Probabilistic Addressing to Support Mobility in Wireless Networks

Master Thesis

Andrea Crotti

RWTH Aachen University, Germany

Chair for Communication and Distributed Systems

Advisors:

Prof. Dr.-Ing Klaus Wehrle
Prof. Alberto Montresor

Registration date: 2010-11-19
Submission date: 2011-06-16

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, August 31, 2011

Abstract

We propose Mobile Probabilistic ADdressing (MPAD), a virtual-coordinates-based routing protocol for Wireless Mesh Networks (WMNs). In contrast to other virtual coordinates based protocols such as Beacon Vector Routing (BVR), our approach collects statistical distributions of hop distances, thereby avoiding explicit link estimation. MPAD therefore captures topology changes arising from link variability and mobility more gracefully.

We implemented our prototype for wireless mesh nodes and the OMNeT++ simulator, and evaluated it on a real testbed and in simulation. In the simulation we evaluated three different mobility scenarios. From the results that we obtained, we can conclude that MPAD is a feasible routing protocol for WMNs.

Acknowledgments

Thanks to Prof. Dr.-Ing. Klaus Wehrle, for giving me the opportunity to work in his group, full of interesting people and fascinating research projects. I am grateful also to my second supervisor Prof. Alberto Montresor, who has always available and interested in the evolution of my work during the last months.

Thanks to my supervisors Muhammad Hamad Alizai, Tobias Vaegs and Hanno Wirtz, who were infinitely patient in answering to my thousands of emails per week. They constantly supported me and followed my progress very closely, and that was really helpful to avoid losing the focus.

Thanks to all the colleagues that helped me, reviewing my several English horrors and many mistakes that looked perfectly correct in my eyes. Thanks in particular to Diego, Danilo and Jó, for the long hours spent reading my own material full of Italianisms and filled with way too much implicit information.

One thing that I surely accomplished is to become a better kicker player, so thanks to all the colleagues that I sent (and less thanks to the ones that sent me) under the kicker table. It was always a very nice way to get some distraction from the monitor.

Thanks to my parents for the support received in all these years of study, and in particular to my little sisters Sara and Anna, who always made me smile also when nothing seemed to work.

Last but not least thanks to my girlfriend Julia, who supported me in all these months of work in the lab, always able to keep me in a good mood and relaxing from the thesis stress.

Contents

1 Introduction **1**

1.1 Problem statement 1

1.2 Our solution 2

1.3 Outline 2

2 Background **5**

2.1 Wireless links 5

2.2 Mesh networks 6

2.3 Routing 8

2.4 Routing in mesh networks 8

2.5 Link estimation 9

2.6 Mathematical background 10

2.6.1 Pearson's χ^2-Test 10

2.6.2 Exponential moving average 11

3 Related work **13**

3.1 OLSR 13

3.2 AODV 14

3.3 BVR 15

3.4 PAD 16

3.5 GPSR 17

3.6 NOGEO 18

3.7 DART 19

4 Design **21**

4.1 Design concepts 21

4.2 Packet types 22

4.3 Landmarks 23

4.3.1 Location service 23

4.4 Addressing 24

4.4.1 Identifier 24

4.4.2 Virtual coordinates 24

4.4.3 Statistical addresses 25

4.5 Routing 26

5 Implementation **29**

5.1 Environment 29

5.2 Data structures 31

5.3 Neighbor table management 33

5.4 Packet structure 35

6 Evaluation **37**

6.1 Evaluation environments 37

6.2 Analysis 38

6.3 Network and mobility patterns 39

6.4 Delivery rate 40

6.4.1 Static 41

6.4.2 Close 42

6.4.3 Around 43

6.4.4 Inside 44

6.4.5 Testbed 45

7 Conclusion **47**

7.1 Future work 48

A Appendix **49**

A.1 Runtime MPAD parameters 52

Bibliography **55**

List of Figures **58**

List of Tables **61**

1

Introduction

Wireless technologies are nowadays extremely pervasive, more and more devices come with IEEE 802.11 networking capabilities. The most common way to use these wireless devices is to connect to an access point, which is connected to the wired network infrastructure. But these access points have a very limited range and the wired infrastructure is costly to set up and maintain. A multihop wireless network where not all access points need to be in radio range would greatly reduce the need for wired infrastructure.

Wireless mesh networks (WMNs) [16] are collections of mobile nodes connected over a wireless medium. As an extension to an existing wired infrastructure this approach gives a big advantage in terms of cost and flexibility. These advantages have, however, the drawback of increased protocol complexity. In traditional infrastructure networks, client nodes only need of communicate with the access point, and there are no client-to-client connections. In mesh networks, each network node has to be able to route data trough the network, becoming itself part of the infrastructure. Therefore it is necessary to implement a reliable and scalable routing protocol in each node.

1.1 Problem statement

Routing in WMNs is challenging. There are many issues to face when designing a robust and scalable protocol. First of all, data packets have to be routed over multiple hops, and given the unreliability of wireless links, this increases the risk of losing data. Moreover, to reach the final destination there can be multiple paths, and choosing the best one is not trivial.

The development of routing protocols for WMNs is still fairly new, and there are no standardized solutions to adopt on top of the MAC layer yet, in particular for the scenario that we are interested in: a WMN where it is possible to set up an infrastructure, and there are one or more mobile nodes. Several solutions have been

proposed in the literature ([21] [20] [18] [25] [27] [24] [31] [19] [1]), but the research of an efficient and scalable protocol remains an open field.

1.2 Our solution

The routing protocol we developed is called MPAD, which stands for **Mobile Probabilistic ADdressing**, and as the name suggests it strongly focuses on mobility. To support mobility a routing protocol needs to adapt quickly to network changes, while minimizing the protocol overhead. The "Probabilistic ADdressing" part of the name stands for the main concept behind the design of MPAD, as explained in detail in Chapter 4.

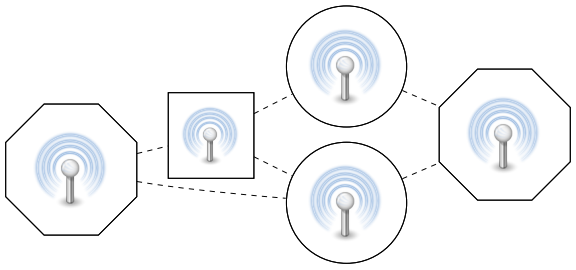


Figure 1.1 Example of an MPAD network, with two landmarks (octagons), one mobile node (square) and two static nodes (circles).

In MPAD the network is characterized by landmarks, static nodes and a distinct set of mobile nodes, as shown in Figure 1.1. The landmarks and the static nodes are the infrastructure of the network, while the mobile nodes represent the clients. MPAD uses virtual coordinates as addressing scheme, which means that every coordinate is a vector of distances to the landmarks, and this vector represents the relations between the neighbors. Whenever relations between nodes change, their coordinates are also likely to change. These coordinates need to be published in the network, and in a dynamic network this might create a big protocol overhead. The main idea behind MPAD is to use statistical functions to analyze the history of the coordinates and minimize the necessary address updates.

1.3 Outline

The rest of the thesis is organized as follows. Chapter 2 provides basic knowledge about wireless mesh networks, routing protocols, link dynamics and a couple of mathematical functions important for MPAD. Chapter 3 gives an overview of other protocols already developed in this area, focusing on the general design and how addressing is implemented in each of them. Then in Chapter 4 we talk about the design of the MPAD protocol, explaining the main data structures and algorithms that compose the protocol. Since Probabilistic ADdressing (PAD) [29] and MPAD are very similar some common design decisions of PAD and MPAD will be discussed here as well.

Chapter 5 gives an overview of the prototype implementation and of the entire simulation environment. Chapter 6 introduces the evaluation environments and

shows some results obtained on the testbed and the OMNeT++ simulation. Lastly Chapter 7 draws some conclusions about MPAD and outlines some possible future enhancements to MPAD.

2

Background

This chapter gives the background knowledge necessary to understand the rest of this work. It first introduces wireless links and related properties, then it defines what are wireless ad hoc and mesh networks. The general concept of routing is then introduced, focusing on routing in Wireless Mesh Networks (WMNs).

Section 2.1 explains the properties of wireless links, and how they differ from wired connections. Section 2.4 defines basic concepts like wireless, adhoc and WMNs. Section 2.3 talks about the general concept of routing, and Section 2.4 focuses on routing in WMNs, and the related challenges. Section 2.5 gives an introduction to link estimation, an important technique that we will see used in some protocols in Chapter 3. The last section defines χ^2-Test and Exponential Moving Average (EMA) functions, both part of MPAD implementation.

2.1 Wireless links

Wireless connections have a very different nature compared to wired network connections. A signal through the wire propagates through copper directly to the destination, and the copper wire is shielded and isolated from external interferences. A wireless signal instead travels through air, without any isolation from the environment. Air contains all kind of pollution and obstacles, which can stop or deflect the signal in unpredictable ways. If the nodes also move while sending, the situation becomes even more complex, and physical phenomena like the Doppler effect may also influence the signal. Another important source of interferences is the vicinity of other devices that emit electromagnetic signals. In the range of 2.4 GHz (in which Wireless Local Area Networks (WLANs) usually work) there are several possible sources of interfering signals, such for example microwave ovens, wireless phones, Bluetooth devices or other WLANs.

An important issue to deal with is the intrinsic asymmetry of wireless links. Whenever A and B are in the same radio range, the fact that A is able to receive data from

B does not directly imply that B is also able to receive data from A. Environmental conditions can cause this phenomena as well as simply the fact that B has a greater transmission power than A, as for example shown in Figure 2.1.

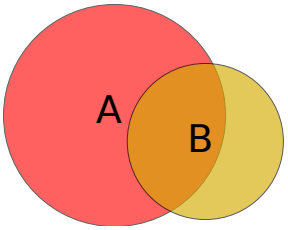


Figure 2.1 Example of two wireless nodes, where B is in the radio range of A but not vice versa

2.2 Mesh networks

A generic *wireless network* refers to any type of network composed by devices that are connected together via wireless. This generic definition includes different kind of networks such as Bluetooth, WPAN, WMAN and so on. In this thesis when speaking about wireless network we refer to WLAN (or Wi-Fi), as described by the IEEE 802.11 standard [12].

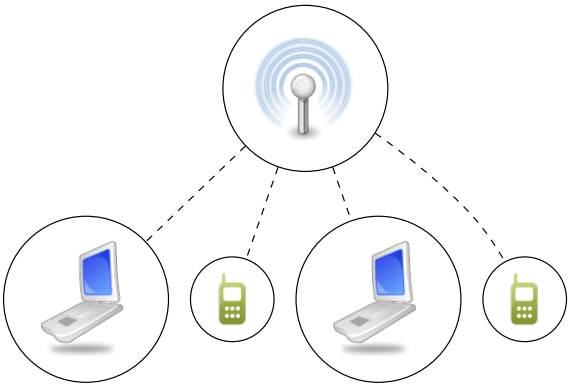


Figure 2.2 Example of an infrastructure wireless network, with an access point and 4 heterogeneous clients connected to it

A IEEE 802.11 wireless network can work in two modes, *infrastructure* and *ad hoc* mode. In infrastructure mode, as in Figure 2.2, each of the wireless clients connects to the same access point, which enables a client-to-client communication, and optionally connections to the external network. The access point announces itself using periodic beacon frames, and the clients connect to it using its Service Set IDentifier (SSID).

In ad hoc mode, there is no access point and every node can communicate with each other, as long as they set the same SSID and use the same channel. Ad hoc networks can be used to gather together a small number of wireless devices in a network, without the cost of the infrastructure.

Wireless mesh networks [16] are networks built with a mix of fixed and mobile nodes interconnected via wireless links to form a multihop ad hoc network. The

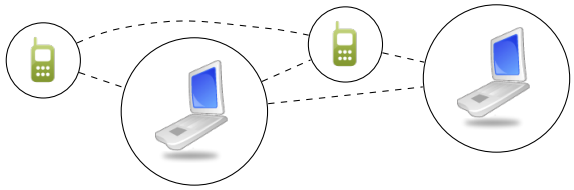


Figure 2.3 Example of an ad hoc mode network, with nodes connected directly to each other

main difference between ad hoc networks and WMNs is their usage. While ad hoc networks are mainly intended for temporary small standalone networks composed of heterogeneous devices, WMNs are designed to be highly scalable and extend existing network infrastructure (as shown in Figure 2.4).

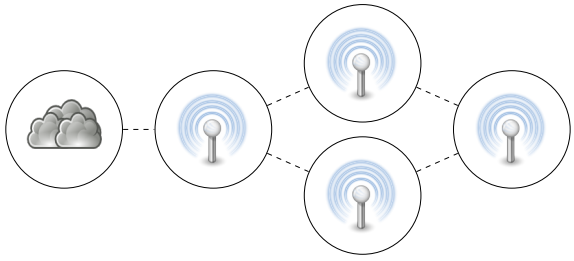


Figure 2.4 Example of a mesh network, where one of the nodes is directly connected to an external network and provides Internet connection to the other nodes

In a mesh network every node has a set of neighbors, where a neighbor is defined as a node from which packet can be received. Given the asymmetry of wireless links, the neighborhood relationship is not symmetric, and it is also likely to vary quickly with time due to link quality fluctuations.

There are a few advantages in deploying a WMNs over a traditional purely infrastructure network:

- **Cost**: The most expensive part of infrastructure wireless networks is the deployment of access points. Access point radio range is limited, and since infrastructure networks are single-hop is necessary to spread a big number of access points to cover the whole area. A mesh network on the other hand supports multi-hop communication with the access points and use other clients to connect to them, increasing the coverage and reducing the cost of the infrastructure.

