  $gconf "num-nodes" $opt(nn)
  $gconf "alg" $opt(alg)
  $gconf "routing-protocol" $opt(rp)
  $gconf "matching-subs" $opt(matching-subs)
  $gconf "sub-coverage" $opt(sub-coverage)
  $gconf "sub-locdep" $opt(sub-locdep)
  $gconf "pub-rate" $opt(pub-rate)
  $gconf "rj-max-retries" $opt(rj-max-retries)
  $gconf "rxrange" $opt(range)
  $gconf "subscription-checkinterval" $opt(subscription-checkinterval)
  $gconf "subscription-timeout" $opt(subscription-timeout)
  $gconf "proactive-routerecovery" $opt(pr)
  $gconf "routerecovery-limit" $opt(rrl)
  $gconf "intermediate-routeupdate" $opt(ir)
  $gconf "static-subscriptions" $opt(ss)
  $gconf "infotext" $opt(infotext)
  if {$opt(pr)==1} {puts "Proactive routecovery ENABLED!" }
  if {$opt(ir)==1} {puts "Intermediate routeupdate ENABLED!" }
  # Set the destination for log info
  set T [new Trace/Generic]
  $T target [$ns_ set nullAgent_]
  $T attach $tracefd
  $T set src_ -1
  $gconf log-target $T
  puts "Creating nodes and agents"
  for {set i 0} {$i < $opt(nn) } {incr i} {

    if { [string compare $opt(rp) "DSR"] == 0} {
      puts "Using DSR nodes"
      set node_($i) [dsr-create-mobile-node $i]
    } elseif { [string compare $opt(rp) "DSDV"] == 0} {
      puts "Using DSDV nodes"
      dsdv-create-mobile-node $i
    } else {
      set node_($i) [$ns_ node]
    }
#     $node_($i) random-motion 0    ;# disable random motion
#     $node_($i) topography $topo

    if { $pampa } {
      set pampa_($i) [new Agent/PampaAgent]
      $node_($i) attach $pampa_($i) 3323
      $ns_ attach-agent $node_($i) $pampa_($i)
      if {$pampa2 } {
        $pampa_($i) "policy-tdelpwr" [concat "0.2 3 " [RXT $opt(targetdist)] "
            0.25 "]
        set pampa2_($i) [new Agent/PampaAgent]
        $node_($i) attach $pampa2_($i) 3324
```

```
      $ns_ attach-agent $node_($i) $pampa2_($i)
      $pampa2_($i) "policy-delpwr2" [concat "3e6 2 " [RXT $opt(range)] ]
      $pampa2_($i) "portnr" 3324
    } else {
      $pampa_($i) "policy-delpwr" "3e6 2"
    }
  }
  set pubsub_($i) [new Agent/PubsubAgent]
  $node_($i) attach $pubsub_($i) 3330
  $ns_ attach-agent $node_($i) $pubsub_($i)
  $pubsub_($i) "node" $node_($i)
  set T [new Trace/Generic]
  $T target [$ns_ set nullAgent_]
  $T attach $tracefd
  $T set src_ [$node_($i) id]
    $pubsub_($i) "log-target" $T
  if { $pampa } {
    $pampa_($i) "log-target" $T
    $pubsub_($i) pampa-agent $pampa_($i)
    if {$pampa2 } {
      $pampa_($i) "log-target" $T
      $pubsub_($i) pampa-agent2 $pampa2_($i)
    }
  }
}
set fp [open "$opt(sc)" r]
set id [read $fp 2]
close $fp
if { ![string equal $id \x1f\x8b] } {
  puts "Loading uncompressed scenario file $opt(sc)"
  source $opt(sc)
} else {
  puts "Loading gzip'ed scenario file $opt(sc)"
  set fp [open "|gunzip -c <$opt(sc)" r]
  fconfigure $fp -buffering line
  gets $fp data
  while {$data != ""} {
      eval $data
      gets $fp data
  }
  close $fp
}
puts "Load complete, starting simulation..."
puts "Init $opt(numsub) subscribers and $opt(numpub) publishers"
if {$opt(hrl)==1} {
  $gconf "add-hotspot-random-set" $opt(numhotspots)
} else {
  set numsqrt [expr {int(sqrt($opt(numhotspots)))}]
  $gconf "add-hotspot-grid-set" $numsqrt $numsqrt
}
$gconf "init-subscribers" $opt(numsub)
$ns_ at 1.0 "$gconf \"init-publishers\" $opt(numpub)"
if {$opt(stop) >= 6} {
  # Disable all publishers 3 seconds before simulation ends to allow every
      message to get delivered
```

