Tobias Hardes

Performance analysis and simulation of a Freifunk Mesh network in Paderborn using B.A.T.M.A.N Advanced

Masterarbeit im Fach Informatik

14. Oktober 2015

Please cite as:
Tobias Hardes, “Performance analysis and simulation of a Freifunk Mesh network in Paderborn using B.A.T.M.A.N Advanced,” Masterarbeit, University of Paderborn, Department of Computer Science, October 2015.

University of Paderborn
Department of Computer Science
Distributed Embedded Systems (CCS Group)

Fürstenallee 11 · 33102 Paderborn · Germany

http://www.ccs-labs.org/

Performance analysis and simulation of a Freifunk Mesh network in Paderborn using B.A.T.M.A.N Advanced

Masterarbeit im Fach Informatik

vorgelegt von

Tobias Hardes

geb. am 22. März 1990
in Paderborn

angefertigt in der Fachgruppe

Distributed Embedded Systems
(CCS Group)

Institut für Informatik
Universität Paderborn

Betreuer: **Christoph Sommer**
Falko Dressler

Abgabe der Arbeit: **14. Oktober 2015**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Tobias Hardes)
Paderborn, 14. Oktober 2015

Abstract

Free networks are established and maintained by more and more people on a voluntary basis to be independent from commercial providers. This also enables an uncensored communication and leads to a network owned by a community. *Freifunk* is such a project and it is based on wireless mesh networks. All participants provide an own wifi router (node) for transferring data between different devices. The routing protocol used in the Freifunk network of Paderborn is called Better Approach To Mobile Ad-hoc Networking advanced (B.A.T.M.A.N. IV or Batman-adv). It uses so called Originator Messages (OGM) to announce the existence of a node and to measure the quality of different links in the network.

The Freifunk network in Paderborn consists of about 800 nodes that periodically broadcast OGMs. This leads to a substantial base load of $\sim$25 GByte per month and node. Instead of performing modifications in the real network, I create a simulation model of the current network structure and implement version IV of the B.A.T.M.A.N. routing protocol. The model is validated by using data from the real network. Based on this model I propose and evaluate five different opportunities to improve the base load without impairing other functionality of the network. The simulation results show that a combination of two improvements can lead to a reduction up to 90 %.

Kurzfassung

Freie Netzwerke werden von immer mehr Menschen ehrenamtlich aufgebaut und gewartet, um unabhängig von kommerziellen Anbietern zu sein. Dies ermöglicht auch einen unzensierten Zugriff auf Informationen und schafft ein Netzwerk, das im Besitz einer Gemeinschaft von verschiedenen Menschen ist. *Freifunk* ist ein solches Projekt, welches aus Mesh-Netzwerken basiert. Alle Teilnehmer stellen einen eigenen WLAN-Router (Knoten) für den Datentransfer zwischen den Teilnehmern zur Verfügung. Das Routing-Protokoll, welches für das Freifunk-Netz in Paderborn genutzt wird, heißt Better Approach To Mobile Ad-hoc Networking advanced (B.A.T.M.A.N. adv oder Batman-adv). Es verwendet sogenannte Originator Messages (OGM), um Informationen über die eigene Existenz im Netzwerk zu verbreiten und um die Qualität verschiedener Pfade zu bewerten.

Das Netzwerk in Paderborn besteht aus ca. 800 Freifunk-Knoten, welche periodisch OGMs versenden. Dies führt zu einer permanenten Grundlast, die nicht zu vernachlässigen ist. Anstelle Änderungen direkt im realen Netzwerk durchzuführen, habe ich eine Simulation des aktuellen Netzwerks erstellt und die Version IV des B.A.T.M.A.N. Routing-Protokolls implementiert. Das Simulationsmodell wird mit Messungen aus der realen Welt validiert. Basierend auf dem korrekten Modell vergleiche und validiere ich fünf Möglichkeiten, um die Grundlast zu verringern, ohne das andere Funktionalitäten des Netzwerks negativ beeinflusst werden. Die Simulation zeigt, dass eine Kombination von zwei Optimierungen zu einer Reduktion von bis zu 90 % führen kann.

Contents

Chapter 1

Introduction

Wireless networks are widely used today and it is an important technology for communication and today's social life. The Internet is the most important network that is used to connect an enormous amount of devices in one big network. Access to this network is provided by an Internet Service Provider (ISP). But there are also other approaches, which are non-commercial and free to use without entering an agreement with an ISP.

Freifunk[1] is a community-driven and non-commercial open initiative with the vision to distribute free networks, to democratize the communication infrastructure and to promote social structures locally in a geographic area such as a city. Freifunk acts as an alternative to commercial network providers and is based on wireless networks. These free wireless networks offer a public space where free content can be distributed and shared without registration and tracking of users. Since no commercial provider and no central entity is needed to talk to each other, the network is managed and established by using mesh networks.

In a wireless network with managed infrastructure, routing information is usually provided by a central entity called *access point*. Opposed to this in an ad-hoc network there is no central entity that provides routing to other networks and between nodes [1]. In this case, mesh technology can be used to collect information to establish, access and to route within the network. Wireless mesh networks are often presented [1, 2] as a next-generation technology for flexible networking to cover local areas such as buildings or even larger areas using a wireless network. Today it is often used to ensure preliminary communication in disasters, for inexpensive network coverage in buildings and in areas without cabling. According to [1] and [3], the primary advantage of mesh networks is the ability to quickly setup cheap and large networks where no existing infrastructure is available. In a mesh network it is the node's task to maintain and control the network. As a consequence, the

[1] `http://www.freifunk.net`

node does not just send and receive data, it also has to relay traffic from other nodes like a router in a network with managed infrastructure. According to [4] and [5], routing protocols can be classified into *proactive routing* and *reactive routing protocols*, depending on their update mechanism. Proactive protocol messages are sent via broadcast to announce their own presence and to build routes before any data must be transmitted by using the network. In a reactive protocol a route is requested on demand. This prevents the continuous topology update if no data is to be transmitted. In both variants, each node makes routing decisions solely based on its local information. There are also several projects developing new routing protocols for wireless mesh networks. An example for such a protocol is described in the IEEE 802.11s standard that is introduced in [6]. However, relating to [1] the IEEE 802.11s protocol is not widely deployed and other protocols are more frequently used like the *Optimized Link State Routing (OLSR)*, *bmx6*, *Babel* or the *Better Approach To Mobile Ad-hoc Networking advanced (B.A.T.M.A.N. or Batman)*. These protocols are often developed by a community. Some of the biggest communities are Freifunk in Germany, and Guifi.Net in Catalonia. Today the network, that is maintained by all Freifunk communities, consists of approximately 18.000 nodes in about 170 independent networks and Guifi.net runs about 46.000 nodes. Those protocols are also compared at the Wireless Battle mesh event[2]. This is an event that aims to test the performance of different routing protocols for ad-hoc networks and it takes place once per year. The protocols are tested in different topologies and scenarios using different metrics like the round trip time or delay measurements. The results of all protocols are compared to each other and documented on the event's homepage . There are additional issues in a mesh network that need to be considered when a new network is established [1, 2]. Since there are mobile nodes in the network like smartphones, often some kind of *roaming* functionality is needed, which means that a connection doesn't get lost if nodes travels within the network. Furthermore, there is an *overhead* to establish routes, to exchange routing information and to discover new nodes in the network. And like in every wireless network the bandwidth and capacity is limited due to one communication medium or frequency that represents a bottleneck. All protocols are operating in a different way, but in general the fundamental operating principle is to minimize this overhead.

1.1 Freifunk

Freifunk is a non-commercial and open initiative to support free radio networks. Beside this goal, the vision of Freifunk is also to democratize the communication

[2] `http://battlemesh.org/`

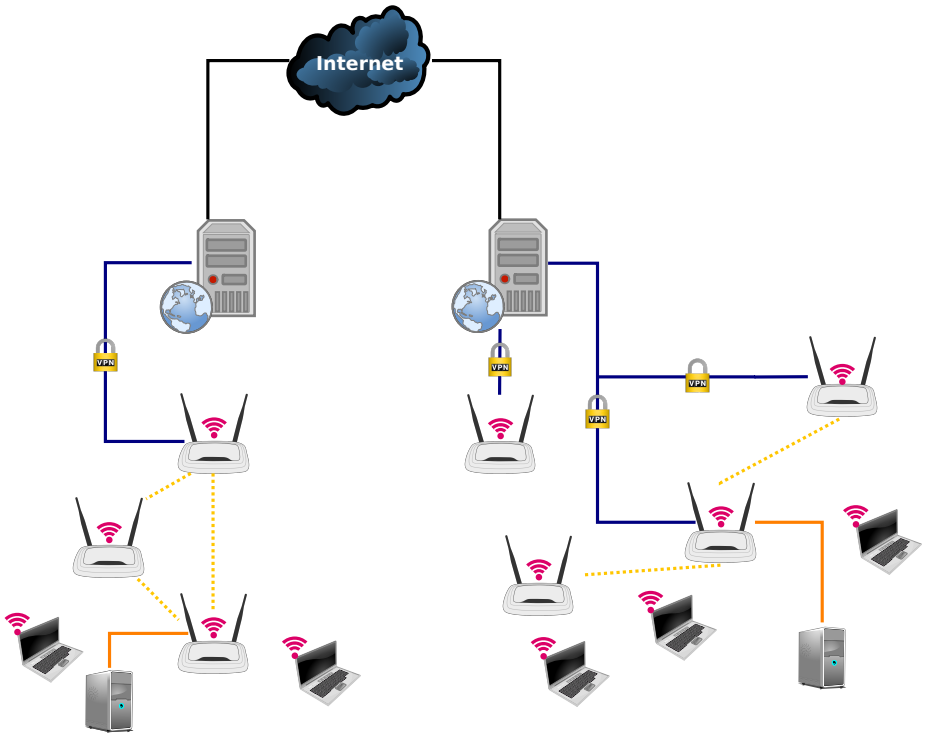


Figure 1.1 – Example mesh network. There are several gateway nodes in the network that provide access to another network like the Internet. Mesh nodes can be connected to each other by using the wireless channel or by using some kind of backbone infrastructure. Non mesh clients are usually connected by using the wireless channel, but it is also possible to use a wired connection, too

infrastructure and to promote social structures locally.

A Freifunk community is a local group of interested participants, developing tools for mesh networks, including an adjusted firmware for the router hardware of the community and *OpenWrt* projects, routing protocols such as OLSR or Batman. Other topics are tools like maps for the network, scanning tools and much more. Recently, there were also efforts to develop "open hardware" like the *Mesh Potato* for villagetelco [7][3]. The network is usually set up and maintained by a local community that is responsible for a geographical area like a city as Paderborn. The network is created by a high number of so called Freifunk routers, which are usually common wifi access points running a special firmware to enable mesh connections with other devices in the network. With several of those small connections a large and decentralized mesh-network is created. This network can be used with any device that supports the 802.11 standard, like a typical smartphone or a computer. Those client devices are not aware of the mesh network, as they are using a usual

[3] `http://villagetelco.org/mesh-potato/`

infrastructure-based wifi network. A Freifunk router can be deployed everywhere. Usually, individuals or local business owners deploy such devices at home or in their businesses, but there are also bigger installations deployed using special hardware for outdoor use.