- **Reliability**: A mesh network provides redundant paths. Robustness and resilience to hardware failures is also ensured by the existence of multiple routes.

- **Self management**: The setup of a mesh network node is totally transparent to the user. When adding nodes, the network reorganizes itself automatically by adding a new route. A mesh network can therefore be easily expanded without additional costs or interruptions of service for the users.

These advantages come at a price, a mesh network does not normally have the same performances of an infrastructure network, and there is a big increase in complexity in the node software, as we will see in more detail in Section 2.4.

An interesting application of WMNs is *community networks*, wireless networks which are WMNs created to provide advantages to a community. The roofnet [15] project is one example of a community network, used to provide Internet connectivity to the MIT campus. Another important community network is the freifunk [4] project. WMNs can be employed to implement an intelligent transport system, as it was done in Portsmouth [9]. In Como, WMNs have been used to create the largest monitor system based on WMNs [3]. In the OLPC project [8] every laptop is designed explicitly to act as a mesh network node, to create a cheap and flexible network in third-world countries lacking infrastructure.

2.3 Routing

Routing can be defined as the process of selecting a path through the network to deliver a data packet from one node to another node. The main general goal of a routing protocol is to determine a path to the destination, and how to compute it is highly dependent on the metric analyzed. The easiest metric to compute is the number of hop counts, which is defined as the number of hops that have to be traversed from the sender to the destination. In case the hop count is the only metric used, the best route is the one with the smallest number of hops. But hop count can be misleading if used as the only metric, because link quality is highly variable.

Other important metrics to consider are:

- **Maximum bandwidth**: bit rate measure of the available resources on the link. The bandwidth indicates an upper bound on the data that can be sent over a certain link, but it gives no indication on how much bandwidth is actually available on the channel.
- **Load**: amount of traffic utilizing the links, dynamically updated on the real conditions of the link.
- **Delay**: measure of the time that it takes for a packet to traverse a route, where the path with the least delay is considered the best path.
- **Reliability**: likelihood that the link will fail to deliver the packet.

2.4 Routing in mesh networks

In WMNs every node must be able to route data since there is no central access point. Unlike wired networks, each node has only one network interface to route data, typically a wireless IEEE 802.11 adapter in ad hoc mode.

A routing protocol for WMNs should have the following properties:

- **Overhead localization**: A change in the state of one node should affect its neighborhood. When this property is not satisfied every local change generates a global overhead, reducing the scalability of the protocol.

- **Decentralized protocol**: The necessary state that has to be maintained on each node should be as small as possible, since the amount of memory on the nodes in the mesh network is limited.
- **Zero configuration**: The need for manual configuration should be minimized. Once an initial set of mesh nodes is set up and the network is configured the routing protocol should adapt automatically to changes, such as for example adding removing or moving nodes in the mesh.
- **Minimal hardware requirements**: A routing protocol should not depend on particular hardware other than a wireless transmitter. GPSR [25] for example requires a GPS sensor to work, thus limiting its use whenever a GPS sensor is not feasible or convenient.

There are three main approaches to address the problem of routing in mesh networks. A first approach continuously spreads the state of the network links through the network, to keep every node up to date about the whole topology. This approach is called the *proactive* routing and it is used for example in OLSR [20].

A second approach is to keep the network idle and floods the network to request routes on demand. This is called *reactive* routing and it is for example employed in AODV [18]. Whenever a node needs to send data to another node in the network, it floods the network with a request and, whenever the destination node is found, a route answer is sent back to the source. Both these approaches generate explicit routes which are then used to send data from source to destination.

Then there is a third possible approach, which does not need to generate explicit routes, but simplifies the routing decisions by moving the complexity to the addressing scheme. These protocols, which include for example BVR [22] or GPSR [25], do not need to generate the route to the destination in the sender, but each node has enough local information to select an optimal next hop to reach the final destination. GPSR uses geographic coordinates and selects greedily the next hop to the destination as the geometrically closest neighbor. BVR instead accomplish the same result using virtual coordinates.

2.5 Link estimation

Link estimation is a technique used to predict the chance of being able to deliver data between two neighboring wireless nodes. A *link estimator* can be **passive** or **active**. A passive link estimator listens to data packets that are sent and received without producing any extra traffic. An active link estimator instead periodically probes the connection to the neighbors to derive the link quality, producing a slight amount of extra traffic to get more precise information about the link quality. The Packet Reception Rate (PRR) represents the percentage of successfully received packets during a given time interval. To take the inherent asymmetry of wireless links into consideration, the product of the PRR in both directions represents the overall quality of the link between two nodes.

There are two possible approaches to determine the quality of links, *long-term* and *short-term* link estimation. A long-term link estimator identifies nodes with a stable

connection over a long period of time. Links with a PRR > 90% are considered *good links*, but good links are often close, so always choosing them might dramatically increase the number of hops to reach the destination. The other available links might provide a bad quality, but some of them (with a PRR between 10% and 90%) oscillate between good and bad link quality, and this goes undetected in long-term estimators.

With short-term link estimators, instead of trying to derive the overall link quality in a long period of time, the link is analyzed just before sending the data packet. Whenever several packets are successfully delivered over a link, the link is used immediately. If even very few packets are lost the link is not used anymore for some time. Short-term link estimators can only work if there are bursts of data to be sent each time. If packets are only sent sporadically and in small chunks short-term estimation is not possible, but if applicable, this form of estimator adapts more quickly to changes in the network topology.

2.6 Mathematical background

This section gives a mathematical introduction to a couple of functions which are used in the implementation of our routing protocol. Pearson's chi^2-test is a widely used test to check variation between different statistical distributions, while the *exponential moving average* is used to compute a weighted average on a sequence.

2.6.1 Pearson's χ^2-Test

Persons's χ^2-Test computes the difference between frequency distributions. The χ^2-Test is based on a test statistic that measures divergence of the observed data from a given null hypothesis. The formula to compute the χ^2-statistic is:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{2.1}$$

Where O_i and E_i are the observed and expected results. For example suppose that we want to analyze the distribution of the results generated by rolling a die n times. The null hypothesis is that the die is perfectly fair, which means that every face has $\frac{1}{6}$ probability, in every roll of the die. So first we create 6 categories, one for each possible number of the die. Those categories hold the occurrences for the observed results for each number.

For example suppose that after 60 rolls the we get the result in Table 2.5. The expected number of occurrences for each of the number under the hypothesis of a fair die is: $\frac{60}{6} = 10$
And the χ^2 is computed as:

$$\chi^2 = \frac{(10-10)^2}{10} + \frac{(10-9)^2}{10} + \ldots = 1.0 \tag{2.2}$$

Face	Occurrences
1	10
2	9
3	12
4	10
5	8
6	11

Table 2.5 Number of occurrences for each face of the die

This value has to be normalized to be meaningful as a fitness test. Normalization implies the computation of the **p-value**, and to compute the p-value it is first necessary to compute the degree of freedom. The **degree of freedom** is the number of independent pieces of information available to estimate another piece of information, and it is normally computed as the number of categories minus one.

The intuition behind it is that since the probabilities of independent events always sum up to 1, once we have $n-1$ probabilities we can also derive the missing one.

Once the degree of freedom and the χ^2 are computed, we can compute the p-value (how to compute it is not relevant for the scope of this thesis). This p-value is used as a threshold to decide if the distributions have a statistically significant variation. As a rule of thumb a p-value of at most 5% or 1% is used to consider the distributions statistically different.

2.6.2 Exponential moving average

The Exponential Moving Average (EMA) is a Finite Impulse Response (FIR) that applies weighting factors which decrease exponentially. The formula to calculate the EMA at periods $t > 2$ is:

$$S[t] = \alpha Y_t + (1-\alpha)S_{t-1} \tag{2.3}$$

- The α coefficient represents the speed of weighting decrease, for $\alpha < 0.5$ older values have a bigger weight, and vice versa when $\alpha > 0.5$.
- Y_t is the value of the observation at time t.
- S_t is the value of the EMA at any time period t.

For example given a sequence of events $\{1, 1, 2, 2\}$ the EMA computed for $\alpha = 0.7$ would be:

$$S[3] = 0.7 \times 2 + (1-0.7)S[2] \tag{2.4}$$
$$S[2] = 0.7 \times 2 + (1-0.7)S[1] \tag{2.5}$$
$$S[1] = 0.7 \times 1 + (1-0.7)S[0] \tag{2.6}$$
$$S[0] = 0.7 \times 1 = 0.7 \tag{2.7}$$

Substituting back the computed values in the recursion tree gives $S[3] = 1.91$, which is the EMA computed on the sequence for $\alpha = 0.7$.

The exponential moving average is used for example when we need to give more weight to the new values, and it is used in MPAD, as shown in Chapter 5.

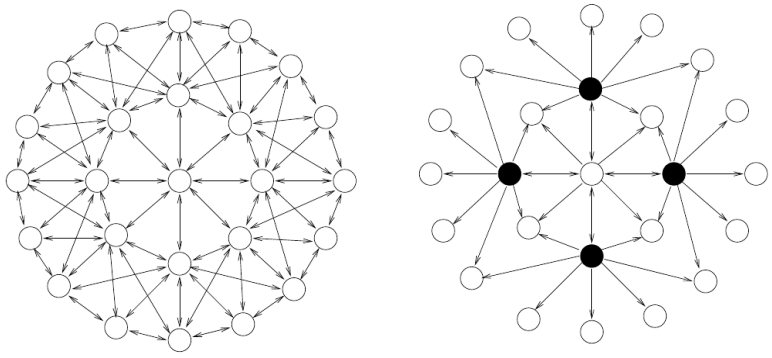
3

Related work

This chapter gives an overview to some of the routing protocols in the current literature. Not all these protocols were specifically designed for Wireless Mesh Networks (WMNs), but we chose protocols from different families to show different interesting approaches that have been developed for routing in wireless networks.

3.1 OLSR

Optimized Link State Routing (OLSR) [20] is an IP-based proactive routing protocol, originally designed for Mobile Ad-hoc NETworks (MANETs) and currently the most used protocol for routing in MANETs and WMNs. OLSR is well suited to large and dense mobile networks. The larger and more dense the network, the more optimization can be achieved. It is also suited for networks where the traffic is random and sporadic between different subsets of nodes.



(a) Without MPR selection [28] (b) With MPR selection [28]

Figure 3.1 TC message flow with and without MPR selection

OLSR uses two types of packets, **Hello** and **Topology Control (TC)**. Hello messages are used to discover the network link status and TC messages to disseminate this information flooding the network. The link information needs to be available to every node in the network, which computes the best route for every possible destination from this information. But flooding in wireless networks is problematic, since it consumes a lot of power and bandwidth. To attenuate this problem OLSR introduced MultiPoint Relays (MPRs). MPRs are a subset of the nodes and they help to reduce the impact of flooding. The rule used to select MPRs is that there must exist a path between each of the 2-hop away neighbors through a selected MPR. To select the subset of MPRs first OLSR spreads Hello messages to detect 2-hop neighbor information, then selects a set of MPRs such that the mentioned rule is satisfied. Figure 3.1 shows how messages are forwarded in the network, with and without MPRs, and how the number of total transmissions of TC packets is drastically reduced using MPRs.

Keeping a synchronized and distributed topology database through the network (as for example OSPF [26] does) is very hard to achieve on mesh networks, so OLSR does not prove that at any moment the topology database is synchronized in the whole network, but only floods the network often enough to keep a sufficiently updated status.

3.2 AODV

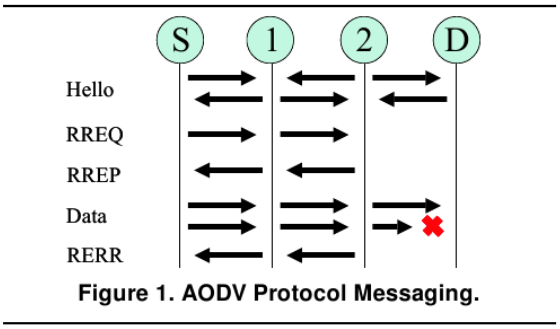


Figure 3.2 AODV protocol message (from [17]), representing a possible data flow between a sender, a destination and two intermediate nodes

Ad-hoc On-demand Distance Vector (AODV) [18] is a reactive routing protocol, as defined in Section 2.4. It was designed for mobile ad hoc networks up to thousands of mobile nodes, to reduce the impact of control traffic and eliminate overhead on data traffic, to improve scalability and performance.

It uses five control message types, as shown in Figure 3.2: **Hello**, **RREQ**, **RREP**, **Data** and **RERR**. The Hello message is a single-hop message broadcast periodically between all the neighbors. If several Hello messages are lost, the link is marked as broken and not used anymore. When a source wants to send data to an unknown destination, it broadcasts a Route Request (RREQ). Every intermediate node that receives the route request creates a route to the source, to be able to send back an answer to the original destination. If the destination is found, a Route Reply (RREP) is sent back as a unicast hop-by-hop packet. In case the source receives

multiple routes, the route with the shortest hop count is selected. If a route is not used for some period, the node removes it from the routing table, since it might have become invalid in the meanwhile. When a broken link is detected, the node sends a Route Error (RERR) back to the source, and all the intermediate nodes remove all the routes that were using that link.