```
    $ns_ at [expr $opt(stop) - 3] "$gconf \"init-publishers\" 0"
    $ns_ at [expr $opt(stop) - 3] "$gconf \"init-subscribers\" 0"
  }
  $ns_ at $opt(stop) "$self finish"
}

Test/pubsub instproc run {} {
  $self instvar ns_
  puts "Starting Simulation..."
  $ns_ run
}

Test/pubsub instproc finish {} {
  $self instvar ns_ gconf node_ pampa_ pampa2_ pubsub_
  global opt pampa pampa2
  puts "NS EXITING..."
  #Tell all nodes when the situation ends
  for {set i 0} {$i < $opt(nn)} {incr i} {
    $node_($i) reset
    if { $pampa } {
      $pampa_($i) reset
      if { $pampa2 } {
        $pampa2_($i) reset
      }
    }
    #$pubsub_($i) reset
  }
  $gconf print-statistics
  $ns_ flush-trace
  $ns_ halt
  puts "finishing..."
  exit 0
}

proc stripzeros {value} {
  set retval [string trimleft $value 0]
  if { ![string length $retval] } {
    return 0
  }
  return $retval
}

proc runtest {arg} {
  global opt pampa pampa2

  if {! [info exists opt(alg)] ||
     ! [info exists opt(sc)] } {
     puts stderr "Required arguments are missing! <alg> and <sc> are mandatory."
     usage
     exit 1
  }
  if { $opt(alg) != "pampa-pubs" &&
      $opt(alg) != "pampa-ucr" &&
      $opt(alg) != "pampa-tree" } {
     puts stderr "Invalid algorithm name"
```

```
      usage
      exit 1
    }

    Phy/WirelessPhy set RXThresh_ [RXT $opt(range)];
    set scenelements [split $opt(sc) "_"]
    if { [llength $scenelements] >= 3 } {
      set numnodes [stripzeros [lindex $scenelements 1]];
      set dens [stripzeros [lindex $scenelements 2]];
      set speed [stripzeros [lindex $scenelements 3]];
      puts "Scenario file name is: nn=$numnodes dens=$dens speed=$speed"
      if { [string is integer -strict $numnodes ] &&
          [string is integer -strict $dens ] &&
          [string is double -strict $speed ]} {
        set opt(density) $dens
        set opt(nn) $numnodes
        set opt(maxspeed) $speed
        set xysize [expr {int(sqrt(1000000 * $opt(nn) / $dens))}]
        if { $xysize > 2490 && $xysize < 2510 } {
          set xysize 2500
        }
        puts "Found density in scen filename: $dens, XY=$xysize,
            numnodes=$numnodes"
        set opt(x) $xysize
        set opt(y) $xysize
      }
    }


    set pampa 0
    set pampa2 0
    if { [string equal -length 5 $opt(alg) "pampa"] } {
      set pampa 1
      if { $opt(alg) == "pampa-tree" } {
        set pampa2 1
      }
    } else {
      puts "TODO"
      exit 1
    }
    set t [new Test/pubsub]
    $t run
}

global argv arg0
default_options
global opt

# load command-line options
for {set i 0} {$i < $argc} {incr i} {
  set arg [lindex $argv $i]
  if {[string range $arg 0 0] != "-"} {
    puts stderr "arg=$arg"
    usage
    exit 1
```