A Freifunk network needs some central servers to implement important services like DHCP or DNS. Those servers are called *gateways*. Usually, there is a number of services provided within the network. Often those are various games[4], chat platforms like Jabber or video chats[5]. Typically those services are not reachable using the Internet, as those services are explicitly provided for the Freifunk network, using a special top-level domain such as *.ffpb* for Paderborn or *.ffhh* for Hamburg. Generally, a connection to the Internet is still required by users of the network. Because of legal restrictions called *Störerhaftung*, it is not possible to share the Internet connection of the person directly, who runs the Freifunk router. A definition and an explanation of this term can be found in [8]. The *Störerhaftung* is a German legislation, and it is not known in other countries. It means that the subscriber of the Internet connection is liable for any commitment of an offense by using this Internet connection. To circumvent this problem, the gateway nodes are used again and thus every Freifunk router is registered at every gateway within the network. All traffic, that must be routed to the Internet, is send to such a gateway to hide the routers ip address. Due to this, the owner of the router gets invisible to the target host and the gateway acts like a proxy server.

Building a mesh network that covers a complete city and is controlled by a community is nearly impossible to build up as the number of required devices becomes very high and not everybody wishes to run and finance a Freifunk device. Due to this, usually there are gaps between routers so that the wifi signal can not be received at the next device. Here the gateway nodes are needed again. A gateway is a central entity the Freifunk router is connected to. So the Freifunk router needs to be registered at every gateway to establish a VPN connection using the Fast and Secure Tunneling Daemon (fastd). As mentioned above, those connections are established by using an Internet connection that is provided by the individual of the Freifunk router. The router is connected to the gateway infrastructure and the complete traffic that needs to be routed to another network, like the Internet, is sent to the gateways nodes. Thus the individual's identity is not visible to the public, as it is hidden by one of the gateways.

[4] `http:/zockerbude.ffpb`
[5] `http://chat.ffhh/`

1.2 Related work

Several bachelor and master theses, like [1, 9–12] or [13], deal with version IV of Batman (Batman Advanced or Batman-adv) or with version III (Batman or batmand). However, there is no real documentation of the Batman-adv algorithm and no scientific investigation in terms of correctness or performance. There is an IETF draft [14] for the old version III of Batman, but not for version IV which is the version used in Paderborn today. The only official documentation, that is available, is a public wiki[6] on the projects homepage `www.open-mesh.org`. Moreover, the wiki gives just a rough overview about the concepts and details about configurations for the network. It is remarkable, that a lot of research projects reference the open mesh wiki to argue about details for the Batman algorithm, as no other publications are available. Due to the simplified information in the wiki, this thesis is mostly based on the source code that is used by a Freifunk router [15] and on the official Batman-adv git [26] on open-mesh.org. The source code in [15] is a special fork of the Batman code from the open-mesh git repository [26], available in the firmware repository for Freifunk devices. As the firmware developers applied special patches to the original code, this fork is maintained by the firmware developers as well. Therefore, the Batman code used on Freifunk devices is not the same as in the open mesh git.

Several research projects examine in the comparison of OLSR and different versions of Batman, but there are no studies on mesh networks in a whole city. There are also no publications about simulations using version IV, but there are a few references simulating the outdated version III. However, those simulations are performed using ns-2 or ns-3 like in [9]. In [9], a first version of Batman III was implemented in ns-3. The main task was to invent and implement a modification in order to support identification and authentication using X.509 certificates. The initial implementation of Batman was verified by using debug information from a small real-life network. More in-depth information about the validation techniques are not provided. The *packet delivery ratio*, the *number of Originator Messages (OGMs)* and the *packet delay* were used as metrics. However, the modifications to support X.509 certificates had no big effect on the used routing metrics, and the new version of Batman performed similar to the real implementation.

The author in [13] compares version III and version IV of Batman by implementing both protocols in the OPNET Network Modeler. The author used several metrics to compare both algorithms. Those metrics are *packet delivery ratio*, *average end-to-end hop count* and the *generated overhead* in several scenarios. Both algorithms have been tested in different scenarios and with different challenges, like a handover scenario or with mobile nodes. The results show that the characteristics of both pro-

[6] `http://www.open-mesh.org/projects/batman-adv`

tocols are highly dependent on the sliding window size. Version III performs better in a mobile environment if the sliding window is small, as a small window leads to a fast detection of a new path. A large sliding window is very well suited in static networks, as established paths become more attractive. For version IV, depending on the window size, the algorithm can give a more or less accurate average evaluation, where more accuracy leads to a higher latency of detecting topology changes.

There are some studies which compare the version III or version IV of Batman with other mesh routing protocols from both layer 2 and layer 3. A lot of publications compare Batman III and OLSR or testing Batman in different real world environments, like in [16], [17] or [18]. In [16], Barolli et al. compare OLSR and Batman III in a network of four computers and one gateway. There are three scenarios where in scenario one all nodes have a fixed position. In scenario two, one node is mobile and travels within the network. In scenario three, the number of mobile nodes is set to two. Both algorithms are compared by using the *throughput*, *round trip time*, *jitter* and the *packet loss* as metrics. The experiments were performed with packets of 512 kB and a flow of 122 packets per second using TCP and UDP data. Based on the experimental results it turned out, that Batman has a slightly higher throughput than OLSR when using UDP traffic. However, the difference is so small that it can be neglected. If one mobile node is used, the UDP throughput of both algorithms is declining slightly, where Batman still behaves slightly better than OLSR. By using TCP, the throughput drops about 50 % after the third hop. According to the authors, this happens due to the overhead when using TCP transmissions. This effect is even more pronounced when a bigger window is used for the OLSR protocol. When a second mobile node is added, the throughput for TCP and UDP traffic decrease even more. Again, the TCP throughput is better when the window for OLSR is fixed to a lower size. However, all the experiments were performed in an experimental environment using a floor in a university building. This leads to traffic interference and other unknown operation conditions.

In [17], Kulla et al. analyze the behavior of OLSR and the Batman III in a stairs environment by using the *throughput*, *delay* and *packet loss* as a metric. As in [16], the experiments are performed by using five nodes. All nodes are distributed within five floors where one node covers one floor. In the first scenario all nodes are stationary, and in the second scenario the node from the fifth floor moves to the first floor. The throughput decreases with the number of hops in both protocols as in [16], but overall the throughput of OLSR is higher if more than three hops are used. The delay and the packet loss increases after two hops in the mobile scenario and after three hops in the stationary scenario.

The publication of Seither et al. [18] compares Batman-adv, Babel and OLSR. This work is quite interesting, as those protocols are compared at the Wireless Battle mesh event. This is an event that aims to test the performance of different routing

protocols for ad-hoc networks[7], and it takes place once a year. The protocols are compared by using the *throughput*, *latency*, *jitter*, *bandwidth* and *packet delivery ratio* as metrics. The experiments are performed using an indoor test environment in an university building. The authors observed that the throughput, latency and the packet delivery ratio of Babel and Batman-adv are similar for all different hop counts. OLSR has an inferior performance. As an overall result, Babel performs slightly better in terms of bandwidth and throughput, and Batman-adv has a lower latency. However, those results are only valid for the given scenario, which might be affected by unknown environment conditions.

It should be noted, that all publications mentioned in this chapter achieve slightly different results when comparing Batman, Batman-adv or OLSR. This might be due to different environments and uncontrolled experimental conditions like interference on the wireless channel or the structure of buildings and so on. None of these publication give details about the exact environment and due to this it is hard to draw a meaningful conclusion. Furthermore, there is no work yet, that handles a real world scenario with Batman-adv. All publications mentioned above are using a special network that is dedicated to the planned experiment with unknown environment conditions. According to the related work above, this thesis is the first publication that uses a model of a network from real world in a simulation with a configurable and controlled environment.

[7] `http://battlemesh.org/`

Chapter 2

B.A.T.M.A.N. IV

Batman was invented as an alternative to the OLSR protocol in 2006 [19] and uses proactive routing [12] to build up a routing table inside every node of the network.

OLSR with Link Quality (LQ) and the fisheye-algorithm is a very popular and open source routing protocol, and even today it is still frequently used by Freifunk communities, like in Berlin. OLSR was invented in 2003 and is specified in the RFC 3626 [20]. Since the RFC has been published, several improvements and a more powerful modular architecture have been developed, which result in OLSR version 2 [21]. The main functionality and basic algorithms are still the same. There were also some publications like [16] or [22] that focus on the performance of OLSR. OLSR is a link-state protocol. It has the typical functions *neighbor discovery*, *link state advertising* for links and *the shortest path calculation* [23]. According to [11], OLSR is a proactive and table driven routing protocol, which means each node stores a table with routes to other nodes in the network. The main idea is to exchange so called Topology Control (TC) packets. These are used to propagate information about the topology and to implement the link advertising. Those TC packets are transmitted after a nodes neighbor table has been changed to propagate this change. Based on the knowledge that is gathered by those TC packets, every node performs a shortest path calculation that is used to perform the next-hop decision to forward data packets. The link to neighbors is checked by sending so called *HELLO*-Messages [17] using the Neighborhood Discovery Protocol (NHDP) [13]. The content of a *HELLO*-Message is basically the address of the node and a list of the direct neighbors. In OLSR a node is called neighbor if and only if that node can be reached with a bidirectional link [16]. Thus, based on the NHDP, the check for bidirectional links is performed and unidirectional links are excluded from the routing. OLSR is implemented as a daemon[8] and it is operating on OSI Layer 3. In early implementations there were many problems with frequently changing topologies, as routes often changed

[8] A background process running in the users virtual memory

and became invalid. Furthermore, loops occurred and made the network hard to use. To fix some of the problems, the link's qualities were included by using the Expected Transmission Count (ETX) metric [24], and to solve the issue with looping routes, the TC packets were sent more often. ETX is the estimated number of transmissions that are required to successfully transmit a packet to a neighbor. Sending TC packets more often would lead to a high overhead. To counter this problem, the TC messages were just sent as a broadcast with a TTL of three, so only to a three-hop neighborhood [1]. This extension is also know as the *Link Quality Fish Eye mechanism* [1,25] or multipoint relay (MPR) approach [13].

Batman operates on OSI layer 3 as well and is implemented as a user space daemon which is called the B.A.T.M.A.N. daemon (batmand). Due to performance issues in Batman, the development of Batman advanced (Batman-adv or Batman IV) started in 2007. In addition to moving the daemon to the kernel, the protocol also routes on OSI layer 2 to avoid expensive copying of packets from and to the user space [1]. This also means that the implementation of Batman-adv encapsulates all traffic and acts like a big network switch between all nodes. The advantage of this is, among other things, a better support for network protocols, as Batman-adv can be used with other protocols than IPv4 and also makes other features easier to integrate. The complete announcement of routes in the network is based on message type called the Originator Message (OGM). On a regular basis every node broadcasts OGMs to inform its link-local neighbors about its own existence. Neighbors, which are receiving the OGMs, are rebroadcasting them according to specific Batman rules. The principle of Batman is similar to Destination-Sequenced Distance Vector routing (DSDV). Frames, which are important for the routing, are marked with unique sequence numbers to check the age of routing information and to prevent from creating routing loops. This sequence number is independently applied to each entry in the routing table. The most important fields of an OGM are the originator address, the TTL, sequence number (SQ) and the Transmit Link Quality (TQ), but there are various packet types to solve different tasks in the Batman algorithm.