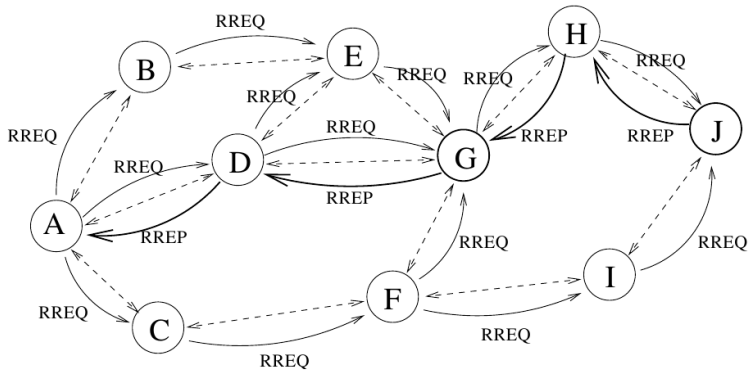


Figure 3.3 Possible route replay given a route request from A to J [28]

Figure 3.3 shows an example of how a route request works: node *A* wants to find a route to node *J*. First a RREQ is broadcast to the all the neighbors (B, C, and D), then when the requests reaches *J*, J sends a RREP packet. The RREP is sent using the same route that has been used to reach J, since D, G, and H cached the route to the sender destination A.

3.3 BVR

Beacon Vector Routing (BVR) [22] is a *distance vector* routing algorithm, based on hop-count as distance metric. It was originally designed for Wireless Sensor Networks (WSNs) networks, since it was necessary to have a *point-to-point* routing protocol to explore new possible application fields. WSNs have different properties and constraints than WMNs, but the same concepts can be applied to both environments.

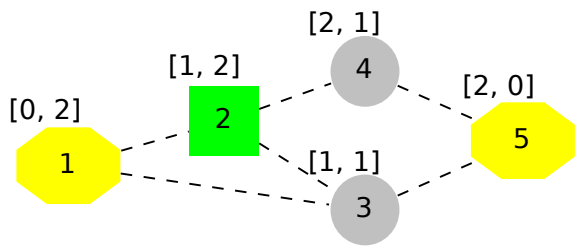


Figure 3.4 Example of coordinate distribution in BVR, where the octagonal nodes represent the landmarks

BVR selects a set of (potentially randomly chosen) landmark nodes to construct virtual coordinates. The virtual coordinate of each node is the vector of hop count distances to the landmark nodes. Figure 3.4 shows how the addressing is done in BVR, with each node's coordinate given as the hop count distance to the two landmark nodes.

Every node periodically sends a beacon packet containing its current coordinate vector and updates its coordinate vector according to the received beacons. In BVR a neighbor is a node from which beacons can be received. But given the unreliable nature of wireless links, not all the potential neighbors are used. To select which neighbors should be used, BVR uses link estimation, whose details are described in the next subsection. Once every node has a BVR coordinate set, the routing protocol uses a distance metric $\delta(p, d)$ to compute the next hop p to reach destination d.

To avoid routing loops, forwarding a packet must always decrease the distance to the final destination. First the sender node identifies a list of suitable neighbors that would decrease the distance to the destination. Then tries to deliver the packet to the first one up to five times. If it fails the next neighbor is tried, until all the suitable neighbors have been tried. If they all fail BVR switches the packet status to *fallback mode*. In fallback mode the packet is delivered directly to the closest landmark to destination. Once the landmark receives the packet in fallback mode, it first tries the normal distance-minimizing routing algorithm, and in case it fails it does a scoped flooding broadcasting the packet n hops away with emphn as the ith component of the destination's address vector. This technique ensures that packet is always delivered to destination, but its use should be minimized since fallback mode involves a much bigger number of transmissions.

BVR Link estimation

Every node in BVR continuously monitors the connection to all its neighbors with beacon packets. BVR has a passive link estimator based on [30], BVR only derives virtual coordinates from and routes packets only to neighbors with high and stable link quality. Each of the nodes in the receiving range can then estimate the packet ratio loss for each of the sources. This technique is useful to estimate the quality of incoming links, but since wireless links are asymmetric, every node also periodically transmits the quality of its incoming links.

3.4 PAD

Probabilistic ADdressing (PAD) [29] is a routing protocol designed to deal efficiently with dynamic communication links in wireless networks. Most of its design ideas are derived from BVR (3.3) and it was originally implemented on top of BVR for TinyOS [11] and TOSSIM.

The basic idea behind PAD is that a node learns from its past locations and computes the probability distribution over its recent locations. PAD is also the protocol on which this work is based on, so we will discuss in more detail in Chapter 4 some of its design choices. Instead of filtering node mobility using a link estimator as BVR does, PAD incorporates the variability into the node addresses. In PAD addresses are composed of a fixed-size history of BVR coordinates. Figure 3.5 clarifies the difference between BVR and PAD addressing schemes. Landmarks are still considered to be 0 hops away from themselves, but the other components of the address are probability distributions.

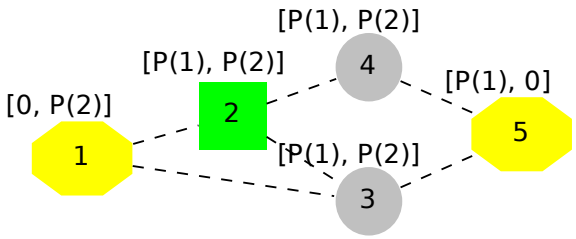


Figure 3.5 Representation of addresses in PAD, with the network used in Figure 3.4, where P_i is the probability distribution of hop count distances from the ith landmark.

The design of the PAD addressing scheme is independent from a specific routing strategy. In PAD routing is done similarly to BVR, greedily choosing the neighbor which minimizes the distance to the destination. The distance function is the sum of the differences in the hop count to the landmarks.

3.5 GPSR

Greedy Perimeter Stateless Routing (GPSR) [25] is a routing protocol that uses *geographic* information to achieve **scalability**. One factor that heavily influences the scalability of a routing protocol is the amount of data that each node needs to store. GPSR is almost *stateless*, since each node only needs to know about its direct neighbors. GPSR is a suitable choice when these three following assumptions are satisfied:

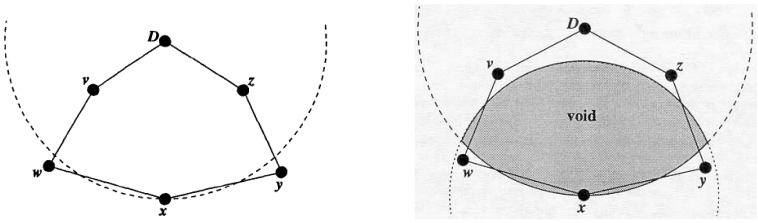
1. Every node can determine its own position, using GPS coordinates or similar techniques.

2. Every node is aware of its neighbor's coordinates, which is achieved by broadcasting the current position to the neighbors.

3. The position of the destination node is known, which requires a distributed location service that maps network addresses to geographic locations.

Routing algorithm uses *greedy forwarding* whenever possible, and *perimeter mode* in case greedy forwarding fails. Every packet is marked by the sender with the location of the destination. Every intermediate node makes a locally optimal choice selecting as next hop the node which is geographically closest to the final destination. Greedy forwarding can fail whenever the current node is geographically closer to the destination than all its neighbors, as shown in Figure 3.6(a). In this case NOGEO switches the forwarding algorithm to perimeter mode.

In Figure 3.6(b) there is a *void* area where there are no neighbors. Perimeter mode tries to go around this void area until it finds a node which is closer to the destination. At that point GPSR continues with greedy routing.



(a) Greedy forwarding failure [25] (b) Void region [25]

Figure 3.6 Two possible critical scenarios during GPSR routing.

3.6 NOGEO

Geographic Routing without Location Information (NOGEO) [27] is a scalable, coordinate-based routing protocol designed to retain the benefits of geographic routing without requiring location information.

NOGEO assigns *virtual coordinates* to each node and then applies greedy geographic routing to them. Virtual coordinates are generated locally from connectivity information available on the node. Since the network can be arbitrarily big, the approach to construct the virtual coordinates has to be scalable. To construct the virtual coordinates NOGEO needs a base set of nodes, called *perimeter nodes* (as in Figure 3.7), which are the nodes that lie on the geographic perimeter of the network. There are three possible scenarios to consider when constructing the virtual coordinates, depending on how much knowledge about the perimeter nodes is available.

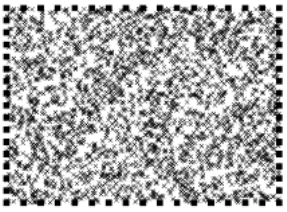


Figure 3.7 Network with 3200 nodes, highlighting the perimeter nodes as squares [27]

- **Perimeter nodes know their location, stage** 1:
  In this first case all non-perimeter nodes can determine their location through an iterative relaxation procedure. Each neighbor relation is represented by a force that pulls the neighbors together. During the relaxation phase of the process, nodes that have a perimeter node among its neighbors "move".

- **Perimeter nodes know their status but not their location, stage** 2:
  In this case an extra step is required, since perimeter nodes do not know their location. So before going to stage 1, perimeter nodes flood the network to compute the hop count distance between all perimeter nodes. Then NOGEO computes the approximate location of the perimeter nodes using a triangulation algorithm given the inter-perimeter distance matrix.

- **No location nor status is known, stage** 3:
  In the case we relax the requirement that perimeter nodes know their status. First NOGEO designates a **bootstrapping node**. This bootstrapping node floods the network with HELLO messages, and all the other nodes determine their distance to this bootstrapping node. To be able to apply the last procedure it is necessary to identify which nodes are the perimeter nodes. NOGEO determines this through the *perimeter node criterion*: "If a node is the farthest away, among all its two-hop neighbors from the first bootstrap node, then the node decides that it is on the perimeter" [27]. Once the perimeter nodes are set then the protocol can pass to stage 2.

Once the virtual coordinates are spread throughout the network, the routing algorithm can select the best possible paths. The implementation given in NOGEO [27] uses a simple algorithm. Every node knows its own coordinate, and the coordinates of its neighbors, which form the *routing table*. Then the packets are routed through the network according to the following rules:

- **Greedy**: The packet is forwarded through the network to the neighbor which is closest to the final destination.
- **Dead-end**: If the packet is not able to make greedy progress and did not reach the destination, it reached a dead-end. As a last resort, the node uses a fallback mode mechanism doing a research with an expanding ring. In case it finds a node closer to destination, the packet goes in greedy forwarding mode again, or is lost when the maximum Time To Live (TTL) is exceeded.

3.7 DART

Dynamic Address RouTing (DART) [21] it is a routing protocol designed to scale efficiently in big networks. The addressing scheme separates node identity from node addresses. The address allocation scheme uses efficiently the address space on topologies of randomly and distributed nodes.

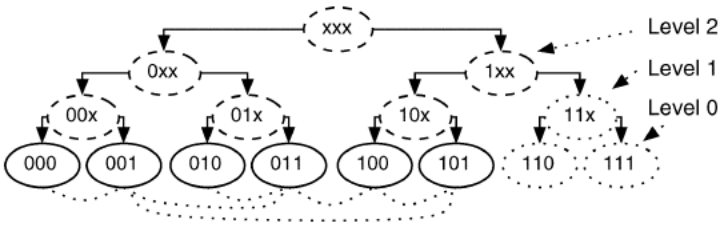


Figure 3.8 Address tree generated ([21])

The *routing address* is dynamic and changes while the node moves to reflect it is location, while the *identifier* is a globally unique which never changes. There are three main functions in the protocol. *Address allocation* maintains one routing address per each network interface. The *routing* function delivers packets to a given

routing address, and *node lookup* is a distributed lookup table mapping each node identifier to the current routing address. When a new node joins the network, it analyzes the periodic routing updates to identify and use an unoccupied address.

Every address is composed by l bits, $a_{l-1}, \ldots, a_o$ and the address space can be seen as a binary *address tree* with $l+1$ levels. A set of nodes from any of the subtrees from figure 3.8 induces a connected subgraph in the network topology. This assertion is very important for scalability, and as a result the longer the shared address prefix is, the shorter is the expected distance between two nodes.

The *identifier of a subtree* are computed as the minimum than the identifiers of all the nodes in that subtree.

4

Design

This chapter is about the design of Mobile Probabilistic ADdressing (MPAD). In section 4.2 we give an overview of the packet types employed by the protocol. Section 4.3 explains the role played by landmark nodes and how the location service is designed. Section 4.4 is about the addressing scheme used, starting from the node identifier to the statistical addressing. The last chapter explains in detail how MPAD does routing.

4.1 Design concepts

The design of MPAD is mainly derived from Probabilistic ADdressing (PAD) [29]. However, PAD was designed originally for Wireless Sensor Networks (WSNs), which have different constraints compared to Wireless Mesh Networks (WMNs). MPAD was designed to solve the problem of routing in WMNs, in particular on WMNs with an infrastructure and one or more mobile nodes, scenario that has not been specifically addressed by other protocols.

MPAD is based on virtual coordinates, as PAD and Beacon Vector Routing (BVR). With mobile nodes one possible issue is that virtual coordinates become too unstable, or do not adapt quickly enough to topology changes. The main goal of MPAD is to compute virtual coordinates such that they are *stable* enough to avoid an excessive protocol overhead while *adaptive* enough to react quickly to changes in the network topology.