```
  }
  set name [string range $arg 1 end]
  set opt($name) [lindex $argv [expr $i+1]]
  incr i
}
puts "ALG=$opt(alg)"
runtest $argv
```

Listing A.1: TCL simulation file

In addition to the options provided by commandline parameters, some attributes are part of the scenario files and are loaded before the movement patterns. The next lines show an example of the first lines in a scenario file:

```
puts "Setting seed and simulation environment..."
$gconf "set-hotspot-seed" "472204646"
$gconf "set-pub-seed"   "1826071338"
$gconf "set-agent-pub-seed" "-638082678"
$gconf "set-agent-sub-seed" "-51736268"
$gconf "set-agent-timeout-seed" "1640726773"
$gconf "set-gconf-seed" "-365821988"
$gconf "num-nodes" 200
$gconf "scen-file" "scen_200_050_03_rwp_000.1"
$gconf "node-speed" 3
$gconf "movement-model" "rwp"
$gconf "pause-time" 0
$gconf "density" 50
$gconf "xy"    2000 2000
puts "Completed, now loading movements"
```

Listing A.2: Example of scenario file header

A.2 XML configuration file of Stuttgart City Model

This file was used for CanuMobiSim (see [can]) to create scenarios for the Stuttgart City Model. The number of nodes (in the file printed below it says 400) was set to the values given in Table 5.5.

```
<?xml version="1.0"?>
<universe>
  <extension class="de.uni_stuttgart.informatik.canu.mobisim.extensions.NSOutput"
      />
  <extension
      class="de.uni_stuttgart.informatik.canu.mobisim.simulations.TimeSimulation"
      param="1800.00"/>
  <extension
      class="de.uni_stuttgart.informatik.canu.spatialmodel.core.SpatialModel"/>
  <extension class="de.uni_stuttgart.informatik.canu.awmlreader.AWMLReader"
          source="stgt.awml"
          min_x="0"
          max_x="1600"
```