In related work about Batman and Batman-adv some terms differ from the ones used in other literature about computer networks and routing. In the following, I give definitions for terms I am using in this document to explain the Batman-adv algorithm. The following definitions are brought together from [1], [10] and [19] :

Node or originator: A participating device in the routing, which acts on OSI-Layer 2 with a unique MAC-Address and multiple interfaces. A node sends OGMs and is addressable in the network.

Gateway: A node providing access to another network, e.g. the Internet. In addition it offers services like DNS, DHCP or special services for the Freifunk network.

Client: A device connected to a node. This device is unaware of the mesh routing and could be a smartphone, a computer or another device like this.

Router: A node that is chosen to be the next best hop towards another node. Its duty is to route a packet towards this destination.

Translation Table (TT) Every client MAC address that is recognized through the mesh interface will be stored in a node local table called *local Translation Table* which contains all the clients the node is currently serving. This table contains the information a node has to spread among the network in order to make its clients reachable. All client information of other nodes is stored in the *global Translation Table* and this table is used as a lookup table for the unicast routing of client traffic.

Originator table: The routing table used in Batman.

The Freifunk network can be modeled as a graph [27] $G = (V, E)$ where V is a finite set of nodes and E is a set of links between two nodes with $E = \{(u, v) \in E : u, v \in V\}$. For each $s \in V$ there is a subset of nodes $V' \subset V$ that are reachable within one hop. Elements in V' are called *one-hop neighbors*. Let $s \in V$ be an originator and $t \in V$ be the destination. Then, a message from s to t is always transmitted using a link $(s, t) \in E$, if $t \in V'$ so t is one of the *one-hop neighbors*. If $t \notin V'$ the message is transmitted using a *multi-hop route* using a link (s, i) and a route $[i, t]$ with $i \in V'$and $(s, i) \in E$. Then the route $[i, t]$ is a set of hops through a subnet S with

$$S = \big(V \setminus s, E \setminus \{(s, i) : i \in V'\}\big). \tag{2.1}$$

A node in the graph represents a real device in the network. Each device runs a special firmware called *Gluon*, that has been developed for the Freifunk use case. Gluon is based on OpenWrt and the main branch is developed primarily by the Freifunk community in Lübeck. The first release was in April 2014 and the project is hosted on GitHub [28].

Since the firmware is based on OpenWrt, it is possible to use almost every wifi router that is initially supported by OpenWrt. The Gluon firmware has to be adapted for every router and so the community of Paderborn decided to support about 30 - 40 devices[9]. The WR841N(D) from TP-Link is offered in a public meeting every week. So most of the nodes in Paderborn are TP-Link WR841N(D) routers as shown in Figure 2.1. This device is able to operate in 2.4 GHz and it has 802.11g and 802.11n support. It has 4MB of flash memory, but the maximum size of the firmware image is limited to 3.6 MB.

[9] `http://firmware.paderborn.freifunk.net/`

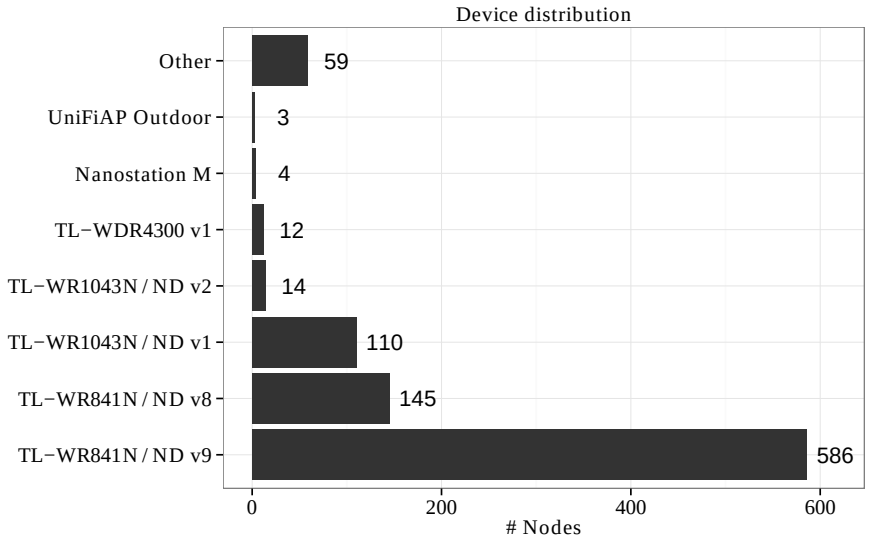


Figure 2.1 – Distribution of devices in Paderborn (July 15th, 2015)

Each community that plans to use the Gluon firmware can just clone the repository and apply its individual configuration for the own community. Here it is possible to define subnetworks, configuration for 2.4 GHz and 5 GHz wifi networks, information for the mesh network and of course information about the existing gateways in the mesh network. The wifi section is the most interesting part. Here the firmware developer is able to set the SSID and BSSID for the client and the mesh network. So there are two visible wifi networks where the mesh network runs in an ad-hoc mode and the client networks runs as an infrastructure mode. A minimal configuration for the 2.4 GHz network is shown in Listing 2.1.[10]

A wifi radio is usually able to transmit on various bitrates. They're configured to transmit at the highest possible rate if there isn't too much noise or the quality is

[10]HT40+: There are two 20 MHz channels. The primary channel (also known as *control channel*) is lower, the secondary channel is above

```
1   wifi24 = {
2                   ssid = 'batmanTestNetwork',
3                   channel = 6,
4                   htmode = 'HT40+',
5                   mesh_ssid = '02:d1:11:13:87:ae',
6                   mesh_bssid = '02:d1:11:13:87:ae',
7                   mesh_mcast_rate = 12000,
8 },
```

Listing 2.1 – Configuration for the 2.4 GHz network in the Freifunk firmware

too bad. However, multicast packets are usually transmitted at the lowest possible bit rate which is 1 Mbit/s. The reason for this is the low packet-loss that comes with a lower bit rate. Transmitting with a bit rate of 1 Mbit/s also means that more time is needed to transmit packets and this could become an issue if the network becomes large. As mentioned, Batman runs on each Freifunk node. With this, the virtual interfaces created by Batman are connected to physical interfaces of the devices. In addition, bridges are used. Figure 2.2 shows the internal interface structure of a Freifunk node operating with the Gluon firmware. The interfaces *br-wan* and *br-client* act as a network bridge. The Ethernet bridge is used to get multiple devices together. This connection is fully transparent, so hosts connected to one device see hosts connected to the other devices directly. As an example the interface *br-client* is used to connect all the client interfaces such as the 2.4 GHz, 5

[11] `https://media.hamburg.freifunk.net/Grafik/Technik/router-interfaces.pdf`

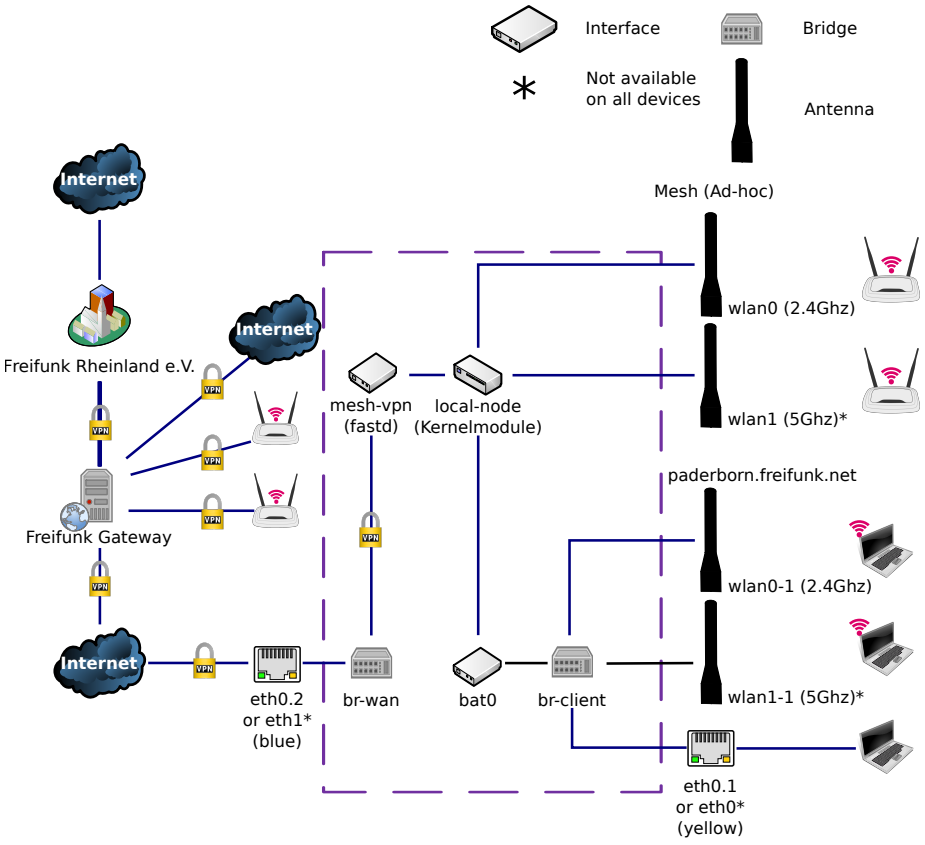


Figure 2.2 – Illustration of the network interfaces - Inspired by Freifunk Hamburg[11]

GHz and RJ-45 connectors. The *mesh-vpn* interface is used for the *fastd* connection and from this point the complete communication is encrypted. To include the wifi mesh connections, the interface *local-node* is used, which is also connected to the *bat0* interface and both are connected to the fastd vpn. As Freifunk uses free radio networks, which are not secured somehow, the complete transmission using the wireless channel is not encrypted and not authenticated.

Figure 2.3 gives a rough overview of the architecture used in the Freifunk Network of Paderborn. As shown, a node creates two wifi networks, one ad-hoc network that is used for the mesh connections and one infrastructure network that is used by regular clients such as computers or smartphones. The network for end users has the SSID *paderborn.freifunk.net*. Using the ad-hoc network, a mesh with other Freifunk devices in the nodes immediate surrounding is established. To solve the gap problem, which was mentioned above, all nodes are connected to the gateway infrastructure which acts as a switch between all nodes. To establish the VPN connection to a certain gateway, the node needs access to the Internet, as all gateways are hosted in several data centers.