Figure 4.1 is an example of how a network is structured in MPAD. A network is composed of a fixed set of landmarks, one or more mobile nodes and optionally other static nodes. Landmarks and static nodes form the infrastructure of the network, and mobile nodes are the clients. As we will see in 4.3 an important assumption for MPAD is that we are able to select a set of landmarks. It is also important that we are able select to for each node a *home landmark* from this previous set. Lastly

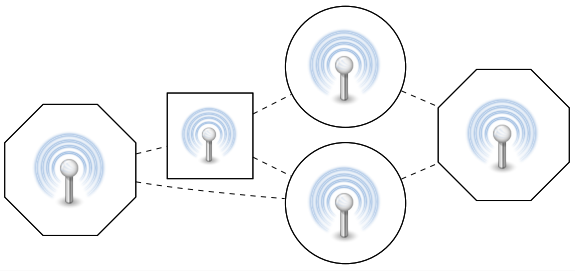


Figure 4.1 Example of an MPAD network, with two landmarks (octagons), one mobile node (square) and two static nodes (circles).

...:	...
3:	Transport layer
2:	MPAD
1:	Data Link layer (MAC)
0:	Physical layer

Figure 4.2 ISO/OSI network layer stack, highlighting the position of MPAD

we assume that landmark nodes are not mobile and that there is a mechanism to replace failing landmarks.

In the ISO/OSI network model MPAD works at level 2.5, as shown in Figure 4.2. Level 2 (the network layer) is responsible to route data over the network and to assign a unique network address to each node of the network. MPAD only accomplish the former, since the addressing scheme does not produce univocally determined node addresses. The virtual coordinates only depends on the distance to the landmarks, thus it is possible that more than one node have the same coordinates. It would be possible to implement another layer on top of MPAD which maps virtual coordinates to unique network addresses, but it is not in the scope of this thesis.

4.2 Packet types

MPAD protocol has the following packet types:

- **Beacon**: Every node periodically sends a beacon packet to signal its presence to the nodes in its radio range. Beacon packets are employed to generate and keep the neighbor tables updated.
- **Coordinate Request**: Whenever a node wants to communicate with another node but does not know its coordinate yet, it sends a coordinate request.
- **Coordinate Answer**: A coordinate answer is sent as an answer to a coordinate request, and contains the mapping between the node identifier and the current coordinate.
- **Address update**: An address update is a packet containing a mapping between a node identifier and the current virtual coordinates, that have to be published on the network.

- **Packet acknowledgment**: Single-hop acknowledgment packet that provides a basic robustness mechanism to MPAD.
- **Data packet**: Communication packet to test MPAD performance. It carries a payload with configurable size which can be used by higher-layer protocols to create real network traffic.
- **Data packet acknowledgment**: Optional multi-hop acknowledgment, acknowledges the data packet and sends the updated coordinates back to the sender. This packet type is optional since multi-hop reliability is not an issue which has to be addressed in the network layer.

4.3 Landmarks

MPAD requires, as first step, selecting a set of landmark nodes, that form the basic infrastructure needed by the protocol. Since virtual coordinates are constructed as a vector of hop count distances to the landmarks, the size of this vector is directly proportional to the number of landmarks. Each node stores a fixed size history of the last σ virtual coordinates, and one or more virtual coordinates are also sent in every packet. Thus network protocol overhead and memory requirement are directly proportional to the number of landmarks. The number of landmarks cannot, however, be too small, for the reasons that we describe in Section 4.3.1.

Once the optimal number of desired landmarks is determined, they must be spread through the network in the best possible way. Positioning landmarks for virtual coordinate based protocols is a complex task and outside the scope of this thesis. See for example [13] for further information about landmark selection.

4.3.1 Location service

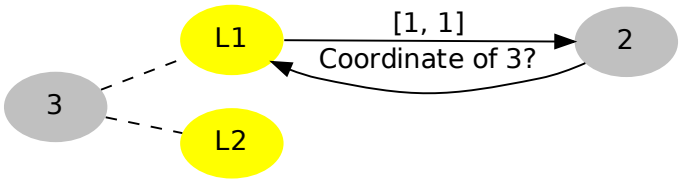


Figure 4.3 Node 2 does not know the coordinate of node 3, so it computes its home landmark and sends a coordinate request.

Landmarks in MPAD have also another important task, they form a distributed location service that stores the associations between node identifiers and virtual coordinates. Every node in the network has an assigned *home landmark*, which is in charge of keeping track of the node current coordinate. Figure 4.3 shows an example coordinate request, where $L1$ is the home landmark of node 3.

In the best scenario nodes are distributed fairly to all the landmarks and the average hop count distance to the home landmarks is minimized, in this way all landmarks are equally critical for the protocol and address updates can be delivered quickly.

One simple algorithm to minimize the distance to the home landmarks consists of selecting the first landmark "seen" as home landmark. In the initial phase the virtual coordinates are not set, but as soon as the landmarks start to send beacon packets all the other nodes will set the corresponding coordinates to the hop count distance to the landmarks. In this simple algorithm we simply pick as home landmark the landmark from which we heard first.

This algorithm would not, however, ensure a fair distribution between the landmarks, and most importantly it would not be possible for the each node to know what is the home landmark of the other nodes. This is an important constraint, because, as seen in Figure 4.3, whenever a node X needs the coordinate of another node Y, it must query for the Y coordinate to Y home landmark.

There are two other possible solutions to this problem. The first is to implement in every node a function which maps node identifiers to landmark node indexes. This function could, however, unfairly distribute the nodes between the landmarks, and also place home landmarks too far away. The second option is to create a map which statically defines the associations, and make it available on every node. Our MPAD prototype implementation looks for a static map if available and falls back on a hash function if it is not available.

4.4 Addressing

In this section we introduce more formally the MPAD addressing scheme. First we talk about the network identifier, then we pass to how the virtual coordinates are computed and in conclusion we introduce the statistical addressing concepts.

4.4.1 Identifier

In MPAD every node needs to be identified univocally in the network. Since MPAD runs on top of the MAC protocol we can assume that there is a mechanism to retrieve the MAC address on each node, and we use this mac address as the node identifier. This identifier has to be sent together with the virtual coordinates, since they do not identify univocally each node in the network alone.

4.4.2 Virtual coordinates

step/node	**1**	**2**	**3**	**4**	**5**
0	0 ∞	∞ ∞	∞ ∞	∞ ∞	∞ 0
1	0 ∞	1 ∞	1 1	∞ 1	∞ 0
2	0 2	1 2	1 1	∞ 1	2 0
3	0 2	1 2	1 1	2 1	2 0

Table 4.4 Coordinate evolution from the network boot to the fully connected status

Table 4.4 shows an example of the evolution of virtual coordinates in the bootstrap phase of the network, whose final status appears in Figure 4.5. At step 0 every virtual

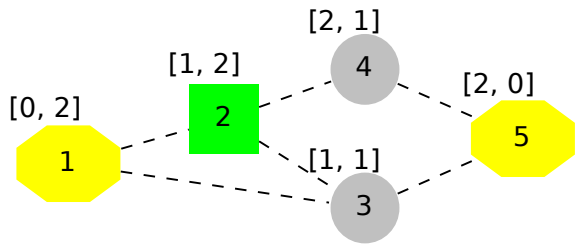


Figure 4.5 Network representation of Table 4.4 at stage 3

coordinate is unknown, except for the distance from a landmark to itself, which is set to 0. At every step each node sends a beacon packet and all its neighbors receive it [1]. So at step 1 node 1 receives a beacon from the first landmark, node 4 receives a beacon from the second landmark, and node 3 receives a beacon from both. At step 3 every node has complete coordinates, and we define this phase as *coordinate stabilization*.

More formally we can define MPAD virtual coordinates of node S as a vector $\vec{c}$ composed by λ (number of landmarks) values, where each value is a distance computed by the function h to the landmark L_i, summarized in Equation 4.1.

$$\vec{c}(S) =< h(S, L_1), \ldots, h(S, L_\lambda) > \tag{4.1}$$

4.4.3 Statistical addresses

Neighbor	Coordinates
1	0, ∞
2	1, 2
3	∞, 0

Minimum coordinates = [1, 1]

Table 4.6 Coordinates received during the last beacon interval

At every beacon interval each node computes its minimum coordinate. This minimum coordinate is composed by the shortest distance to the landmarks, computed over the last beacon interval. Suppose for example that one node S received beacons from its neighbors containing coordinates as given in Table 4.6. S is one hop away from each of the landmarks. The coordinate of node 2 is discarded and the minimum coordinate computed is composed of the smallest hop count distance to the landmarks incremented by one, **[1, 1]**.

MPAD is designed for a mesh network scenario with a few mobile nodes. When a node moves through the network it triggers coordinate updates in every area it traverses. This happens because for the time that the node is in the radio range, it can become a shortcut to one of the landmarks, decreasing the hop count distances to the landmarks. If every update in the minimum coordinate would trigger an address update to the location service the protocol overhead would be too high in case of node mobility. MPAD smoothes down the coordinate changes by keeping an history of these minimum coordinates and doing statistical analysis on it.

[1] Even though in real scenarios also beacon packets might get lost.

Every node stores a history H of fixed size σ of minimum virtual coordinates, defined as:

$$H(S) = \begin{pmatrix} \vec{c_1}(S) \\ \vdots \\ \vec{c_\sigma}(S) \end{pmatrix} = \begin{pmatrix} < h_1(S, L_1), \ldots, h_1(S, L_\lambda) > \\ \vdots \ldots \vdots \\ < h_\sigma(S, L_1), \ldots, h_\sigma(S, L_\lambda) > \end{pmatrix} \tag{4.2}$$

From this history we can compute the frequency distribution of hop count distances for each of the landmarks. This frequency distribution F for each landmark i is a list of couples $(distance, \#occurrences)$, formalized as below:

$$F_i(S) = \{(h_1(S, L_i), f_1), \ldots, ((h_\delta(S, L_i), f_\delta)\} \tag{4.3}$$

To minimize the number of necessary address updates MPAD follows Algorithm 1. The first step to do is to check with the χ^2-test if the current probability distribution of hop counts is significantly statistically different from the last one published on the network. If the distributions are significantly different it means that new coordinates should be published. From now on we will refer to the virtual coordinates published as the *network address* of a given node. The address to publish is computed from the exponential moving average of the history (explained in 2.6.2) rounded to integer coordinates, and it is only published if it differs from the last published address.

```
add_to_history(current_min_coordinate)
if chi_square_test(history, last_published_history)
  then
        avg := compute_moving_average(history)
        if avg ≠ last_avg
          then
                rounded := round(avg)
                if rounded ≠ last_published
                  then
                        send_address_update(rounded)
                        last_published_history := history
                fi
        fi
fi
```

Algorithm 1 Algorithm to decide if it is necessary to publish an updated address

4.5 Routing

Routing in MPAD is strictly correlated to the addressing scheme. The next hop is the neighbor that minimizes the distance to the final destination, computing the distance between two virtual coordinates as the sum of the differences. Given two

virtual coordinates $\bar{p}$ and $\bar{d}$, the distance δ_k is computed as the sum of the differences of the components, formally defined as:

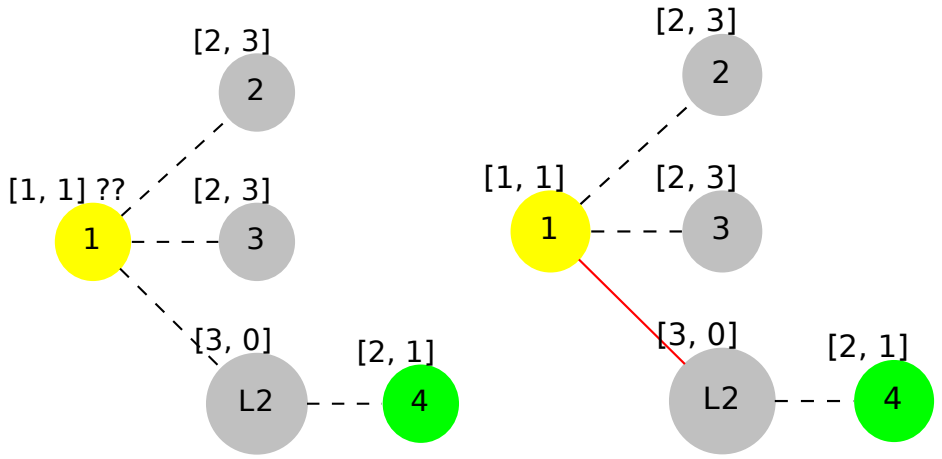
$$\delta_k(\bar{p}, \bar{d}) = \sum_{i \in C_k(d)} |\bar{p}_i - \bar{d}_i| \tag{4.4}$$

Whenever a node wants to send a packet it follows the algorithm shown in 2. If the destination node is in the neighbor list then send the packet directly. If the coordinates of the destination are not known the node sends a coordinate request, and stores the packet in a queue. When the sender node receives a coordinate answer it scans the queue and sends all the packets in queue for the received destination to the next hop. The next hop is computed as the closest neighbor to the destination, with the formula in Equation 4.4.

At this point it can happen that the current node is anyway closer to the destination than the computed next hop. In this case the packet is set into *fallback mode*. A packet in fallback mode is routed to the landmark closest to the final destination (computed from the coordinate) and when it reaches the landmark is broadcasts to the neighbors with a Time To Live (TTL). If the destination node is still close to that landmark there are good chances that the destination will be reached before the TTL expires.

```
if is_neighbor(destination)
  then send_directly(packet, destination)
   else
        if destination_coordinate_unknown()
          then request_coordinate()
           else
                next_hop := closest_neighbor_to(destination)
                if distance_to(destination) < distance(next_hop, destination)
                  then send_in_fallback_mode(packet)
                   else send_directly(packet, next_hop)
                fi
        fi
```

Algorithm 2 Routing algorithm



(a) Node 1 wants to send to node 4, but all its neighbors are farther than 1 itself.

(b) Node 1 sends the packet in fallback mode to landmark 2.