```
        min_y="0"
        max_y="1600"/>
  <step>1.0</step>
  <extension
      class="de.uni_stuttgart.informatik.canu.tripmodel.generators.RandomInitialPositionGenerator"
        name="PosGen">
  </extension>
  <extension
      class="de.uni_stuttgart.informatik.canu.tripmodel.generators.RandomTripGenerator"
      name="TripGen"
        path_algorithm=
            "de.uni_stuttgart.informatik.canu.tripmodel.pathalgorithms.PedestrianStochPathSelection
        theta="0.1">
    <minstay>0.0</minstay>
    <maxstay>30.0</maxstay>
  </extension>
  <nodegroup n="400">
    <extension class="de.uni_stuttgart.informatik.canu.uomm.SmoothMotion"
        initposgenerator="PosGen" tripgenerator="TripGen">
      <minspeed>0.5</minspeed>
      <maxspeed>3.0</maxspeed>
      <minacc>-1.50</minacc>
      <maxacc>1.50</maxacc>
      <speedchangeinterval>10.0</speedchangeinterval>
    </extension>
  </nodegroup>
</universe>
```

Listing A.3: XML configuration file of Stuttgart City Model

A.3 Simulation area of Stuttgart City Model

In Figure A.1 we show a map[1] of the approximated geographical region, used for our simulations of the Stuttgart City model. I represents a square region between the GPS coordinates 48.78335;9.15902 and 48.76834;9.1818.

[1]downloaded from `www.openstreemap.org` at 31/12/2009

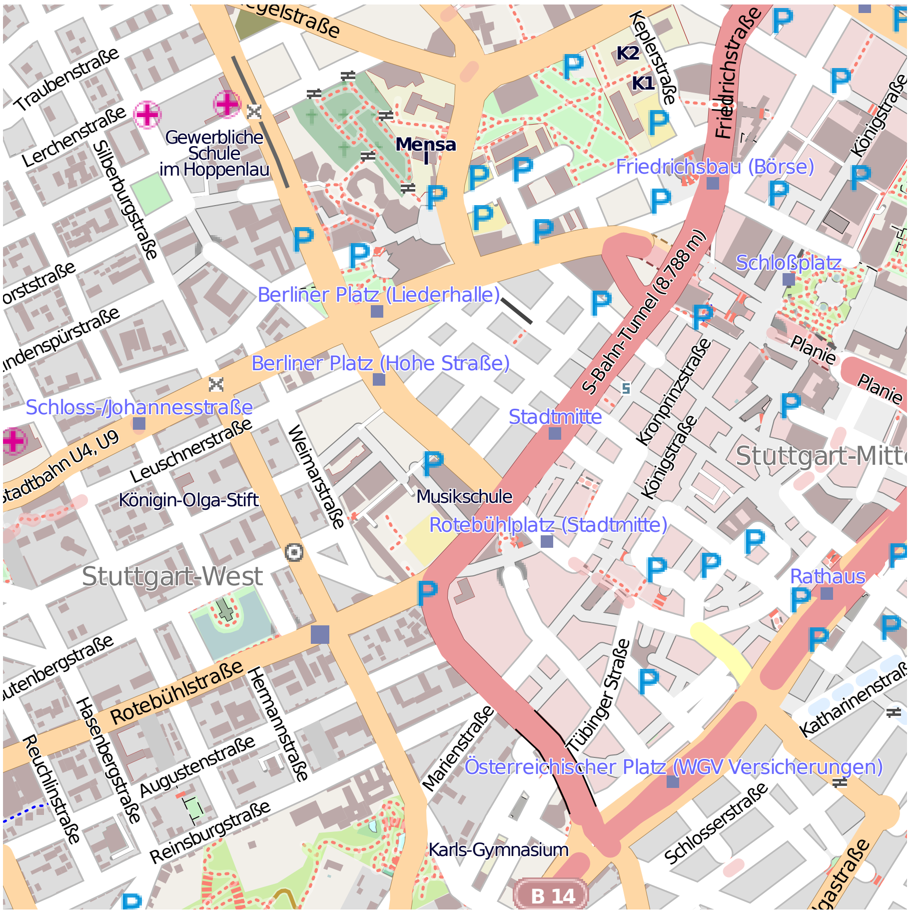

Figure A.1: Map of Stuttgart City Model

Bibliography

[61802] *How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks*, volume 1, 2002. (Cited on pages 3 and 37)

[AMMMQ07] Al Mobaideen, Wesam; Mimi, Hani Mahmoud; Masoud, Fawaz Ahmad; Qaddoura, Emad: Performance evaluation of multicast ad hoc on-demand distance vector protocol: *Comput. Commun.*, 30(9):1931–1941, 2007. (Cited on page 14)

[AT04] Akhtar, Nadeem; Tafazolli, Rahim: Traffic-based multipath routing for mobile ad hoc networks: In *Proceedings of the 2004 International Workshop on Wireless Ad-hoc Networks (IWWAN'04)*, Oulu, Finland, June 2004. Centre for Wireless Communications - University of Oulu. (Cited on page 73)

[Bir93] Birman, Kenneth P.: The process group approach to reliable distributed computing: *Commun. ACM*, 36(12):37–53, 1993. (Cited on page 11)

[BJ87] Birman, Kenneth P.; Joseph, Thomas A.: Exploiting virtual synchrony in distributed systems: *SIGOPS Oper. Syst. Rev.*, 21(5):123–138, 1987. (Cited on pages 3 and 11)

[BKR08] Briones, Jorge A.; Koldehofe, Boris; Rothermel, Kurt: SPINE: Publish/Subscribe for Wireless Mesh Networks through Self-Managed Intersecting Paths: In *Proceedings of the 8th International Conference on Innovative Internet Community Systems (I2CS 2008); Schoelcher, Martinique, June 16-18, 2008*, Schoelcher, Martinique, Juni 2008. online proceedings. (Cited on pages 12 and 15)

[Blo70] Bloom, Burton H.: Space/time trade-offs in hash coding with allowable errors: *Commun. ACM*, 13(7):422–426, 1970. (Cited on page 22)

[BMB$^+$00] Bacon, Jean; Moody, Ken; Bates, John; Hayton, Richard; Ma, Chaoying; McNeil, Andrew; Seidel, Oliver; Spiteri, Mark: Generic support for distributed applications: *Computer*, 33(3):68–76, 2000. (Cited on pages 12 and 15)

[bon] Bonnmotion – a mobility scenario generation and analysis tool: `http://net.cs.uni-bonn.de/wg/cs/applications/bonnmotion/`. (Cited on page 65)