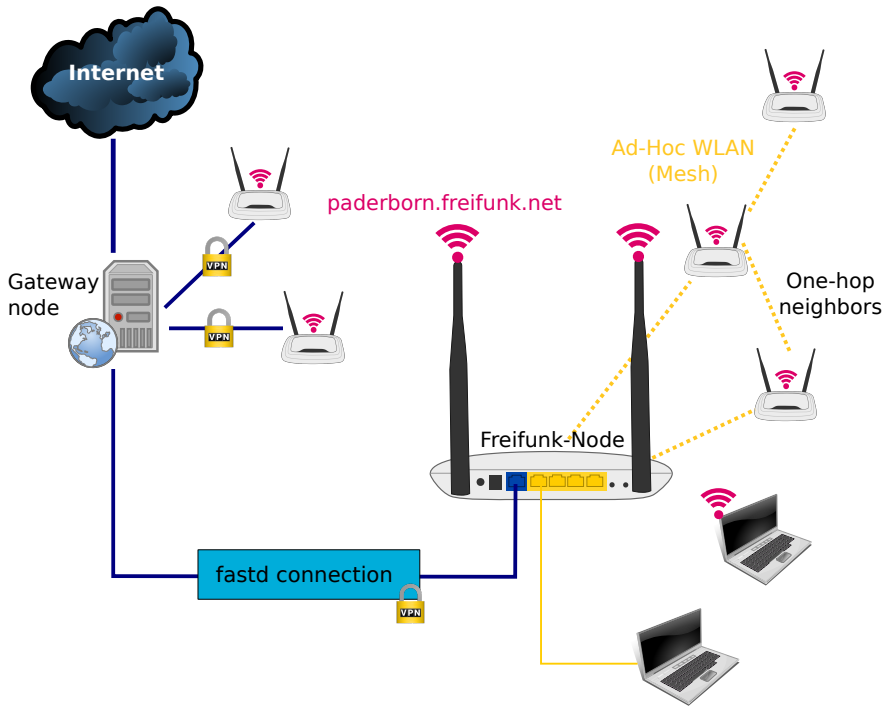


Figure 2.3 – Functional principle of a Freifunk node in Paderborn. To be part of the network, a nodes needs a connection to one gateway. This can be realized by using a direct VPN connection or by using wifi mesh connections. - Based on [29, Funktionsweise_Knoten.odg]

The implementation of Batman-adv uses different packets to implement various types of data transport, to gather information about the network and to discover nodes. All the packets are defined in the source code [15, packet.h,types.h]. In the following I will address important packet in a short manner.

B.A.T.M.A.N. packet / Originator Message The *Batman packet* is also known as the Originator Message (OGM). It is the most important packet of the Batman implementation and it was already used in the batmand implementation. An OGM is used to discover new information and routes within the network. This packet type is sent as a broadcast by every node with a fixed interval that is set to one second in the default implementation. This interval is called *originator interval*. An OGM is transmitted on the mesh interface *bat0*. According to Figure 2.2, those packets can be monitored on all interfaces which are responsible for the mesh routing. Those are *mesh-vpn* and *wlan0*. It is possible to monitor such packets using Wireshark[12] and tcpdump[13]. According to [15, packet.h] an OGM contains the following fields:

Version: The version of the Batman implementation. In Freifunk Paderborn the version 2013.4 is used, which belongs to version IV of Batman.

TimeToLive: Limits the number of nodes that can be reached by a single OGM. The value is used to ensure flooding termination and it is set to 50 [15, main.h].

Flags: Specifies whether there is a direct link to the sender or it is chosen as the next best hop.

Sequence number: Used to detect duplicates and to measure the link quality (see Section 2.3).

Originator: Gives the MAC Address of the node which created the OGM.

Previous sender: Gives the MAC address of last sender.

Transmit Link Quality (TQ): Describes the link quality for the complete route towards the originator.

If an OGM gets analyzed using Wireshark or tcpdump, additional fields are visible:

Gateway flags: This flag is used if the node provides a connection to another network for example the Internet. Furthermore, it announces the throughput by using this node.

[12] `https://www.wireshark.org/`
[13] `http://www.tcpdump.org/`

TT version number: Version of the local TT.

CRC of TT: CRC value for the TT.

Those fields are attached to each OGM using an extension of the Type-Length-Value (TLV) mechanism which is described at the end of this subsection.

An OGM has two major objectives. First of all, an OGM announces the presence of a node. The second objective is to carry quality information about all possible routes towards the announced node. Both objectives are described later in this chapter.

The *Gateway flags* are not important for the routing in the mesh network, as those flags are needed to announce the possibility to act as a gateway to another network like the Internet. Acting as a gateway node does not lead to a higher priority in the route selection process.

Unicast Packet A Batman unicast packet encapsulates unicast data that is provided from an upper layer. As the OGM, a unicast packet contains a destination address and a TTL field. Those packets are forwarded to other nodes if the destination non-mesh device is not connected to this node. This routing is based on the TT, which is build up by using the information from all received OGMs. If the non-mesh client connected to the node, the packet is immediately moved towards the destination MAC address.

Fragmented Unicast Packet Batman encapsulates the payload of a message and therefore it might be possible, that the length of a unicast packet exceeds the MTU[14]. Thus, the packet must be fragmented on the sender side and aggregated at the destination. Each fragment of the packet carries a part of the payload, and using the sequence number it is possible to aggregate the packet on the destination side.

Broadcast Packet Broadcast packets are forwarded to reach all nodes in the network. The sequence number, a TTL and the originators mac address are used to avoid duplicates.

Internet Control Message Protocol Packet The Batman-adv routing protocol operates at the link layer, and thus nodes in the network cannot always be reached by their IP addresses. Because of this Batman-adv supports ping requests, replies and failures using ICMP messages to support this subset of features from the IP-Version of ICMP on layer 2.

[14]Maximum size that can pass the layer

TVLV Looking at the OGM format and the purpose of such messages, some problem becomes quite obvious. As mentioned above, the OGM contains a number of fields needed for the routing and additional feature fields like the gateway flags or the version of the TT. Furthermore, there is a version number to indicate the version of Batman itself. The version number is used to interpret the routing information in the packet, and here lies a problem when developers want to add additional information in a new version of Batman. As long as a new feature doesn't have an impact on the mesh core functionality, it just needs some kind of announcement. This wouldn't lead to a problem.

During the Google Summer of Code (GSoC) [30] event in 2012, the task of the Batman-adv developers was to provide an infrastructure for sending, receiving and parsing information containers while preserving backward compatibility. The result was a feature called Type-Version-Length-Value (TVLV) which is based on the Type-Length-Value (TLV) mechanism. Due to experiences in the past, features evolve over time. Because of this, the new *version* field was added to differentiate between feature variants, so if a node sends a TVLV type, which is not known at another node, it is not required to process this packet furthermore and it gets dropped. However, a TVLV is transmitted periodically with each OGM and the transmission is completely transparent to the caller. Thus, every TVLV type is registered in some handler [15, main.c] to get separated from the real OGM, unicast packet and so on, to process the TVLV packet individually. A TVLV type could be a gateway- or translation table message, which were mentioned above.

2.1 Node discovery

The most important mechanism to establish a network using Batman-adv is the periodic transmission of OGMs. OGMs are used to inform other nodes about the own existence and to determine the link quality and distribute this information within the network. OGMs are sent on each interface that is configured to be used by Batman-adv.

Before version 2010.2.0 of Batman-adv, all configured interfaces were aggregated into a single virtual one which was called *bat0*. Since version 2010.2.0 it is possible to participate in multiple mesh clouds at the same time. For this to work, it is necessary to have multiple *batX* interfaces, one for every individual mesh cloud. All interfaces, that are supported by Batman-adv, have a sub folder named *batman_adv* in their folder located in */sys/class/net/*, so for example */sys/class/net/wlan0-1/batman_-adv/*. Here an interface can also get activated or deactivated for the usage of Batman-adv by using the following command.

```
root@thardes: cat /sys/class/net/mesh-vpn/batman_adv/iface_status
active
```

A Freifunk router has a number of interfaces created by Batman-adv and the Freifunk firmware. Those interfaces and the relations between them are shown in Figure 2.2. According to the description above there are two active Batman interfaces which are *wlan0* and *mesh-vpn*. By executing the following command, the real Batman-adv interface can be determined as shown below.

```
root@thardes: cat /sys/class/net/mesh-vpn/batman_adv/mesh_iface
bat0
```

Using Batman debugging tools like *batctl*, we can see which interface is used to reach a certain node in the network. Table 2.1 shows a small example for an originator table.

The interface *bat0* doesn't have a batman_adv folder, because it is a virtual interface created by Batman-adv and it is used as the entry point to the mesh network. It has a special *mesh* folder, which stores all parameters needed for the mesh routing. An example for the interface *bat0* is shown in listing 2.2. The details of all theses properties are skipped here as they are explained later in this document when they are needed.

The transmission of OGMs depends on the interfaces that are active and used to get connected with the mesh network. In a first step, every node periodically

Originator	last-seen	(#/255)	Nexthop	outgoingIF	Pot. nexthops
ea:98:f6...	0.864s	(176)	c0:ff:ee...	[mesh-vpn]	c0:ff:ee:... (176)
ea:98:f6...	9.220s	(105)	26:a4:3c...	[wlan 0]	26:a4:3c... (105)

Table 2.1 – Example for an originator table - Mac addresses are shortened

```
root@thardes: /sys/devices/virtual/net/bat0/mesh# ls -l
-rw-r--r--  1 root  root  4096 May 12 10:13 aggregated_ogms
-rw-r--r--  1 root  root  4096 May 12 10:13 ap_isolation
-rw-r--r--  1 root  root  4096 May 12 10:13 bonding
-rw-r--r--  1 root  root  4096 May 12 10:13 bridge_loop_avoidance
-rw-r--r--  1 root  root  4096 May 12 10:13 distributed_arp_table
-rw-r--r--  1 root  root  4096 May 12 10:13 fragmentation
-rw-r--r--  1 root  root  4096 May 12 10:13 gw_bandwidth
-rw-r--r--  1 root  root  4096 Apr  2 11:46 gw_mode
-rw-r--r--  1 root  root  4096 May 12 10:13 gw_sel_class
-rw-r--r--  1 root  root  4096 May 12 10:13 hop_penalty
-rw-r--r--  1 root  root  4096 Apr  2 11:46 orig_interval
-r--r--r--  1 root  root  4096 May 12 10:13 routing_algo
```

Listing 2.2 – Mesh folder of the bat0 interface.

broadcasts OGMs to inform other nodes in the network about its existence. Therefore, the network is flooded with OGMs by single-hop neighbors in the second step and by two-hop neighbors in the third step and so forth.

The complete processing of OGMs is handled in the file *bat_iv_ogm.c* of the Batman-adv source code. Therefore, all explanations in the following are gathered from [15, bat_iv_ogm.c], if not stated otherwise. If a node receives an OGM, it has to perform some major checks before it can proceed, and it is dropped if one of the following preliminary conditions become true:

1. The Batman version of the OGM is different from the node's version.
2. The sender is a multicast or broadcast address.
3. The destination is a multicast address.
4. The sender is the address of the node itself, so a node receives a direct broadcast of itself.

If those steps were successful, the originator must be proven to be a potential router. Those router checks ensure, that no packet is routed within the network in an infinite loop and always the best hop is chosen for a given destination.