Figure 4.7 Possible situation that triggers the fallback mode algorithm

The network in Figure 4.7 is an example of a possible fallback mode situation. Suppose that node 1 wants to send a packet to node 4, and has the information in the figure about its neighbors coordinates. Using the simple distance metric it turns out that none of the neighbors' would be closer to node 4 than 1. This situation is unrealistic under normal link dynamics, but is perfectly plausible in the presence of mobile nodes. At this point the routing mode switches to fallback mode, and sends the packet to the base landmark, which is the closest landmark to the destination node.

5

Implementation

This chapter describes the implementation of the MPAD prototype. First we introduce the platform on which MPAD was developed, and the architecture of the whole project. Then Section 5.2 introduces some data structures used by MPAD. Chapter 5.3 is about the management of the neighbor tables, and how the reliability mechanism of MPAD is implemented. Lastly Section 5.4 explains how the packet objects are constructed and serialized when they are sent over the network.

5.1 Environment

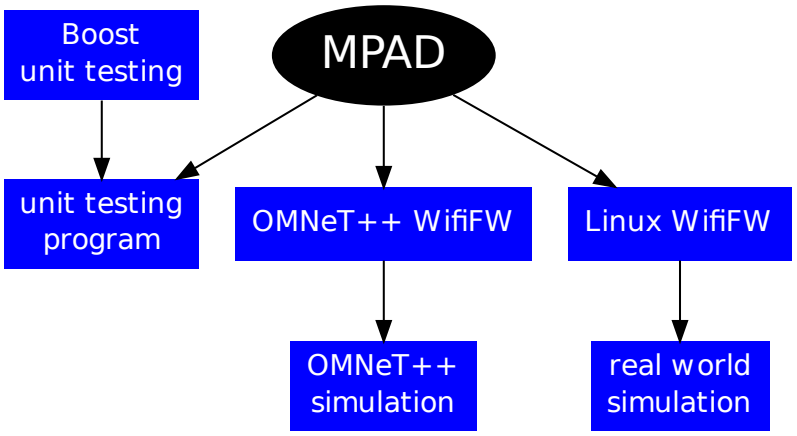


Figure 5.1 Architecture of the simulation and testbed used by MPAD

MPAD is implemented on top of the Wifi Framework (WifiFW) [14], an abstraction layer composed of a Linux and an OMNeT++ part. The OMNeT++-WifiFW is integrated into the INET [5] module, an extension to OMNeT++ that provides mobility patterns and various network protocols. The first task for this thesis was to port the WifiFW, that was written for OMNeT++ 3.x and the INET version of 2007, to OMNeT++ 4.x and the newer version of INET. The Linux-WifiFW part is

an independent Linux program and it has also been modified to uniform according to the changes in the WifiFW interface.

Figure 5.1 gives a schematic representation of how MPAD is implemented: we compile MPAD against the Boost [2] unit testing framework to unit test MPAD internal data structures and functions. Programming with unit testing in mind was useful to write more decoupled and modular code, and was very helpful to detect bugs quickly in later development stages. Then both the Linux-WifiFW and the OMNeT++-WifiFW programs load MPAD as a library compiled with dynamic linking. With this design the three components are completely independent and other routing protocols can be plugged in easily.

The Linux-WifiFW and OMNeT++-WifiFW interfaces provide the following functions:

- **sendToUpperLayer**:
  Send a packet to the upper layer, not used in our prototype.
- **sendMulticastToLowerLayer**:
  Send a multicast packet to the lower layer; to broadcast packets to every node in radio range.
- **sendUnicastToLowerLayer**:
  Send a unicast packet to the lower layer; to send a packet to a neighbor with a given hardware address.
- **addEvent**:
  Schedule an event in the future.
- **getHardwareAddress**:
  Return the hardware address of the node.
- **getSignalStrength**:
  Return the signal strength between the current and a given node, not used in our prototype.
- **getTime**:
  Return the current time. It is very important that all three environments use the same unit measure for time, which in the current implementation is in milliseconds.

Linux-WifiFW and OMNeT++-WifiFW expect our protocol to implement the following functions:

- **initialize**:
  Initialize the status of the routing protocol, for example data structures, variables and timers.
- **handleDataFromUpperLayer**:
  Handle data coming from the upper layer, not currently implemented in our prototype implementation.

- **handleDataFromLowerLayer**:
  Handle raw data coming from the data link layer. In this function the raw packet is converted to a packet object and manipulated accordingly.

- **handleEvent**:
  Handle and event, codified as an integer.

- **finish**:
  Releases allocated memory and optionally dump statistics (by default MPAD does it in real time).

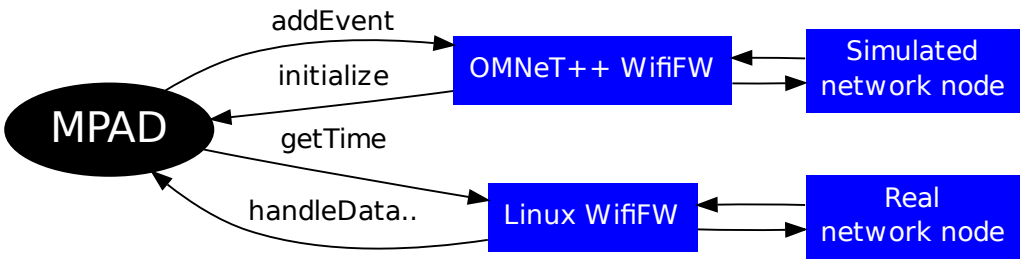


Figure 5.2 Example of how the MPAD library is correlated to OMNeT++ and the Linux-WifiFW.

Figure 5.2 shows an artificial example of how MPAD is correlated with the Linux-WifiFW and the OMNeT++-WifiFW environment. The simulated network node communicates bidirectionally with the OMNeT++-WifiFW environment, and the real network node communicates bidirectionally with the Linux-WifiFW environment. MPAD can call functions like *addEvent* or *getTime* to both environments, and the two environments can for example *initialize* the MPAD status or pass data received from the upper or lower network layer.

5.2 Data structures

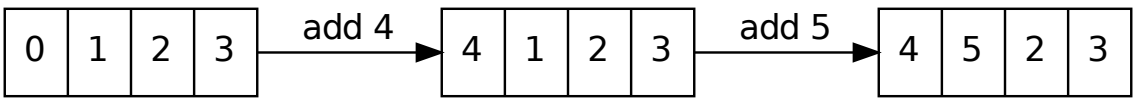


Figure 5.3 Example of a ring buffer data structure, where adding a new element automatically overwrites the oldest element

One important data structure in MPAD is the ring buffer. A ring buffer is essentially a fixed-size buffer which automatically overwrites the oldest position when it is full, as shown in Figure 5.3. The ring buffer data structure is widely used in the implementation, since it is very convenient to create dynamic lists which cannot grow infinitely. The ring buffer class is implemented as a templated class wrapper of the *std::deque* data structure.

```
template<typename T>
class RingBuffer : public Serializable
{
private:
    size_t max_size;

protected:
    std::deque<T> buffer;
...
```

Another heavily used data structure is C++ *std::map*, which allows constant time access to data indexed by a unique key.

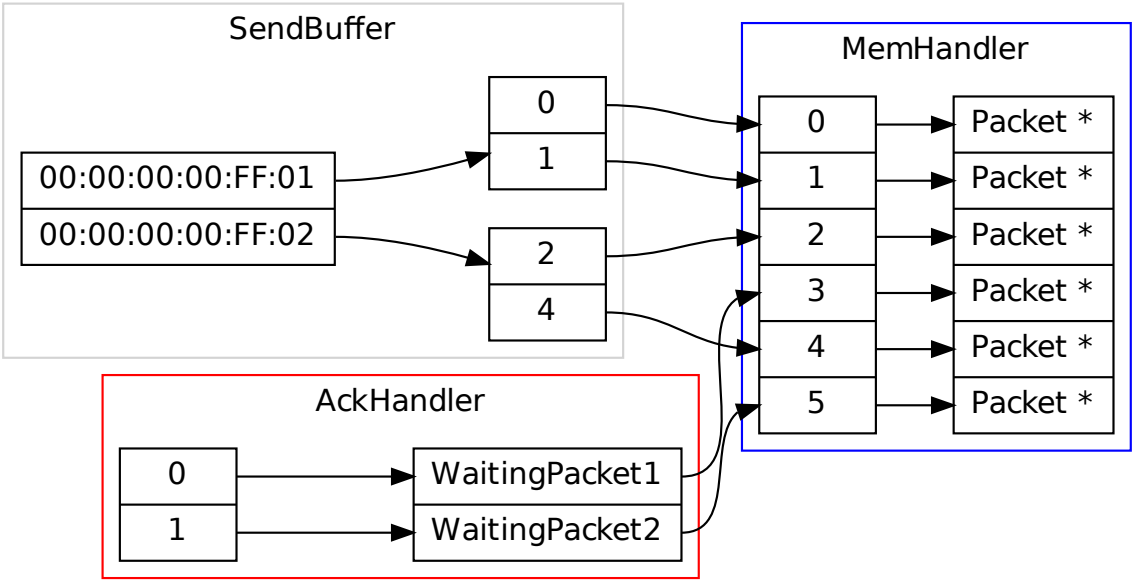


Figure 5.4 Possible situation of the memory handler and the two queues that handle packets in memory

When a node wants to send a packet, it first allocates the memory for it and then creates a new entry in the *MemHandler*. The MemHandler is a mapping between an index and a pointer to a packet allocated in memory. This index is what identifies the packet in every other MPAD function.

If the coordinate of the destination is not yet known, the packet is stored in a *SendBuffer* queue, as a mapping between node identifier and a ring buffer that contains packet indexes. Whenever the sender node receives the coordinate of the receiver, it can send all the packets and free the corresponding ring buffer. In MPAD all packet types except beacons and acknowledgments need to receive an acknowledgment; thus we cannot free the packet memory as long as we have not received an acknowledgment, or the packet is considered lost.

So whenever we send a packet that has to be acknowledged, we add the index of the MemHandler packet to the *AckHandler* structure. This structure keeps track of all the packets that are waiting for an acknowledgment, with some auxiliary fields to implement a basic reliability mechanism, as explained in Section 5.3. It is worth noting that the AckHandler is not implemented as a ring buffer, but can theoretically grow indefinitely. In this case it is safe to use an unbound data structure because

the AckHandler has an upper bound given by the maximum number of timeouts before the packet is declared lost.

In Figure 5.4 we can see an example of how memory and queued packets are handled. the AckHandler structure does not use the index from the MemHandler, but has a mapping mechanism between the two indexes. This is necessary because the AckHandler index has to correspond to the sequential number contained in the acknowledgment packet, and not all the packets that are stored in memory need to be stored also in the acknowledgment queue.

5.3 Neighbor table management

The *Beacon* packet is critical for the management of neighbor tables and the coordinate distribution.

Every beacon packet carries the following information:

- **NodeID**: Node identifier of the sender, a wrapper around the MAC address.
- **Node Coordinate**: Current virtual coordinate, the minimum computed in the last beacon interval, as explained in 4.
- **Neighbors**: List of neighbors seen during the last beacon interval.

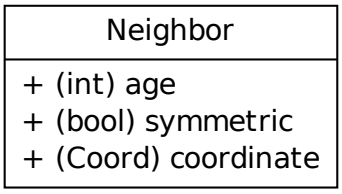


Figure 5.5 Fields in the neighbor class

Each node maintains an updated neighbor table, which is implemented as a map between nodeIDs and a neighbor structure. The neighbor data structure contains the fields shown in Figure 5.5, an integer for the age, a boolean flag for the symmetric status and the last coordinate that was received from the neighbor.

Whenever a node receives a beacon from another node, if the node is already in the neighbor table, it resets the age to 0 and updates the coordinate. If the neighbor is not yet present it creates a new entry setting the age to 0. Every beacon interval each neighbor's age is incremented by one, and if the age is greater than a given threshold the neighbor is removed from the neighbor list.

As seen before, each beacon carries a list of the last neighbors seen. Every time a node X receives a beacon, it extracts the list of the last neighbors seen by the beacon sender Y. Now if X nodeID is present in this list, it means that the this link is symmetric. So for every beacon received from a node Y, X sets the symmetric flag in neighbor Y to true if X was found in the list of last seen neighbors, or to false if it is not found.

One possible problem with this approach is that once the flag is set to true, it will not be set to false if no more beacon packets are received from that neighbor, until eventually the node gets removed completely from the neighbor table. In our implementation it is possible to configure at run-time the symmetric neighbor behavior with the following flags:

- **neighbor_good_strategy**: If set to 0, directly send a packet to a neighbor only if it is marked as symmetric, otherwise send in any case.
- **parent_node_must_be_symmetric**: If set to 1, always try first to use a symmetric neighbor as next hop for a packet directed to a landmark node.
- **route_only_to_symmetric_nodes**: If set to 1, always first try to use a symmetric neighbor as next hop for a packet directed to a non landmark node.

However it is not always possible to select a symmetric neighbor, in that case, the last two options will fall back on selecting the best of the asymmetric neighbors.

During the implementation we experienced that knowing how the lower layer works is sometimes necessary to implement a higher layer protocol. In the first implementation, every node was able to send beacon packets but none of them was receiving any packets, even if they were in radio range. It turned out that every node was sending the beacon packet at exactly the same time, and the channel was always busy, thus not allowing anyone to receive data.