[BRS03] Bettstetter, Christian; Resta, Giovanni; Santi, Paolo: The node distribution of the random waypoint mobility model for wireless ad hoc networks: *IEEE Transactions on Mobile Computing*, 2(3):257–269, Jul/Sep 2003. (Cited on pages 52, 56, 65 and 68)

[can] Canu mobility simulation environment (canumobisim): `http://canu.informatik.uni-stuttgart.de/mobisim/index.html`. (Cited on pages 65, 68 and 86)

[CBRP05] Chakeres, Ian D.; Belding-Royer, Elizabeth M.; Perkins, Charles E.: Dynamic MANET On-demand (DYMO) Routing Rrotocol: *IETF Internet Draft, draft-ietf-manet-dymo-01.txt*, May 2005: (Work in Progress). (Cited on pages 10 and 35)

[CCG+07] Costa, Paolo; Coulson, Geoff; Gold, Richard; Lad, Manish; Mascolo, Cecilia; Mottola, Luca; Picco, Gian Pietro; Sivaharan, Thirunavukkarasu; Weerasinghe, Nirmal; Zachariadis, Stefanos: The runes middleware for networked embedded systems and its application in a disaster management scenario: In *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 69–78, Washington, DC, USA, 2007. IEEE Computer Society. (Cited on page 4)

[CDNF01] Cugola, Gianpaolo; Di Nitto, Elisabetta; Fuggetta, Alfonso: The jedi event-based infrastructure and its application to the development of the opss wfms: *IEEE Trans. Softw. Eng.*, 27(9):827–850, 2001. (Cited on pages 12 and 15)

[CE95] Corson, M. Scott; Ephremides, Anthony: A distributed routing algorithm for mobile wireless networks: *Wireless Networks*, 1(1):61–81, 1995. (Cited on page 73)

[CIP02] Caporuscio, Mauro; Inverardi, Paola; Pelliccione, Patrizio: Formal analysis of clients mobility in the siena publish/subscribe middleware, 2002. (Cited on pages 12 and 15)

[CJ03] Clausen, T.; Jacquet, P.: Rfc-3626 optimized link state routing protocol (olsr), 2003. (Cited on page 10)

[CRW01] Carzaniga, Antonio; Rosenblum, David S.; Wolf, Alexander L.: Design and evaluation of a wide-area event notification service: *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001. (Cited on pages 12 and 15)

[DBJB01] David B. Johnson, David A. Maltz; Broch, Josh: *DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. in Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5, pp. 139-172*: Addison-Wesley, 2001. (Cited on pages 10, 19 and 71)

[DPR00] Das, Samir R.; Perkins, Charles E.; Royer, Elizabeth M.: Performance comparison of two on-demand routing protocols for ad hoc networks, 2000. (Cited on pages 11, 20 and 25)

[DRWT96] Dube, Rohit; Rais, Cynthia D.; Wang, Kuang-Yeh; Tripathi, Satish K.: Signal stability based adaptive routing (ssa) for ad-hoc mobile networks: Technical report, College Park, MD, USA, 1996. (Cited on pages 13, 15 and 30)

[DSM09] Denko, Mieso K.; Shakshuki, Elhadi; Malik, Haroon: Enhanced cross-layer based middleware for mobile ad hoc networks: *J. Netw. Comput. Appl.*, 32(2):490–499, 2009. (Cited on pages 13, 15 and 17)

[EFGK03] Eugster, Patrick Th.; Felber, Pascal A.; Guerraoui, Rachid; Kermarrec, Anne-Marie: The many faces of publish/subscribe: *ACM Comput. Surv.*, 35(2):114–131, 2003. (Cited on pages 5 and 11)

[ENP01] Elisabetta, Gianpaolo Cugola; Nitto, Elisabetta Di; Picco, Gian Pietro: Content-based dispatching in a mobile environment: In *In Proceeding of WSDAAL 2000*, 2001. (Cited on page 5)