In order to realize the router checks, a list with all originators is needed, which is called *originator list*. This list stores all reachable originators in the mesh network. An example is shown in Table 2.1. First of all, there is one list entry for each known originator in the network, which is stored together with important routing data. The columns are discussed in the following:

Originator: The MAC address of the originator that was found in the network. As Batman-adv operates on OSI layer 2, the MAC address is enough to clearly identify a mesh participant.

last-seen: A time stamp to save the time when the last OGM of this originator was received. This value is important in order to purge outdated entries. This could be the case if an originator leaves the network as the route to this device becomes invalid and is not usable any more.

(#/255): This value identifies the Transmit Link Quality (TQ) value. The TQ is used as a routing metric in order to measure the quality of a link.

Nexthop: Batman-adv remembers in which direction to send packets if data should be received by a certain node. The direction manifests itself in the form of the *next best hop* which is basically the next step towards the destination.

outgoingIF: The outgoingIF specifies the hardware interface to use if the originator should be reached, as Batman-adv supports multiple interfaces.

Potential nexthops... : A list of alternative hops, if the best next hop cannot be used.

After an incoming OGM passes the preliminary checks, which were mentioned above, the originator is added to the originator list, if it is not known yet. A new entry is added without any routing information, because the information from the OGM may lead to routing loops or may be outdated and so on. If the node was already registered, the last received sequence number has to be compared with the one, which was received with the OGM as the node might already have newer values. In addition, the sliding window is checked for that sequence number as this OGM might be a duplicate. If one of these conditions is true, the OGM is dropped. If the packet wasn't deleted, the sequence number for the originator is updated.

After that, the TQ value is checked. Each OGM with a TQ greater than 0 or which was created from a single hop neighbor is accepted and further processed. Otherwise, the OGM gets dropped. As mentioned, an OGM is transmitted within a raw Ethernet frame which contains a source and destination MAC address. Using this information and the field with the originator address from the OGM, it is possible to check whether the sender is a direct neighbor, as the sender MAC from the Ethernet frame and the originator address are equal in that case.

In a next step, the TQ value needs to be updated. The OGM's TQ field is multiplied with the TQ stored for the neighbor, the OGM has been received from. The calculation of the TQ is described later in this chapter. However, if the new TQ value becomes zero, the link is not considered to be bidirectional which is related to the formula of the TQ calculation. With the calculation of the TQ value, all steps for the OGM are passed. The procedure for checking incoming OGMs is summarized again in Figure 2.4. The OGM handling is continued with the router ranking, which is explained in the following.

According to the checks mentioned above, the OGM wasn't deleted yet and the corresponding originator and the previous sender are valid originators, which can be considered as potential loop-safe routers. Furthermore, the OGM contains newer information because of the sequence number checks. The OGM might also have been received from a neighbor node which wasn't registered as a router for the originator yet. If the neighbor was not known, a new entry is created and filled with the information from the OGM processing mentioned above. If there is already an entry for this neighbor, the entry is updated with the new information. As shown in Table 2.1, there is always one router which is chosen to be the next best hop towards an originator. This router is chosen during the router ranking. The router ranking is explained as a flowchart in Figure 2.4.

First of all, it is checked, whether any router is already selected for this originator. If this is the case and the chosen neighbor is equal to the selected router, no update

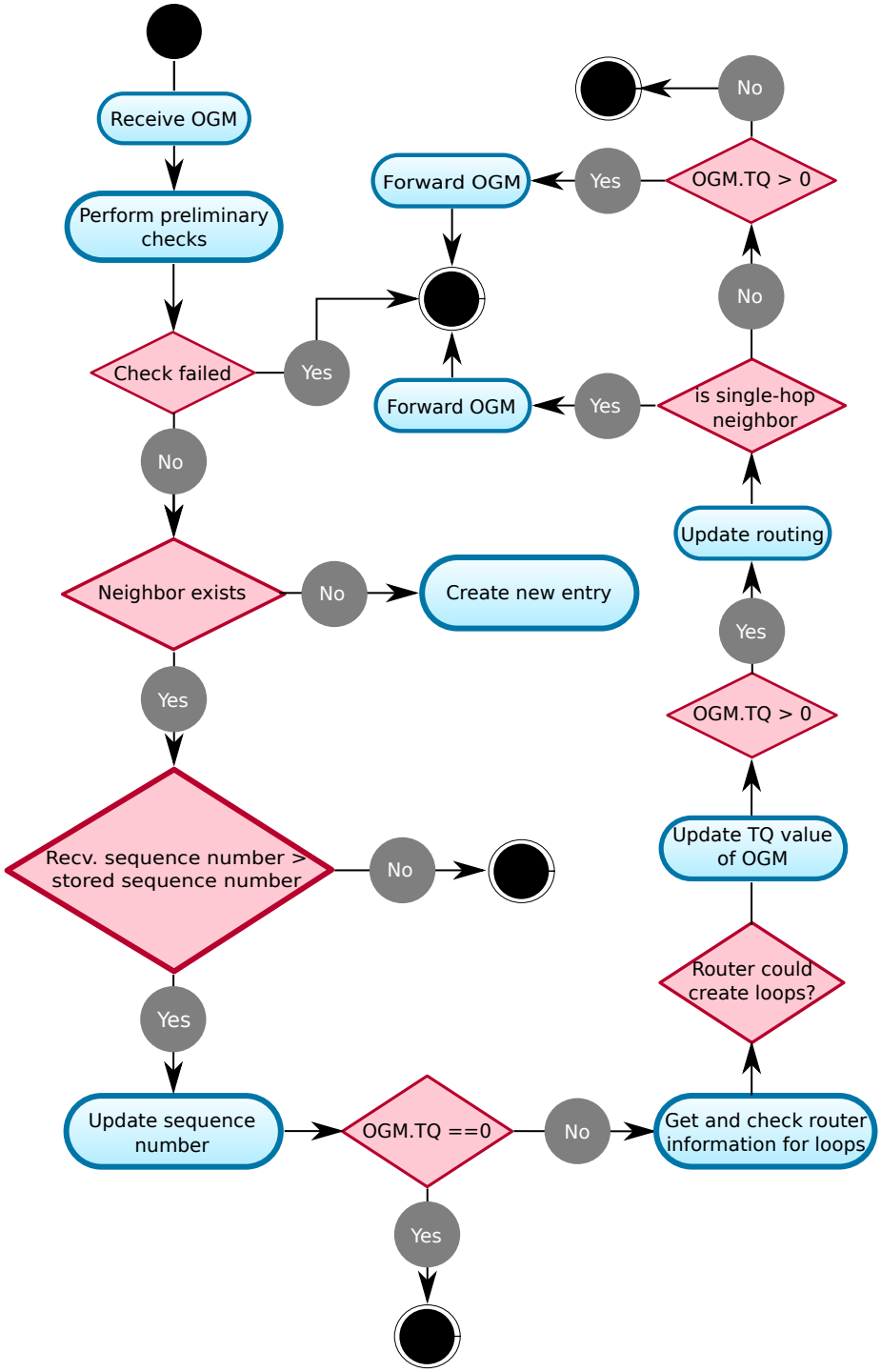


Figure 2.4 – Potential router checks as a flowchart

is required, as the information is already up to date. If both entries are different, the neighbor with the better TQ value is chosen, and if there wasn't any router yet, the new neighbor will become the best next hop.

It might happen that a neighbor becomes unreachable. In this case there are no further OGMs received from this node. This scenario leads to a stale entry for a node, and the value for the *last-seen* property is not updated any more. However, if the best next hop disappears, the TQ value of the other routers will never be better than the TQ value of the best router that disappeared. Therefore, a mechanism is needed to purge outdated routers to be able to select other routers to be the best next hop. Batman-adv solves this problem by using a timeout mechanism. A constant value of 200 seconds [15, main.h] is used to validate, whether an originator is outdated or not. As the receive time of an OGM for a given originator is stored in form of the last-seen property, it is easy to check whether the node has been seen during the last 200 seconds or not. If the node is outdated, the originator entry is removed from the list and all neighbor entries are also deleted. However, since the best next hop might be removed, the routes are also updated after a node was removed and another router gets selected.

Finally, an OGM needs a rebroadcast. The TTL has to be decremented by one, and if the TTL becomes zero, the packet is not forwarded but gets deleted. The second step is to apply the hop penalty which affects the TQ value of the OGM. The hop penalty is a value, that is applied to the TQ value of each forwarded OGM, and it is used to propagate the cost of an extra hop in the network. The default value is set to 15, which means, that the TQ gets reduced by 15 points.

If an OGM shall be forwarded, it gets added to the packet queue of the node. Due to this process, it is possible to aggregate multiple OGMs into a single packet to reduce the protocol overhead. The aggregation of OGMs is related to the overall packet size of the aggregated packet afterwards, as the size has to be smaller than the MTU for the network. Furthermore, the estimated time, when the OGM is scheduled to be sent, is important and Batman also considers the interfaces on which the packet should be scheduled and on which it has been received. An own OGM never get aggregated. Using this mechanism, multiple packets gets aggregated and encapsulated within a single Ethernet frame.

2.2 Translation tables and client announcements

A node in the network is represented by its MAC address, and the node is a part of the mesh network itself. A client is a non-mesh entity, which uses the network to exchange data or to connect to external services like the Internet. A client could be mobile and therefore a moving object within the network. Otherwise it is a stationary

device. Since it is a non-mesh participant, there are no OGMs exchanged. In order to get the location of those clients there is another mechanism. This mechanism is implemented with the Translation Table (TT). If the clients position wouldn't be published in the network, no client would be able to receive traffic because no other node would know where to send the packets to. The existence of the TT is also needed to provide roaming functionality and to avoid a connection loss in this scenario.

According to [31], we can distinguish between two different types of the TT, the *local* and the *global* Translation Table. The local TT is stored on each node in the network with local information, such as clients which are connected to this node. The global TT is also stored on each node, but it contains client information from the entire network.

The local TT has a version that is transmitted within an OGM. The version is specified by the Translation Table Version Number (TTVN), which is an individual version number for the local TT of each node. By using the TTVN it is possible to determine, whether the client data for a certain node in the network is up to date. This is possible, as the stored TTVN for a node is the same like the TTVN from the received OGM in this case. After the version has changed, all other nodes in the network need to receive the changes to update the global TT. Thus the node with a new client has to propagate the changes by using the OGM mechanism. However, the node will not transmit the complete local table again. To reduce the overhead, only changed values are transmitted. At boot time the node initializes the TT as empty and sets the TTVN to zero. From this point, each time the node recognizes a local change, like a client joining or leaving the node during an OGM interval, the node will increase the own TTVN by 1 and update the TT with the new information. Nevertheless, the TTVN is just updated once per OGM interval, independent from the number of changes. Thus, an incrementation of the TTVN means that there was at least one change during the last interval.

The global TT is used like a routing table with a client address as the key and the originator address as the result. This is because a client is always connected to a certain originator. So every time a packet has to be sent to a client, the originator address of the node has to be retrieved as the destination for this packet. Thus, it is important to keep the global TT up to date by using various update mechanisms.