To solve this issue now every node i sends its first beacon to a different time t_first_beacon, as seen in formula 5.1, while the beacon interval remains unchanged. But when the network becomes very big, this distance interval between the beacon event time of each node becomes linearly smaller. We noticed that with big and very connected networks, keeping the beacon interval at 1 second was not yielding good results, because most of the beacon packets were lost. The conclusion is that the beacon interval is a critical parameter, and its setting depends on the network size and average connectivity.

$$t\_first\_beacon_i = beacon\_interval + \frac{(beacon\_interval * i)}{number\_of\_nodes} \tag{5.1}$$

MPAD does not try to ensure that a packet traversing multiple hops always reaches destination, since that is an issue that has to be addressed in the transport layer. It does, however, have a single-hop acknowledgment mechanism, which works as follows: whenever a node sends a packet which needs to be acknowledged, it stores the packet in the AckHandler data structure seen in Section 5.2. After a given interval of time if the acknowledgment is not received, the packet is resent two more times. If the first neighbor fails to send back all acknowledgments before the given timeout, the protocol tries the next two neighbors, and if this also fails the packet is considered lost.

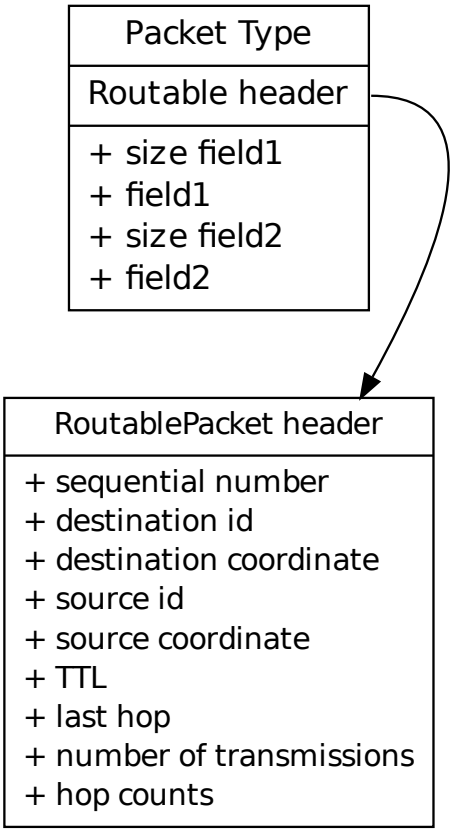


Figure 5.6 Generic packet structure after serialization

5.4 Packet structure

The Linux-WifiFW and OMNeT++-WifiFW environments expect to get a buffer of char to send to the lower layer. This means that higher level data structures like vectors or maps have to be serialized to a plain buffer of chars, which has, however, to be reconverted to a packet object on the receiver side. To make this process as flexible as possible, we created a *Stream* class, which implements some basic conversion mechanism from different data types to chars stream.

Each object that needs to be serialized has to implement the *Serializable* interface, and in particular the *toStream* function. Every packet is composed of a vector of pointers to serializable objects. This vector has to be set in the function *setFields*, which is called by the packet constructor. Every packet is then able to transform its content into a Stream object with *toStream*, or directly to an array of characters with *toBuffer*.

We implemented a *FieldHeader* class, which takes the object and creates a new stream, composed of the object size and the serialized object. Figure 5.6 shows how a packet is serialized, before every field we store the size of the next field. In this way on the receiver side it is easy to reconstruct the original packet, without requiring fixed sizes.

To summarize the Packet class has the following members:

```
std::vector<Serializable *> fields;
void toBuffer(char *, int *) const;
Stream toStream() const;
```

```
virtual void setFields() = 0;
```

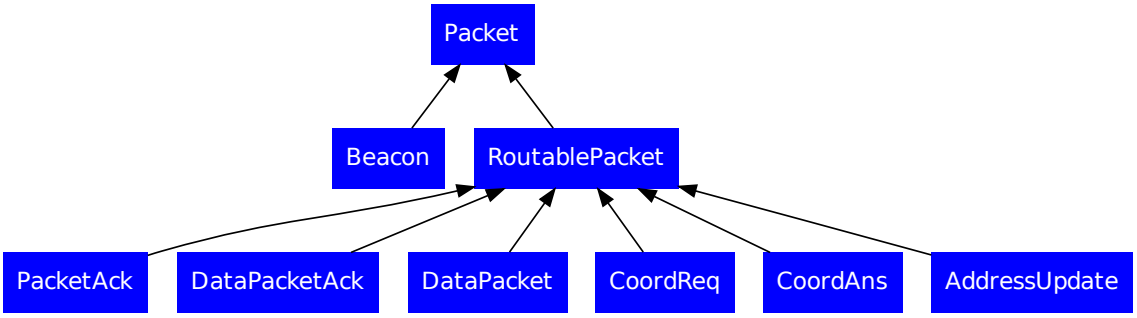


Figure 5.7 Packet type class hierarchy

As we can see from the hierarchy in Figure 5.7 every packet type except Beacon is also a subclass of *RoutablePacket*, since beacons are inherently one-hop packets, which never needs to be routed via multiple hops. Another difference is that beacon packets are sent as *multicast*, while all the other packet types are sent as *unicast*.

Every RoutablePacket has the following fields in its header (summarized in Figure 5.6):

- **Sequential number**: Sequential number of the packet.
- **Destination ID**: Node ID of the destination.
- **Destination coordinate**: Coordinate of the destination.
- **Source ID**: Node ID of the sender.
- **Source coordinate**: Coordinate of the sender. We choose to always send the last computed minimum coordinate, to give the receiver a current coordinate for replies.
- **TTL**: Time To Live of the packet, only used when the packet enters fallback mode.
- **Transmission count**: Total number of times that the packet has been transmitted over the network, incremented by 1 every time it is transmitted.
- **Hop count**: Number of hops traversed to reach the final destination, incremented by 1 at every intermediate node.

Hop count and number of transmission fields are only used to analyze the performance of MPAD, they are not strictly necessary for the protocol. Moreover not all packets need to have information about the sender, since they do not expect a reply. So a simple optimization would be to create another header class removing the *source ID* and *source Coordinate* fields, and use this header for *Coordinate Answer*, *DataPacketAck*, *PacketAck* and *Address Update*.

6

Evaluation

In this chapter we present the results of the MPAD evaluation. First we introduce the two environments on which the protocol was evaluated. Then Section 6.2 explains how the logging and the analysis of the results was done. Section 6.3 introduces the network and the mobility patterns evaluated on it.

6.1 Evaluation environments

MPAD has been evaluated on OMNeT++ and on a testbed [32] with Linux routers. The OMNeT++ simulation runs a generated *omnetpp.ini* configuration file and uses the mobility patterns provided by INET.

The testbed consists of 51 identical mesh routers, all equipped with two IEEE 802.11 b/g wireless network interfaces and two tri-band 5dBi omnidirectional antennas, spread over three floors and two different buildings. The Linux driver employed is Madwifi [6] and to test our prototype we only need one interface in ad hoc mode. This is the output of *iwconfig* on the network card:

```
Rate:0 kb/s   Tx-Power=16 dBm   Sensitivity=1/1
Retry:off   RTS thr:off   Fragment thr:off
Power Management:off
Link Quality=21/70  Signal level=-74 dBm  Noise level=-95 dBm
Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

There are a few important differences between the testbed and the simulation environment:

- **Global state**: The testbed is truly and completely distributed, so the code cannot rely on global variables. In OMNeT++ global state is also highly discouraged but it proved to be useful during testing.

- **Boot process**: Every node in the OMNeT++ simulation is booted at the same time. On the testbed, instead, there might be a significant offset due to the fact that the nodes are booted remotely with a network command. This is not an issue for the overall behavior, but, since the timer of each node starts when the node is booted, merging the results will generate temporal distortions.

- **Node identifier settings**: With OMNeT++ we set up hardware addresses for the nodes manually. With the testbed this is not feasible and thus MPAD needs to load the hardware addresses from a static configuration file previously generated.

- **Non-determinism**: Every run on the real testbed produces different results, even if the parameters have not been modified. The OMNeT++ simulation on the contrary, given the same parameters, will always produce the same results.

6.2 Analysis

We could not use OMNeT++ facilities to analyze the results of our simulation, since (as shown in Figure 5.1) we have the WifiFW as intermediate layer. So we implemented the whole analysis framework from scratch in Python [10], NetworkX [7], matplotlib [23] and a few accessory shell scripts. Every event which might be helpful for evaluation or debugging is listed in the *LOG_EVENTS* enum, here the first entries:

```
enum LOG_EVENTS {
  ADDRESS_AVERAGE_CHANGED,
  ADDRESS_MIN_CHANGED,
  ADDRESS_UPDATED_TO_LANDMARK,
 ...
```

Each of these events can have 0 or 1 arguments, where the argument is the string representation of a simple type, a map or a vector. This log event is then parsed and interpreted by the python analysis code.

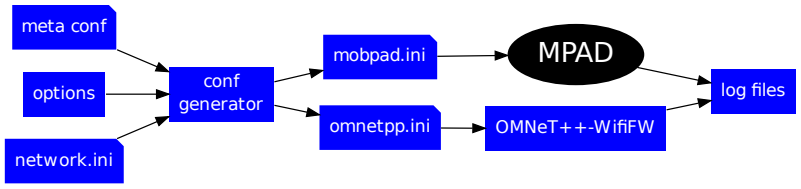


Figure 6.1 Flow diagram of the OMNeT++ simulation

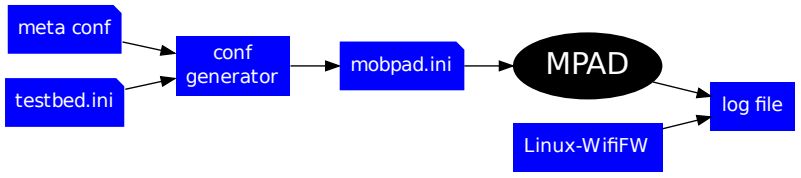


Figure 6.2 Flow diagram for the testbed simulation

Figure 6.1 and 6.2 show how we generate log files. One important difference is that in the OMNeT++ simulation we directly generate all the log files for the nodes, while in the testbed every node only generates one log file, and the analysis program merges them together.

6.3 Network and mobility patterns

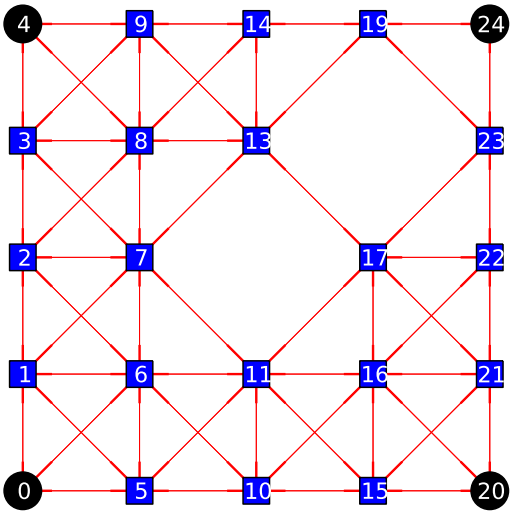


Figure 6.3 Grid network evaluated

We evaluate MPAD on the network in Figure 6.3. This network is generated from a grid where two nodes have been removed to make it less regular. The edges seen in the graph are set from the geometric distance between the nodes, but they realistically demonstrate the neighbor relations seen during the simulations. This network has 23 nodes and 4 landmark nodes situated in the corners.

We start to send real network traffic into the network when the coordinates of all the nodes are *stable* (as explained in Subsection 4.4.2). In the OMNeT++ simulation it is possible to detect automatically when the coordinates are stable, and we use this information to start sending the data. This is not, however, possible in the testbed so we wait for a minimum high enough number of beacon intervals before starting the communication.

On the OMNeT++ network we analyze the following mobility scenarios:

- **static**: No mobility, base-line for mobility scenarios.
- **close**: The destination node moves around another node, in the vicinity of its home landmark.
- **around**: The destination node moves around the whole grid network. In this scenario the mobile node never becomes a shortcut for other nodes.
- **inside**: The destination node moves inside the network, passing through the radio range of many other nodes. This scenario is the worst possible for the addressing scheme, since while the node traverses the network it might potentially trigger address updates in the whole network.

PARAMETER	VALUE
beacon interval	2
is neighbor	10
burst length	3
packets per burst	40
payload size	800
mobility speed	6

Table 6.4 Default values of a few important parameters

6.4 Delivery rate

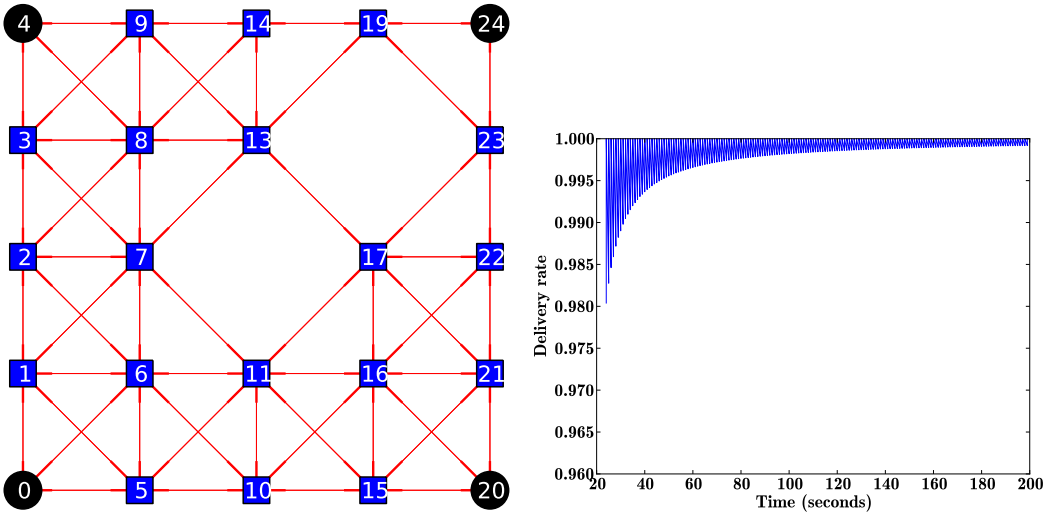
Table 6.4 shows some important default parameters used in the simulations. The beacon interval is set to 2 seconds, and every three beacon intervals the protocol starts a new data burst. At every data burst the sender node queries the location service for a new coordinate and sends 40 packets in each burst. Each data packet is composed of a sequence number, 800 bytes of payload and the *RoutablePacket* header, as seen in Figure 5.6. The *is neighbor* parameter influences the neighbor table management. It means that once a node is added to the neighbor table it will be kept as neighbor unless 10 consecutive beacon packets from this neighbor are lost. In all the mobility patterns analyzed the mobile node moves according to the INET turtle mobility pattern with speed set to 6.

To evaluate the results we collected for each scenario the following results:

- **Delivery rate**: Ratio between sent and received packets $\frac{\text{\#data packets received}}{\text{\#data packets sent}}$
- **Data sent**: Number of data packets sent.
- **Hop count**: Average number of hops required to reach the destination.
- **Hop count over transmissions**: Average of the ratio $\frac{\text{\#transmission}}{\text{\#hop count}}$, where the number of transmissions and hop counts is computed as seen in Section 5.4.
- **Duplicate data packets**: Number of duplicate data packets received.
- **Fallback mode**: Number of data packets that were received in fallback mode.
- **Address updates**: Average number of address updates that were sent over the network.
- **Coordinates changes**: Average of the number of times that the nodes changed their minimum virtual coordinate.

The spikes seen in the following result plots depend on how the analysis of the results is done. To produce the graph at every *DATA_RECEIVED* or *DATA_SENT* event, we recompute the delivery rate metric with the current timeline. To generate an interesting plot we removed the first 100 values, which are not relevant for the general behavior, and we always plot the results in the 0.75 to 1.00 range.

6.4.1 Static



(a) Static scenario, sending data from node 0 to node 21.

(b) Delivery rate with "static" mobility pattern

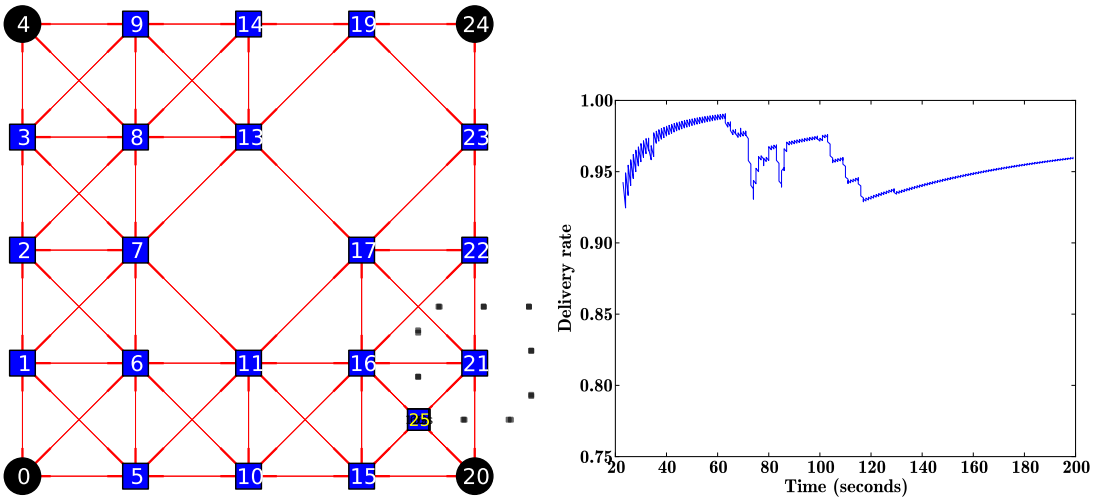
Figure 6.5 Static scenario, in this case the scale of the plot has been changed to see more clearly the oscillations.

METRIC	VALUE
Delivery rate	1
Data packet sent	1224
Average hop counts	4
Hop counts/Transmissions	1
Received in fallback mode	0
Duplicate data packets	0
Average address updates	3.48
Coordinates changes	25.39

Table 6.6 Results obtained in the static mobility pattern

In this first mobility pattern (represented in Figure 6.5(a)), there is no mobility. All the packets are received correctly, and the small spikes seen in Plot 6.5(b) are due to the phenomena previously explained in Section 6.4.

6.4.2 Close



(a) Destination node 25 moves around node 21. (b) Delivery rate with the "close" mobility pattern

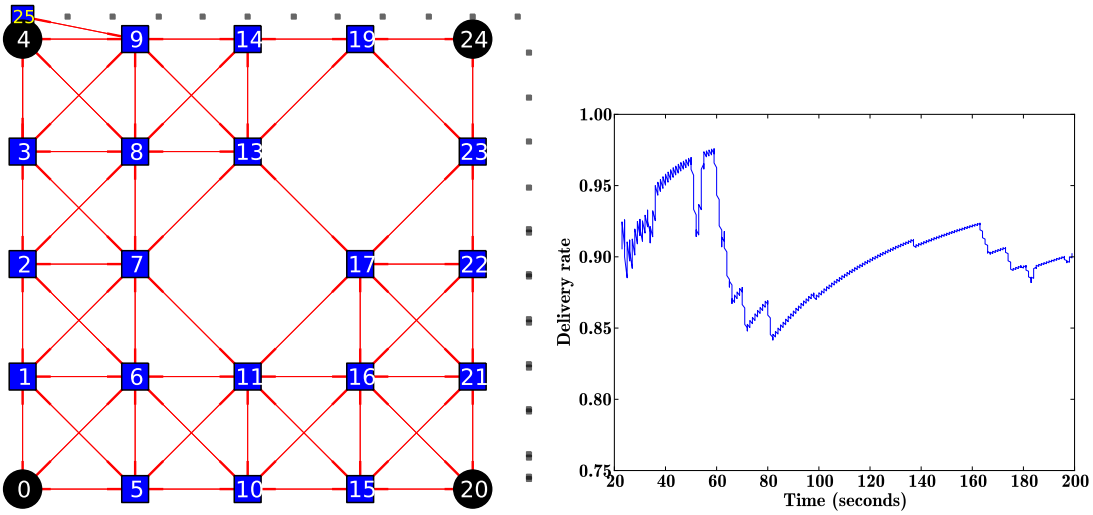
Figure 6.7 Close mobility pattern

METRIC	VALUE
Delivery rate	0.96
Data packet sent	1226
Average hop counts	3.96
Hop counts/Transmissions	1.00
Received in fallback mode	23
Duplicate data packets	29
Average address updates	8.58
Coordinates changes	34.96

Table 6.8 Results obtained in the close mobility pattern

In the close mobility pattern destination node 25 moves around node 21. It is worth noting that now the average hop count is $3,96$, instead of the 4 computed in the static mobility pattern. This means that (probably when node 25 was in the closest position) MPAD managed to deliver very few packets in 3 instead of 4 hops. Moreover there are now 23 packets which were received in fallback mode, which corresponds to the two upward spikes seen shortly before and after the 80 second mark in Figure 6.7(b).

6.4.3 Around



(a) Destination node 25 moves around the whole network.

(b) Delivery rate with the "around" mobility pattern

Figure 6.9 Around mobility pattern

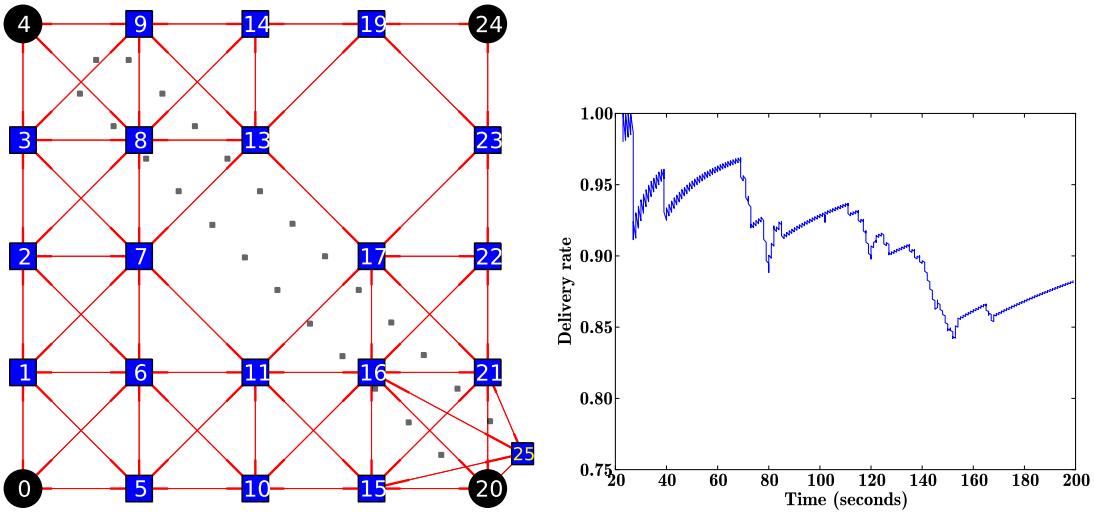
METRIC	VALUE
Delivery rate	0.90
Data packet sent	1227
Average hop counts	4.95
Hop counts/Transmissions	1.00
Received in fallback mode	46
Duplicate data packets	49
Average address updates	12.50
Coordinates changes	49.46

Table 6.10 Results obtained in the around mobility pattern

In the around mobility pattern destination node 25 starts at the top left corner and moves around the network. After reaching the bottom right corner it starts to go back on the same way, until the simulation time reaches 200 seconds. As we can see from the plot 6.9(b) after 60 seconds there is a downward spike in the delivery rate, which corresponds to the maximum hop count distance been reached.

In this scenario the average number of coordinate changes is much higher than in the close mobility pattern, because node mobility influences the coordinates of more nodes. The number of address updates that are really sent, however, is only about 25% of the coordinates changes.

6.4.4 Inside



(a) Destination node 25 traverses the network diagonally.

(b) Delivery rate with the "inside" mobility pattern

Figure 6.11 Inside mobility pattern

METRIC	VALUE
Delivery rate	0.88
Data packet sent	1226
Average hop counts	3.52
Hop counts/Transmissions	1.01
Received in fallback mode	19
Duplicate data packets	23
Average address updates	12.42
Coordinates changes	44.54

Table 6.12 Results obtained in the inside mobility pattern

In the inside mobility pattern destination node 25 starts close to landmark node 20, and diagonally traverses the network. Compared to the around mobility the coordinates changes drop by 10%, but the average address updates remain the same. This means that the coordinate changes generated in the network by this mobility pattern are more significant, and they are more likely to trigger address updates.

6.4.5 Testbed

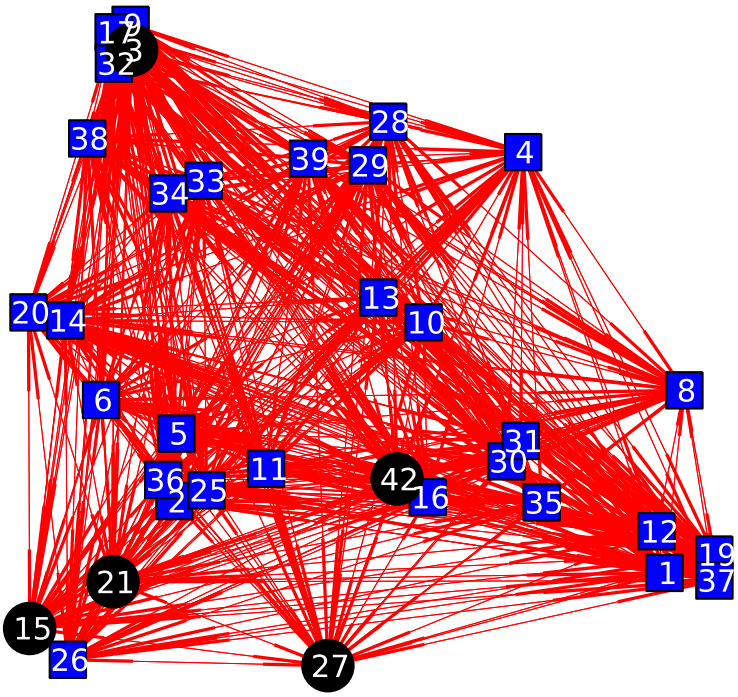


Figure 6.13 Graph representing the testbed nodes. This graph does not represent the real positions of nodes but only their neighbor relations.

METRIC	AVERAGE	STD
Delivery rate	0.67	0.13
Data packet sent	520.22	179.05
Average hop counts	2.62	0.43
Hop counts/Transmissions	1.28	0.18
Received in fallback mode	39.89	22.93
Duplicate data packets	195.56	178.30
Average address updates	21.73	4.80
Coordinates changes	63.76	6.75

Table 6.14 Results from the testbed sending data from node 11 to 35, generated from 10 different runs

In Table 6.14 we can see the results on the testbed, sending data from node 11 to node 35, which are physically located in two different buildings. The delivery rate is as expected, lower than in the OMNeT++ simulation, and the variability of the results is also very high. These results can be, however, greatly improved in the future tuning other configuration parameters.

7

Conclusion

In this thesis we presented Mobile Probabilistic ADdressing (MPAD), a routing protocol based on virtual coordinates for Wireless Mesh Networks (WMNs). The design of MPAD is based on Probabilistic ADdressing (PAD), a multi-hop point-to-point routing protocol for Wireless Sensor Networks (WSNs). The focus of this thesis was to show the feasibility of statistical addressing approaches in a different network domain, specifically IEEE 802.11-based networks.

WMNs have different constraints from WSNs, for example bandwidth, mobility, power requirements, computing capabilities and operating system support. A core challenge was therefore to adapt the concepts to WMNs, while keeping the simplicity of the original design.

While PAD introduced statistical addressing, address lookup itself was not specified. To test the complete approach, we proposed a lookup scheme based on assigning a home landmark to each node, which is responsible for tracking its current address.

The implementation runs on top of the Wifi Framework (WifiFW), which allows us to reuse code between the Linux and the OMNeT++ implementation. In the course of this thesis we also had to port WifiFW to the current versions of OMNeT++ and INET.

To analyze routing performance under mobility, we constructed three mobility scenarios: inside, close and around, see Section 6.4. Worst case delivery rate was still above 85%. In general we found to be feasible in WMNs. As MPAD is therefore applicable to both WSNs and WMNs, we conclude that developing this idea further can yield improved network performance.

7.1 Future work

In the evaluation chapter we have seen how MPAD behaves and how statistical addressing performs on some simple test cases. There are, however, many parameters which have not been tuned and analyzed (listed in Appendix A.1). Moreover, it might be possible to improve MPAD performance by using a more sophisticated distance metric, which would take more advantage of the statistical addressing than the current metric.

During the evaluation we also experienced how hard it is to analyze results from routing protocol simulations, and it is important to have some automatic analysis and visualization tools to get a better feeling what is really happening in the network. The WifiFW was a very good approach to produce code that runs on a real as well as in a simulated environment, but it completely lacks analysis and visualization functions.

Finally, to really show that MPAD is a suitable choice, an important task left for the future is to compare its performance against other currently adopted routing protocols, such as for example OLSR or AODV.

A

Appendix

Acronyms

AODV Ad-hoc On-demand Distance Vector. 14

BVR Beacon Vector Routing. v, 15–17, 21

DART Dynamic Address RouTing. 19

EMA Exponential Moving Average. 5, 11, 12

FIR Finite Impulse Response. 11

GPSR Greedy Perimeter Stateless Routing. 17

MANET Mobile Ad-hoc NETwork. 13

MPAD Mobile Probabilistic ADdressing. v, 2, 3, 21–26, 29–32, 34, 37–39, 42, 47–49

MPR MultiPoint Relay. 14

NOGEO Geographic Routing without Location Information. 18, 19

OLSR Optimized Link State Routing. 13, 14

PAD Probabilistic ADdressing. 2, 16, 17, 21, 47

PRR Packet Reception Rate. 9, 10

SSID Service Set IDentifier. 6

TC Topology Control. 14

TTL Time To Live. 19, 27

WifiFW Wifi Framework. 29, 30, 38, 47, 48

WLAN Wireless Local Area Network. 5, 6

WMN Wireless Mesh Network. v, 1, 5–8, 13, 15, 21, 47

WSN Wireless Sensor Network. 15, 21, 47

A.1 Runtime MPAD parameters

Here below there is the list of all the parameters that can be tweaked to change MPAD behavior.