[Fee01] Feeney, Laura Marie: An energy consumption model for performance analysis of routing protocols for mobile ad hoc networks: *Mobile Networks and Applications*, 6(3):239–249, 2001. (Cited on pages 27 and 42)

[FLP+00] Fabret, Francoise; Llirbat, Francois; Pereira, Jo Ao; Rocquencourt, Inria; Rocquencourt, Inria; Shasha, Dennis: Publish/subscribe on the web at extreme speed: In *In Proc. of ACM SIGMOD Conf. on Management of Data*, pages 627–630, 2000. (Cited on pages 11 and 15)

[FMG02] Fiege, Ludger; Mühl, Gero; Gärtner, Felix C.: A modular approach to build structured event-based systems: In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 385–392, New York, NY, USA, 2002. ACM. (Cited on pages 11 and 15)

[FV] Fall, Kevin; Varadhan, Kannan: The *ns* manual, from january 6, 2009: `http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf`. (Cited on page 43)

[GM08] Gui, Chao; Mohapatra, Prasant: A framework for self-healing and optimizing routing techniques for mobile ad hoc networks: *Wirel. Netw.*, 14(1):29–46, 2008. (Cited on page 73)

[GMR] Garbinato, Benoît; Miranda, Hugo; Rodrigues, Luís. (Cited on pages 4, 7 and 11)

[gry01] Gryphon: Publish/subscribe over public networks: Technical report, 2001. (Cited on pages 11 and 15)

[HHL02] Haas, Zygmunt J.; Halpern, Joseph Y.; Li, Li: Gossip-based ad hoc routing: In *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, volume 3, pages 1707–1716. IEEE, June 23–27 2002. (Cited on pages 8 and 76)

[HHL06] Haas, Zygmunt J.; Halpern, Joseph Y.; Li, Li: Gossip-based ad hoc routing: *IEEE/ACM Trans. Netw.*, 14(3):479–491, 2006. (Cited on page 8)

[JF08] Jerzak, Zbigniew; Fetzer, Christof: Bloom filter based routing for content-based publish/subscribe: In *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*, pages 71–81, New York, NY, USA, 2008. ACM. (Cited on page 22)

[JH01] Jetcheva, Jorjeta G.; Hu, Yih-Chun: A Simple Protocol for Multicast and Broadcast in Mobile Ad Hoc Networks: *IETF Internet Draft, draft-ietf-manet-simple-mbcast-01.pdf*, July 2001. (Cited on page 7)

[JM96] Johnson, David B.; Maltz, David A.: Dynamic source routing in ad hoc wireless networks: In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996. (Cited on pages 10, 19, 65, 71, 72 and 73)

[Khe07] Khelil, Abdelmajid: *A Generalized Broadcasting Technique for Mobile Ad Hoc Networks*: PhD thesis, Stuttgart, DE, April 2007. (Cited on page 7)

[KS93] Krishnan, R.; Silvester, J.A.: Choice of allocation granularity in multipath source routing schemes: In *Proceedings of the Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future. (INFOCOM '93)*, pages 322–329, San Francisco, CA USA, March 1993. (Cited on page 73)

[KSK99] Kalasapur, Swaroop; Senthivel, Kumarvel; Kumar, Mohan: Elizabeth m. royer and charles e. perkins: In *Proceedings of MobiCom '99*, pages 207–218, 1999. (Cited on pages 13 and 21)

[KSK06] Kalasapur, Swaroop; Senthivel, Kumarvel; Kumar, Mohan: Service oriented pervasive computing for emergency response systems: In *PERCOMW '06: Proceedings of the 4th annual IEEE international conference on Pervasive Computing and Communications Workshops*, page 517, Washington, DC, USA, 2006. IEEE Computer Society. (Cited on page 4)