Figure 2.5 gives a flow chart for the update process of the global TT. In order to exploit the transmission of full sets of changes within one OGM interval, each OGM can carry the last set of changes that was triggered with the incrementation of the TTVN by using the TVLV mechanism. However, this is only possible if the size of the last change set is small enough to fit in one OGM. Otherwise, nothing is attached and an explicit update request is required. If an update is required, the node sends a *TT_request* to the node which recently updated its local TTVN. According to

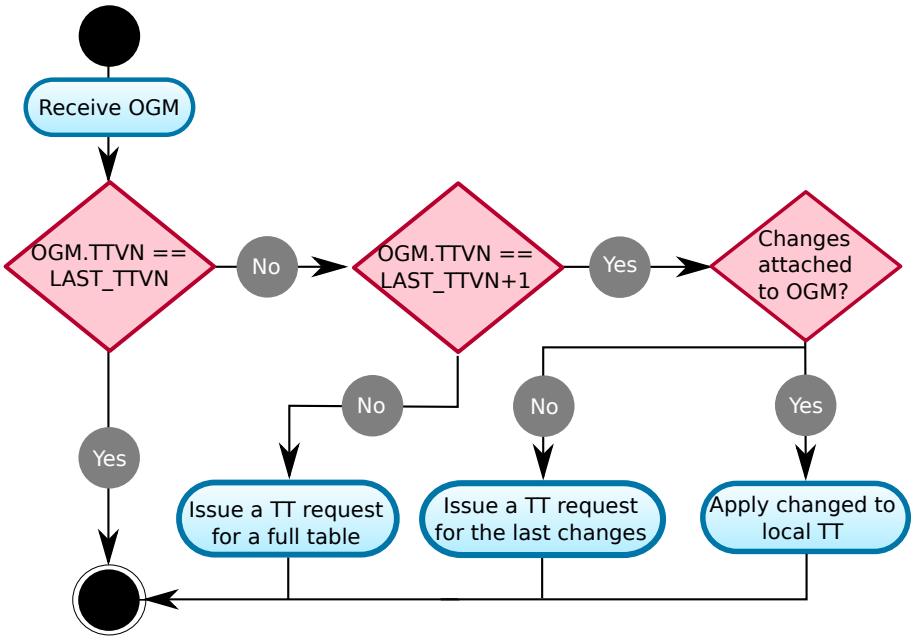


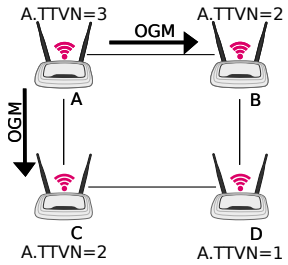
Figure 2.5 – Flow chart for the update process of the TT - Based on [31]

Figure 2.5, the reply can either be just a change set for one interval or a full table request. A full table request might be required if a new node joins the network as its global TT doesn't contain any entries yet. If a TT update is announced with an OGM and the OGM doesn't carry the changes, all nodes in the network ask for the corresponding updates of that node. However, this leads to a very high overhead of table request operations. To reduce this overhead, each node on the path checks its local content and decides whether it can provide the required information to answer the request [31]. If the node is not able to answer the request, it will just route the packet by using the next best hop towards the destination. Figure 2.6 shows a possible scenario with four nodes, where one node updates the local TT and the changes get distributed within the network by the OGM mechanism and by TT_requests and TT_responses.

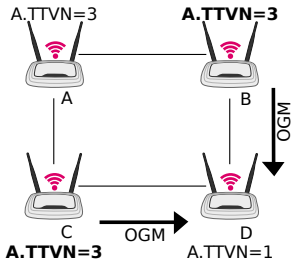
The command *batctl tl* gives the local translation table of the selected node. It shows all clients, that are connected at the moment, with additional parameters like flags[15] and a timestamp when it was seen the last time. *batctl tg* gives the TT for the global network. The Listing 2.3 shows an example. The columns for the local and global TT are mainly identical:

Client: This column stores the MAC address of the non-mesh client. As MAC addresses are unique, this address is enough to identify the client.

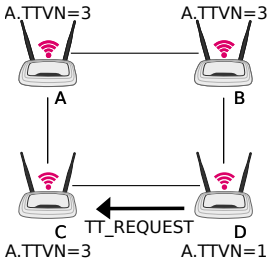
[15] `http://www.open-mesh.org/projects/batman-adv/wiki/Understand-your-batman-adv-network`



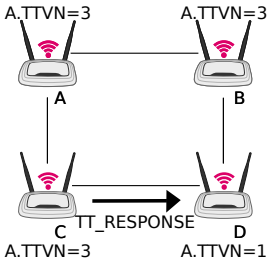
(a) Node A added a new client to its local table and propagates this change with an OGM.



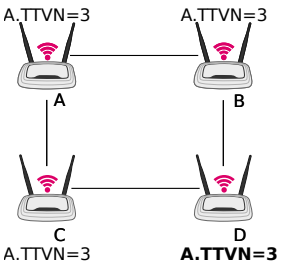
(b) Node B and C receive this OGM and update their global tables with the attached changes. Both nodes rebroadcast the OGM from A.



(c) Node D also detects the changes and issues a TT_Request as an unicast to node A via the next best hop. This is done as the difference between the stored TTVN for node A and the received one is too high.



(d) Node C inspects the packet and checks whether it can answers the request. If this is possible it directly answers the message without forwarding it. As this is possible in this scenario, it directly answer the request.



(e) Node D updates its local TT by using the data provided by node C.

Figure 2.6 – Scenario for updating the global Translation Table - Based on [31]

```
root@thardes:~# batctl tl
Locally retrieved addresses (from bat0) announced via TT ↘
     (TTVN: 129 CRC: 0xdd09):
        Client        Flags   Last seen
 * e8:94:f6:63:13:7c [.P...]   0.000
 * bc:f5:ac:fe:c4:ad [....W]  24.730
 * 00:21:6a:08:0c:f6 [....W]  26.920
root@thardes:~#
root@thardes:~# batctl tg
Globally announced TT entries received via the mesh bat0
   Client       (TTVN)    Originator   (Curr TTVN) (CRC ) Flags
 * 48:43:7c:... ( 51) via c6:72:1f:... ( 55)       (0xde) [.W.]
 * 84:38:35:... (150) via c6:6e:1f:... (181)       (0x6a) [.W.]
...
```

Listing 2.3 – Example for the local and global TT. MAC addresses in the global TT are shortened

TTVN: The TTVN, the client was announced the first time.

Originator: The MAC address of the originator this client is connected to.

Current TTVN: The last announced TTVN of this originator.

CRC: Checksum for this entry.

Flags: Flags which were attached to this client.

Additionally, the local TT stores the last-seen timestamp for each client. This value is later required to purge outdated entries. As mentioned in the Listing 2.3, there are flags stored for each client. Those flags are important as they provide certain properties for a non-mesh client [15, main.h,translation-table.c]

R (Roaming): The corresponding client already moved to another node. It is kept in the table until the next OGM is sent.

P (noPurge): This entry must not be purged. As the real hardware interfaces like the wireless interface are used by Batman-adv, but not directly controlled, the interface is also stored as a non-mesh participant.

N (New): The client was added to the list but is wasn't advertised yet because no new OGM has been sent.

X (Delete): The client has to be deleted, which is done with the next OGM of this node.

W (Wireless): This client is connected to the node through a wireless device.

Some of the flags mentioned above are just temporary. They are necessary to ensure consistency in the network. For example the roaming flag is just used until the non-mesh client gets registered and announced within the new node which it is connected to. Temporary roaming and purging of non-mesh clients is explained in Section 2.5.

2.3 Neighbor ranking and Link Quality Estimation

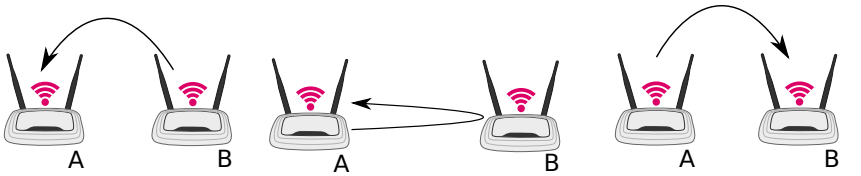
With Batman version IV, there were fundamental changes in the mechanism for neighbor ranking. Version III used the number of received OGMs [32] to rank nodes in the network. With version IV the TQ metric was introduced, that was published in a similar way by De Couto et al. in [24]. The TQ metric is used to find a trade-off between low hop count and stable links [24,33], and it is representing the probability of successfully transmitting a packet to another node.

To calculate the TQ value, Batman-adv keeps track of a value representing the probability of successfully receiving packet from another node (Received Link Quality (RQ)) and the probability of successfully receiving an own OGM that was broadcast before (Echo Link Quality (EQ)) [1].

2.3.1 Received Link Quality

As already mentioned in Chapter 2, Batman-adv is sequence number oriented. In fact, the sequence number of an OGM is one of the most important aspects of information that is transmitted.

The calculation of the RQ value takes place by using a *sliding window* of size N. The sliding window has a size of 64 bit [15, main.h] which leads to 2^{64} possible entries. It keeps track of the last received sequence numbers of OGMs and the current received from each node in the network. Sequence numbers are distinguished between *out-of-range* and *in-window* sequence numbers. The *in-window* sequence numbers are those, which fit in the window below the current sequence number. If an *out-of-range* sequence number is received, it is set as the current sequence number



(a) Received Link Quality (RQ) **(b)** Echo Link Quality (EQ) **(c)** Transmit Link Quality (TQ)

Figure 2.7 – Illustration of RQ, EQ and TQ values - Based on [1] and [9]

and the sliding window is moved accordingly. Sequence numbers, that are not in the sliding window any more, are deleted.

Figure 2.7 gives a visual definition for all the three values. For the RQ value, node B transmits an OGM, that is received by node A. The task of A is to calculate the RQ value based on the number of received packets of a certain node. This is simply the percentage of received OGMs within the window size N.

$$\mathrm{RQ} = \frac{\#\mathrm{MAC(Node)}}{N} \tag{2.2}$$

2.3.2 Echo Link Quality

Every time an OGM is received from a neighbor, it is rebroadcast according to the rules mentioned previously in this chapter. As shown in Figure 2.7b node B rebroadcasts the OGM, that was initially created from node A. Since B is a one-hop neighbor, A receives this rebroadcast of its own OGM and with this the EQ value is calculated in the same way as in Equation (2.2) using the sliding window of size N and the number of received broadcasts of node A within the sliding window.

So the OGM was sent by node A, received and broadcast by B and again received by A. It is the probability of transmitting a packet correctly both from A to B and back from Node B to A like an echo. This brings us to the definition of the Transmit Link Quality.

2.3.3 Transmit Link Quality

The TQ is the most important value as the RQ value is determined by the number of received OGMs, whereas the EQ is the number of rebroadcast OGMs from a neighbor. With both values, it is possible to determine, whether the link is a bidirectional one and to create a metric to compare different links. The TQ value is also used to inform all other nodes in the network about the link quality at this point. That's the reason why the TQ value is sent within an OGM.