```
# 0: disabled, 1: enabled
singlehop_ack_mode = 1

# This variable influences how many coordinate requests are sent over
# the network when the coordinate is unknown
# 0: disabled, 1: every burst, 2: every packet,
# 3: smart (send an ack when over a threshold)
multihop_ack_mode = 1

# if n > 0, send +- n background packets to a random destination
background_packets_per_beacon_interval = 0

# beacon interval length, in seconds
beacon_interval = 1

# p-value min parameter to declare the change significant
chi_square_max_p_value = 0.05

# length of the data burst, in beacon intervals
data_burst_time = 3

# how many packets to send out at every burst
data_packets_for_every_burst = 40

# size in bytes of the payload to send in the data packets
data_packet_payload_size = 800

# size of the ringbuffer that holds the ack for data packet received
data_received_acks_size = 30

# length of the probability distribution to use
distribution_size = 30

# 0: disabled, > 0 drop the packet if the hop count became too big
drop_packet_after_many_hops = 10

# alpha used in the moving average function, > 0.5 to give more
# weight to last results
exponential_alpha = 0.6

# 0: disabled, 1: enabled
fallback_mode = 1

# size of the queue of sequential numbers for fallback packets
fallback_seqs_queue_size = 30

# beacons to miss to remove a neighbor from the neighbor list
is_neighbor = 10

# 0: false, just neighbor is enough, 1: true, must be symmetric
neighbor_good_must_be_symmetric = 0

# 0: false, 1: true
```

```
parent_node_must_be_symmetric = 0

# 0: disabled, 1: enabled
realtime_logging_enabled = 0

# number of best neighbors to try
resend_to_next_neighbor = 3

# 0: false, 1: true
route_only_to_symmetric_nodes = 0

# length of the packets to keep in queue while waiting for the
# coordinate
send_queue_length = 10

# if > 0 means that the sending should be started after n beacon
# intervals otherwise it will just wait for the whole network to be
# connected. Must be a fixed time on the testbed since there is no
# way to read the global connected status of the network
start_send_data_fixed_time = 0

# how many timeout intervals before trying the next neighbors
timeouts_to_expire_packet = 3

# how many broadcast from fallback mode can be tried before declaring
# the packet lost
# 0: compute automatically from the coordinate,
# > 0: fixed number of hops
tot_broadcast_needed = 3
```

Bibliography

[1] Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.). http://www.open-mesh.org/.

[2] Boost, portable C++ libraries. http://www.boost.org/.

[3] Fluidmesh. http://www.fluidmesh.com/en/press-room/press-releases/15-largest-wireless-mesh-cctv-system-in-europe-installed-in-como-italy-using-vi html.

[4] Freifunk. http://start.freifunk.net/.

[5] INET,communication networks simulation package for OMNeT++. http://inet.omnetpp.org/.

[6] MadWifi, WLAN Linux driver. http://madwifi-project.org/.

[7] NetworkX, Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. http://networkx.lanl.gov/.

[8] OLPC mesh network. http://wiki.laptop.org/go/Mesh_Network_Details.

[9] Portsmouth intelligent transport system. http://www.portsmouth.gov.uk.

[10] Python programming language. http://python.org/.

[11] TinyOS, operating system designed for low-power wireless devices. http://www.tinyos.net/.

[12] IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)* (12 2007), C1 –1184.

[13] Baskakov, S. Landmarks selection algorithm for virtual coordinates routing. In *Proceedings of the Third International Conference on Wireless Algorithms, Systems, and Applications* (Berlin, Heidelberg, 2008), WASA '08, Springer-Verlag, pp. 17–28. http://dx.doi.org/10.1007/978-3-540-88582-5_5.

[14] Bertsch, J. A framework for developing and deploying wireless network protocols. Master's thesis, RWTH Aachen, 2007.

[15] Bicket, J., Aguayo, D., Biswas, S., and Morris, R. Architecture and evaluation of an unplanned 802.11b mesh network, 2005.

[16] Bruno, R., Conti, M., and Gregori, E. Mesh networks: commodity multihop ad hoc networks. *Communications Magazine, IEEE 43*, 3 (2005), 123 – 131.

[17] Chakeres, I. D., and Belding-Royer, E. M. Aodv routing protocol implementation design. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC (ICDCSW'04) - Volume 7* (Washington, DC, USA, 2004), ICDCSW '04, IEEE Computer Society, pp. 698–703. http://portal.acm.org/citation.cfm?id=977399.977938.

[18] Charles E. Perkins, E. M. R. Ad hoc on-demand distance vector routing.

[19] Chroboczek, J. The babel routing protocol. Tech. rep., RFC, 2011. http://www.rfc-editor.org/rfc/rfc6126.txt.

[20] Clausen, T., and Jaquet, P. RFC 3626 - optimized link state routing protocol (OLSR). Tech. rep., 2003. http://tools.ietf.org/html/rfc3626.

[21] Eriksson, J., Faloutsos, M., and Krishnamurthy, S. V. Dart: dynamic address routing for scalable ad hoc and mesh networks. *IEEE/ACM Trans. Netw. 15* (February 2007), 119–132. http://dx.doi.org/10.1109/TNET.2006.890092.

[22] Fonseca, R., Ratsanamy, S., Zhao, J., Ee, C., Culler, D., Shenker, S., , and Stoica, I. Beacon vector routing: Scalable point-to-point in wireless sensornets. *NSDI* (2005).

[23] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering 9*, 3 (May-Jun 2007), 90–95.

[24] Johnson, D. B., Maltz, D. A., and Broch, J. Ad hoc networking. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001, ch. DSR: the dynamic source routing protocol for multihop wireless ad hoc networks, pp. 139–172. http://portal.acm.org/citation.cfm?id=374547.374552.

[25] Karp, B., and Kung, H. T. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking* (New York, NY, USA, 2000), MobiCom '00, ACM, pp. 243–254. http://doi.acm.org/10.1145/345910.345953.

[26] Moy, J., and Moy, G. J. The OSPF specification. Tech. rep., RFC, 1998. http://www.ietf.org/rfc/rfc2328.txt.

[27] Rao, A., Ratnasamy, S., Papadimitriou, C., Shenker, S., and Stoica, I. Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking* (New York, NY, USA, 2003), MobiCom '03, ACM, pp. 96–108. http://doi.acm.org/10.1145/938985.938996.

[28] Tønnensen, A. Implementing and extending the optimized link state routing protocol. Master's thesis, UniK University Graduate Center University of Oslo, 2004.

[29] Vaegs, T., Alizai, M. H., and Wehrle, K. Probabilistic addressing in wireless networks.

[30] Woo, A., Tong, T., and Culler, D. Taming the underlying challenges of reliable multihop routing in sensor networks. In *In SenSys* (2003), ACM Press, pp. 14–27.

[31] Yang, C.-C., and Tseng, L.-P. Fisheye zone routing protocol: A multi-level zone routing protocol for mobile ad hoc networks. *Comput. Commun. 30* (January 2007), 261–268. `http://portal.acm.org/citation.cfm?id=1223501.1223573`.

[32] Zimmermann, A., Güneş, M., Wenig, M., Ritzerfeld, J., and Meis, U. A hybrid testbed for wireless mesh networks. In *Proceedings of the 1st International Workshop on Operator-Assisted (Wireless Mesh) Community Networks (OpComm'06)* (2006), IEEE Communications Society Press.

List of Figures

1.1 Example of an MPAD network, with two landmarks (octagons), one mobile node (square) and two static nodes (circles). 2

2.1 Example of two wireless nodes, where B is in the radio range of A but not vice versa 6

2.2 Example of an infrastructure wireless network, with an access point and 4 heterogeneous clients connected to it 6

2.3 Example of an ad hoc mode network, with nodes connected directly to each other 7

2.4 Example of a mesh network, where one of the nodes is directly connected to an external network and provides Internet connection to the other nodes 7

3.1 TC message flow with and without MPR selection 13

3.2 AODV protocol message (from [17]), representing a possible data flow between a sender, a destination and two intermediate nodes 14

3.3 Possible route replay given a route request from A to J [28] 15

3.4 Example of coordinate distribution in BVR, where the octagonal nodes represent the landmarks 15

3.5 Representation of addresses in PAD, with the network used in Figure 3.4, where P_i is the probability distribution of hop count distances from the ith landmark. 17

3.6 Two possible critical scenarios during GPSR routing. 18

3.7 Network with 3200 nodes, highlighting the perimeter nodes as squares [27] 18

3.8 Address tree generated ([21]) 19

4.1 Example of an MPAD network, with two landmarks (octagons), one mobile node (square) and two static nodes (circles). 22

4.2 ISO/OSI network layer stack, highlighting the position of MPAD . . . 22

4.3 Node 2 does not know the coordinate of node 3, so it computes its home landmark and sends a coordinate request. 23

4.5 Network representation of Table 4.4 at stage 3 25

4.7 Possible situation that triggers the fallback mode algorithm 28

5.1 Architecture of the simulation and testbed used by MPAD 29

5.2 Example of how the MPAD library is correlated to OMNeT++ and the Linux-WifiFW. 31

5.3 Example of a ring buffer data structure, where adding a new element automatically overwrites the oldest element 31

5.4 Possible situation of the memory handler and the two queues that handle packets in memory 32

5.5 Fields in the neighbor class 33

5.6 Generic packet structure after serialization 35

5.7 Packet type class hierarchy 36

6.1 Flow diagram of the OMNeT++ simulation 38

6.2 Flow diagram for the testbed simulation 38

6.3 Grid network evaluated 39

6.5 Static scenario, in this case the scale of the plot has been changed to see more clearly the oscillations. 41

6.7 Close mobility pattern 42

6.9 Around mobility pattern 43

6.11 Inside mobility pattern 44

6.13 Graph representing the testbed nodes. This graph does not represent the real positions of nodes but only their neighbor relations. 45

List of Tables

2.5 Number of occurrences for each face of the die 11

4.4 Coordinate evolution from the network boot to the fully connected status 24

4.6 Coordinates received during the last beacon interval 25

6.4 Default values of a few important parameters 40

6.6 Results obtained in the static mobility pattern 41

6.8 Results obtained in the close mobility pattern 42

6.10 Results obtained in the around mobility pattern 43

6.12 Results obtained in the inside mobility pattern 44

6.14 Results from the testbed sending data from node 11 to 35, generated from 10 different runs 45