[LG01] Lee, S.-J.; Gerla, M.: Split multipath routing with maximally disjoint paths in ad hoc networks: In *Proceedings of IEEE International Conference on Communications (ICC 2001)*, volume 10, pages 3201–3205, Helsinki, Finland, June 11–14 2001. (Cited on page 73)

[LMP] Luca Mottola, Gianpaolo Cugola; Picco, Gian Pietro: A self-repairing tree topology enabling content-based routing in mobile ad hoc networks. (Cited on pages 14, 15, 21 and 73)

[MBGK+08] Mocito, José; Briones-García, J. Alfonso; Koldehofe, Boris; Miranda, Hugo; Rodrigues, Luís: Geographical distribution of subscriptions for content-based publish/subscribe in manets: In *Companion '08: Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, pages 102–103, New York, NY, USA, 2008. ACM. (Cited on page 21)

[MC02] Meier, René; Cahill, Vinny: Steam: Event-based middleware for wireless ad hoc network: In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 639–644, Washington, DC, USA, 2002. IEEE Computer Society. (Cited on pages 13 and 15)

[Mir07] Miranda, Hugo: *Gossip-Based Data Distribution in Mobile Ad Hoc Networks, PhD Thesis*: PhD thesis, Lisbon, PT, October 2007. (Cited on pages 9, 18, 27, 29 and 33)

[MLRR06] Miranda, Hugo; Leggio, Simone; Rodrigues, Luís; Raatikainen, Kimmo: A power-aware broadcasting algorithm: In *Proceedings of The 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'06)*, Helsinki, Finland, September 11–14 2006. University of Oulu. (Cited on pages 9 and 27)

[MLRR09] Miranda, Hugo; Leggio, Simone; Rodrigues, Luís; Raatikainen, Kimmo: An algorithm for dissemination and retrieval of information in wireless ad hoc networks: *Concurr. Comput. : Pract. Exper.*, 21(7):889–904, 2009. (Cited on pages 9 and 21)

[MM] Matos, João; Miranda, Hugo: Algoritmos de difusão para protocolos de encaminhamento em redes ad hoc sem fios? (Cited on pages 11, 30 and 73)

[noa] Noah – no ad-hoc routing agent for ns-2: `http://icapeople.epfl.ch/widmer/uwb/ns-2/noah/`. (Cited on page 43)

[ns2] The network simulator – ns-2: `http://www.isi.edu/nsnam/ns/`. (Cited on page 43)

[NTCS99] Ni, Sze-Yao; Tseng, Yu-Chee; Chen, Yuh-Shyan; Sheu, Jang-Ping: The broadcast storm problem in a mobile ad hoc network: In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, New York, NY, USA, 1999. ACM. (Cited on pages 7 and 8)

[NV09] Nitnaware, Dhiraj; Verma, Ajay: Energy evaluation of proactive and reactive protocol for manet under on/off source traffic: In *ICAC3 '09: Proceedings of the International Conference on Advances in Computing, Communication and Control*, pages 451–455, New York, NY, USA, 2009. ACM. (Cited on page 10)

[pam] Pampa – power-aware message propagation algorithm: `http://homepages.di.fc.ul.pt/~hmiranda/pampa/`. (Cited on pages 29 and 43)

[PB94] Perkins, Charles; Bhagwat, Pravin: Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers: pages 234–244, 1994. (Cited on page 10)

[PB02] Pietzuch, Peter R.; Bacon, Jean: Hermes: A distributed event-based middleware architecture: In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 611–618, Washington, DC, USA, 2002. IEEE Computer Society. (Cited on page 11)