The TQ is a numeric value between zero and one. It is stored in one byte [15, bat_iv_ogm.c], 0x00 representing a zero and 0xff representing a one. It can be derived using RQ and EQ according to [1] and [15, bat_iv_ogm.c]:

$$TQ_{\mathrm{local}} = \frac{\mathrm{EQ}}{\mathrm{RQ}} \tag{2.3}$$

Since an OGM is rebroadcast, the TQ value in the packet has to be changed. For this, the receiving node calculates the own local TQ value and takes the received TQ value from the OGM to perform the following calculation [15, bat_iv_ogm.c], [9]:

$$TQ_{\mathrm{global}} = TQ_{\mathrm{local}} \times TQ_{\mathrm{received}} \tag{2.4}$$

If the OGM from node A reaches another node like shown in Figure 2.8 (node C), the node calculates the new value for TQ_{global} and chooses the highest result to get the next best router towards the destination when forwarding the message again.

Before the TQ value is stored in the OGM the *hop penalty* is applied. The *hop penalty* is a value that is applied to the TQ value of each forwarded OGM and it is used to propagate the cost of an extra hop in the network. Like the TQ, it is a value between 0x00 and 0xff. It is applied as shown in Equation (2.5), where the variable HP is the value for the hop penalty [13]. The *hop penalty* can be configured for the Batman-adv interface as shown in the first part of Chapter 2.

$$TQ = TQ_{\text{received}} \times (TQ_{\text{max}} \times \text{HP}) \tag{2.5}$$

The reason why the *hop penalty* is applied is simple. The *hop penalty* degrades a path with a lot of hops. It is used to reduce latency and to save bandwidth [13].

Because of Equation (2.3) it is possible to perform the bidirectional link check by using the TQ value. As the RQ is the number of OGMs, which were received by the other node, and the EQ gives the number of rebroadcasts that were received, the link can be considered to be bidirectional if the result of Equation (2.3) is greater zero. The default limit to consider a link to be bidirectional is set to 0x01 [15, main.h]. However, this limit could be increased, but it will reduce the coverage of an OGM within the network.

Figure 2.8 shows an example calculation of TQ values according to [9]. Here, node A starts with sending an OGM that is received by B. The initial value for the TQ is 100 %. B receives the OGM with the initial TQ and has a measured one stored, which is set to 80 % in this example. We can apply the calculation shown in

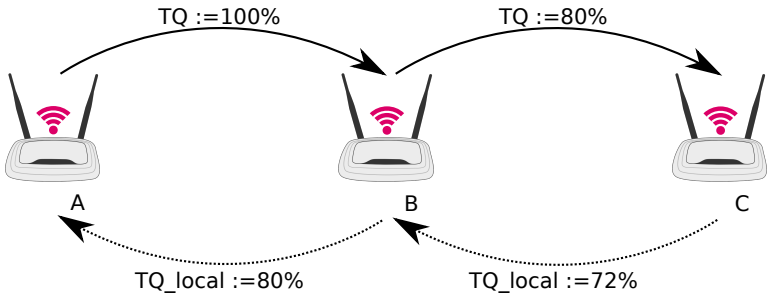


Figure 2.8 – Calculating TQ values from OGM. The TQ is changed every time when an OGM gets forwarded. The initial value is set to 255 - Based on [1] and [9].

Equation (2.6):

$$\begin{aligned} TQ_{\text{global}} &= TQ_{\text{local}} \times TQ_{\text{received}} \\ &= 80\% \times 100\% \\ &= 80\% \end{aligned} \tag{2.6}$$

So the result is $TQ_{global} = 80\%$, which is used for the rebroadcast. To simplify the scenario, the hop penalty is skipped here. For node C we assume a stored TQ value of 90 % and so we get the calculation shown in Equation (2.7):

$$\begin{aligned} TQ_{\text{global}} &= TQ_{\text{local}} \times TQ_{\text{received}} \\ &= 90\% \times 80\% \\ &= 72\% \end{aligned} \tag{2.7}$$

Every time the value for TQ_{local} is transmitted to the last node, this information is used to choose a route within the network.

Batman-adv forwards data packets using a path that is chosen to have the highest probability for a successful transmission. The TQ value stored in the OGM is replaced with the one that applies for the best ranked neighbor. With this, every node always advertises its best possibility to forward packets to the destination.

Assuming a real world scenario, where unicast packets get acknowledged, it is possible that a unicast packet gets transmitted successfully, but the acknowledgement packet always gets lost. This results in a retransmission of the unicast packet until there is a timeout for this transmission on the senders side. In fact, studies like [34] or [35] have shown that wireless links in networks are vulnerable to be asymmetric in their transmission quality. There are many reasons for a link to be asymmetric, since the radio wave propagation is affected by obstacles and different environment conditions like noise levels and node densities. Furthermore, there are different hardware and antennas available, which have a special direction pattern. That's the reason why a correct evaluation and quality rating of asymmetric links has a high impact on the used routing technique as shown in [36]. Because of the problem with asymmetric links, Batman-adv introduced an additional penalty for asymmetric links which is shown in Equation (2.8) where AP is the asymmetric penalty.

$$\begin{aligned} AP &= 1 - ((1 - RQ)^3) \\ TQ_{\text{new}} &= TQ_{\text{old}} \times AP \end{aligned} \tag{2.8}$$

The influence of the RQ is non linear, which leads to a smaller decrease of the TQ if the RQ gets high and a fully asymmetric link gets a value of zero. This completely avoids asymmetric links. Therefore, this penalty is also applied to the OGM.

Batman-adv does not need knowledge about the full network, but only information about the one-hop neighborhood. Thats the reason why just the next best hop is chosen if a packet reaches a new node in the network. In the scenario of Figure 2.9, node F transmits a packet that should reach node A. F just knows node D and E and in the scenario node D is chosen as the best hop, because of the global TQ values that characterize the path through node D and E. In the next step, node D performs the same step based on its knowledge about node B, C and E. Here, node C is chosen, which knows Node A and transmits the packet directly.

As mentioned above, there is a virtual interface called *bat0*, which is used as an entry point to the mesh network. If a packet goes through this interface and it is passed to Batman-adv as a layer 2 frame with the *bat0* properties as the source address. In fact the originator needs to know the destination of a non-mesh client which can be found by using the global TT and by choosing the next best hop before it is handed to the device driver for physical transmission [1]. Afterwards, the routing is handled by Batman-adv searching the destination MAC in the global TT.

2.4 Gateway nodes

As shown in Figure 2.3, each node is connected to gateways using a VPN connection. In fact, Paderborn uses two gateways per node to ensure reliability. In principle, gateways are used if a connection to the Internet should be provided. This connection

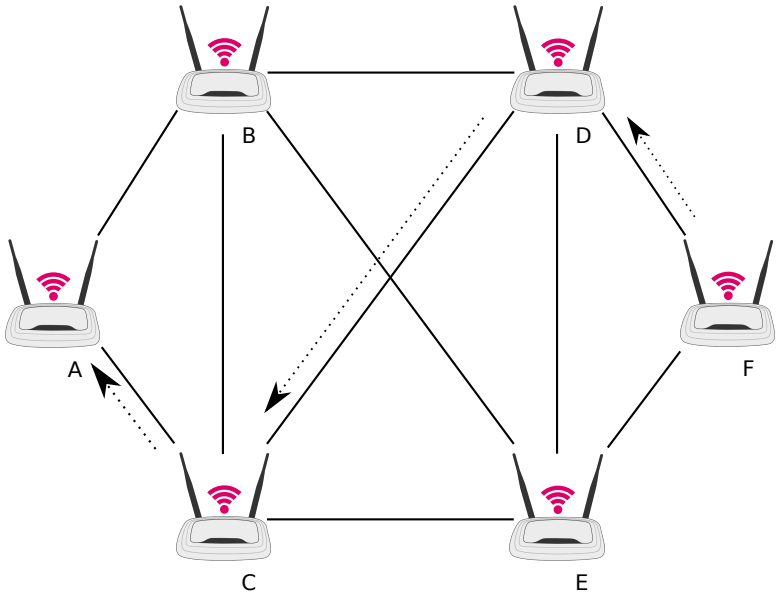


Figure 2.9 – The decision for the next best hop is based on the TQ value that is stored for a certain originator in the network - Based on [1].

cannot be directly established by the node, because of legal restrictions mentioned in Section 1.1. Due to the TVLV mechanism, a gateway is just a Batman-adv node with certain flags. If the gateway feature is activated, the network is flooded with the gateway information using the OGMs. So each node in the network knows about the new gateway including the link quality [15, packets.c].

According to [31] and [15, gateway_client.c,gateway_common.c], the gateway functionality is based on DHCP. Batman-adv assumes that each gateway runs a DHCP Server and a client runs a DHCP client. Usually, a client issues a DHCP request when it joins a network, which is then broadcast in the network. All DHCP servers reply with a unicast message to the client and the client chooses one of the servers. However, Batman-adv changes this routine in order to avoid DHCP messages in the network. Each node knows about the gateways in the network by using the OGMs. The idea is to provide a preselected DHCP server at the node to avoid broadcast messages of the clients in the network. Thus, the node forwards the DHCP message just to the selected gateway in order to receive an answer that is then directly forwarded to the client.

As the node knows the TQ values and the announced throughput for all gateways, it is possible to choose a gateway based on different criteria. There are four different criteria to choose a certain gateway in the network. Those parameters can be configured by the user or by the used firmware [37] [15, gateway_client.c].

20 - Late switch This parameter is used in Freifunk Paderborn. It says, that the gateway should be switched if the difference of the TQ values is more than 20 %.

1 - fast connection: Using the advertised throughput and the TQ value. The selected gateway is then used until it disappears.

2 - stable connection: Use the TQ value and use this gateway until it disappears.

3 - fast switch connection: Use the gateway with the best TQ value and switch to another, if the TQ value is even better.

XX - late switch connection: Take the gateway with the best TQ value and switch to another, if the TQ value is at least XX points higher than the current selected one.

A node in the network will choose the corresponding gateway based on the criteria above. If a client sends a DHCP request, Batman-adv will not broadcast this message within the entire network, but only send it to the server that was previously chosen. With this, the client has to use the chosen gateway of the connected node. If the node changes the selected gateway, Batman-adv forces the client to request a new IP address by dropping the renewal packet.

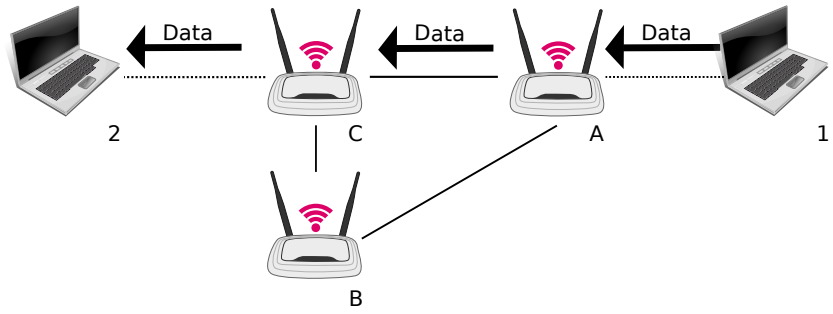
2.5 Client roaming

The procedure of a non-mesh client switching to a new mesh node is called *roaming*. Clients are announced using the TT mechanism, and as clients are mobile and traveling within the network, roaming is an important functionality. The overall procedure to support roaming is simple. It can be divided into different steps. All steps are explained visually in Section 2.5.