[PR02] Perkins, Charles E.; Royer, Elizabeth M.: Ad-hoc on-demand distance vector routing: In *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 2002. (Cited on pages 10, 11, 19, 71 and 73)

[RHKS01] Ratnasamy, Sylvia; Handley, Mark; Karp, Richard; Shenker, Scott: Application-level multicast using content-addressable networks: pages 14–29, 2001. (Cited on page 15)

[RKCD01] Rowstron, Antony; Kermarrec, Anne-Marie; Castro, Miguel; Druschel, Peter: Scribe: The design of a large-scale event notification infrastructure: In *In Networked Group Communication*, pages 30–43, 2001. (Cited on pages 11 and 15)

[RW97] Rosenblum, David S.; Wolf, Alexander L.: A design framework for internet-scale event observation and notification: *SIGSOFT Softw. Eng. Notes*, 22(6):344–360, 1997. (Cited on page 3)

[SA97] Segall, Bill; Arnold, David: Elvin has left the building: A publish/subscribe notification service with quenching: In *In Proceedings of the 1997 Australian UNIX Users Group*, Queensland, Australia, 1997. Distributed Systems Technology Centre University of Queensland. (Cited on pages 11 and 15)

[SB] Mohamed OthmanShafinaz Buruhanudeen, Mazliza Othman Borhanuddin Mohd Ali. (Cited on page 65)

[SCS03] Sasson, Y.; Cavin, D.; Schiper, A.: Probabilistic broadcast for flooding in wireless mobile ad hoc networks: In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 2, pages 1124–1130 vol.2, March 2003. (Cited on page 8)

[TMS03] Tickoo, Omesh; Member, Student; Sikdar, Biplab: On the impact of ieee 802.11 mac on traffic characteristics: *IEEE Journal on Selected Areas in Communications*, 21:189–203, 2003. (Cited on pages 3 and 37)

[umt98] Universal Mobile Telecommunications System (UMTS); Selection procedures for the choice of radio transmission technologies of the UMTS (UMTS 30.03 version 3.2.0): `http://www.3gpp.org/ftp/Specs/archive/30_series/30.03U/3003U-320.zip`, European Telecommunications Standards Institute, 1998. (Cited on page 65)

[WC02] Williams, Brad; Camp, Tracy: Comparison of broadcasting techniques for mobile ad hoc networks: In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 194–205, New York, NY, USA, 2002. ACM. (Cited on page 7)

[WK97] Wesel, Ellen Kayata; Khayata, Ellen: *Wireless Multimedia Communications: Networking Video, Voice and Data*: Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. (Cited on page 30)

[YB04] Yoneki, Eiko; Bacon, Jean: An adaptive approach to content-based subscription in mobile ad hoc networks: In *PERCOMW '04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, page 92, Washington, DC, USA, 2004. IEEE Computer Society. (Cited on page 14)

[YB05] Yoneki, Eiko; Bacon, Jean: Dynamic group communication in mobile peer-to-peer environments: In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 986–992, New York, NY, USA, 2005. ACM. (Cited on pages 14 and 15)

[YLN03] Yoon, J.; Liu, M.; Noble, B.: Random waypoint considered harmful: In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1312–1321 vol.2, March-3 April 2003. (Cited on page 65)

All links were last followed on January 8, 2010.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Ostfildern, 12.01.2010 ______________________________

Stephan Schnitzer