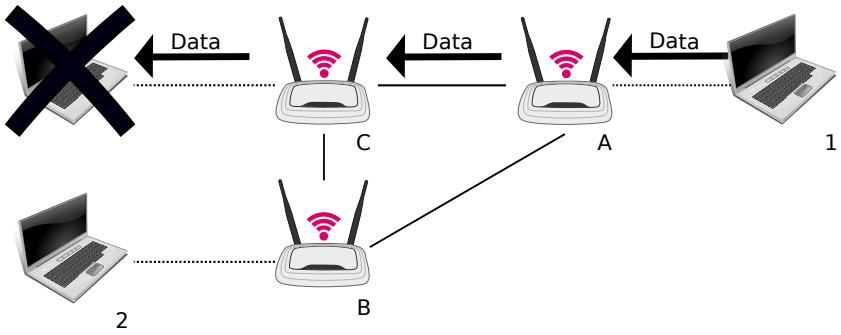
Whenever a client travels and changes a node, the node needs to inform the entire network that this particular client is now connected to another node. As explained, clients are announced by using the OGM mechanism, so even with perfect condition it would take at least one OGM interval to announce the new client and to enable the client to be reachable within the network by other participants. This bottleneck is unacceptable and can be avoided because Batman-adv is in full control over payload traffic of non-mesh clients. The mechanism, that is used by Batman, is an explicit advertisement of a roaming event, so that the involved nodes can apply the according changes to their own routing table. According to [31], this enables nodes to make already established connections continue to work even if they are not optimal.

Detecting a roaming client can easily be done by using the global TT, as this table contains all non-mesh clients in the network. If a node registers a new non-mesh client and there is a positive match in the global TT, this node immediately sends a *roaming advertising message* to the originator that is stored in the global TT for this client mac address (Figure 2.9c). With this message, the old node can update its global TT, so that payload traffic gets rerouted to the new node (Figure 2.8d). If the old node receives data for the client, the destination field of the unicast packet is adjusted to be sure to reach the new correct router for the client. The data is rerouted to the other destination and the information in the header are updated until the mesh network gets in sync again by sending the next OGM (Figure 2.8e). The old originator will mark the client in its local TT with a special flag to mark it as a roaming client. Section 2.5 explains the roaming of a client in a visual way.

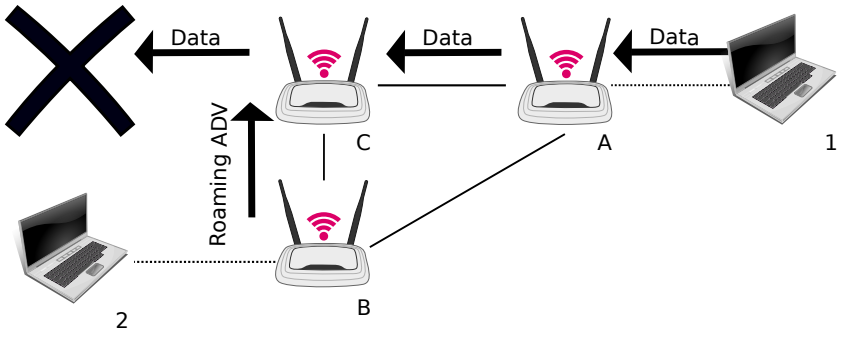
There could be a different scenario than illustrated in Section 2.5, where a non-mesh client travels along more than one node. As an example, a client moves from A to B and then from B to C. The first move is handled in the same way as mentioned above. When the client moves from B to C, node C has no knowledge about the previous move from A to B, as no OGM was sent in the mean time. Because of this, C sends a roaming advertisement to node A, and node A updates the global TT again to route new incoming packets to C. The complete network will be synced again with the next OGM.



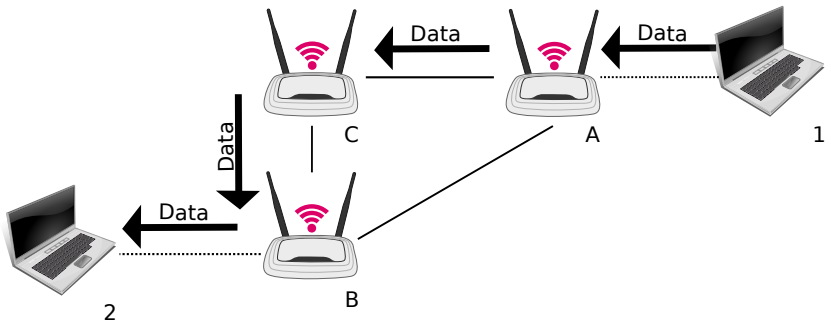
(a) Client 1 and client 2 are connected to two different nodes of the network. Client 2 receives data from client 1 using nodes A and C.



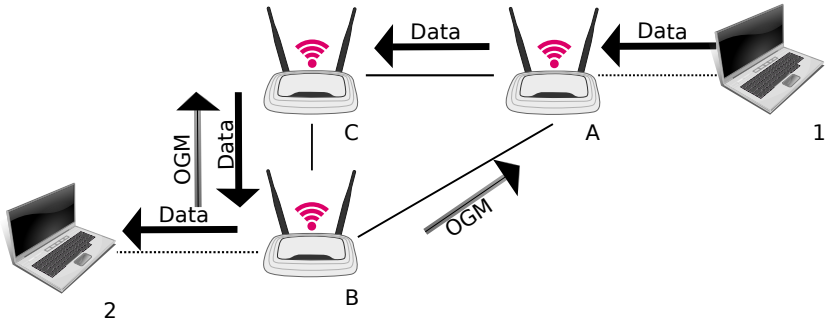
(b) Client 2 travels and moves to node B. All data gets lost at node C.



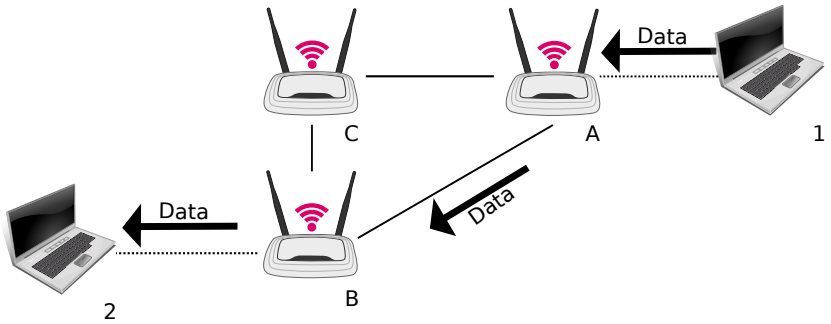
(c) Client 2 gets registered at node B and B sends a roaming advertisement to A.



(d) C updates its TT and reroutes all traffic to node B.



(e) Node B transmits an OGM which contains the new position of client 2.



(f) A updates the route towards node B and now Client 2 receives data from client 1 using nodes A and B.

Figure 2.7 – Roaming client in a Batman network. A roaming client is announced by a special message type. The roaming information is just sent to the node, which was previously used by the client. The complete network becomes consistent again with the next OGM of the new node. - Based on [31]

2.6 Comparison of B.A.T.M.A.N. III and B.A.T.M.A.N. IV

Taking a look at B.A.T.M.A.N. III (batmand)[16] [14, 19] and B.A.T.M.A.N. IV (Batman-adv)[17] [10, 32, 33] differences and similarities can be identified between both versions.

Both protocols are split into three different tasks:

1. Checking the usability of local links.
2. Usage of OGMs to flood the network.
3. Rating the quality of a path in the network.

Furthermore, in both versions, a single node has only detailed knowledge about their direct neighbors. This knowledge is distributed within the complete network to enable other nodes to create routes.

For checking the usability of local links, batmand uses OGMs that travel in both directions to form a bidirectional link. In Batman-adv, this is done by using a sliding window mechanism (see Section 2.3). The TQ metric was also introduced to determine the quality of links between nodes and to check whether these are bidirectional.

In order to reduce the overhead, which results from distribution the OGMs, Batman-adv supports aggregation of those messages. Instead of sending several small packets, those small packets get aggregated into a big one. Whenever one OGM needs to be forwarded, all further messages received before a certain timeout are attached to it, until the size is within a predefined interval and other checks are successful.

Batman III used an additional packet type called Host Network Announcement (HNA), to announce new neighbors. This is solved by using the TT, the TVLV mechanism and the version number of the table in the OGM by Batman-adv.

To improve the performance of batmand, Batman-adv exploits all available interfaces that are connected to the mesh cloud. The *alternating interface strategy* modifies the routing in order to forward packets on a different interface than on the receiving one. Using this strategy, interference is reduced, as a wifi interface can either receive or send data at a given time, and on the other hand this feature balances the network load.

If multiple interfaces and multiple paths towards a destination are available, Batman IV is able to distribute all frames using both paths. This feature is called

[16] `http://www.open-mesh.org/projects/batmand/wiki`
[17] `http://www.open-mesh.org/projects/batman-adv/wiki`

interface bonding and it could increase the throughput. However, paths can have different speeds which is not detected by Batman-adv, and due to this, the throughput can decrease. In the current implementation, the interface alternating feature is enabled and the interface bonding strategy is disabled by default. While routing protocols like OLSR, Babel or batmand work on OSI layer 3, Batman-adv uses the link layer. It is actually running in the kernel, so strictly speaking it is between the data-link and the network layer, as it is not directly working on the NIC. There are also other mesh protocols, like the IEEE 802.11s standard[18], which is working on OSI layer 2 as well. Acting on OSI layer 2 has both advantages and disadvantages. The implementation represents a new element in the network stack [31] which uses its own headers and services.

Advantages for routing on layer 2 The first argument is transparency. As the mesh networks acts like a Ethernet switch, the mesh is completely transparent to upper layers. With this, the network communication does not depend on mechanisms like IP or other protocols, which are operating on higher layers. Furthermore, it is possible to work with multiple interfaces, as the only requirement is to send Ethernet frames. Those different interfaces can be included or excluded from the mesh. With this mechanism, multiple networks can easily be connected. Another possibility is to use multiple interfaces to create various links along a path, to send and receive packets alternatively on multiple interfaces at the same time.

Disadvantages for routing on layer 2 Using layer 2, the addressing of devices can become hard, as just MAC addresses are used. IP addresses are structured and can carry information about the networks topology. This problem arises with the client announcement, when a new client is made public by using the OGMs.

Another disadvantage is the fragmentation. Since fragmentation of IP cannot be used, a separate mechanism has to be implemented.

2.7 B.A.T.M.A.N. V

B.A.T.M.A.N. IV uses OGMs to calculate the quality of links and sends this information to direct neighbors, spreading the link quality information through the whole mesh network. However, this approach has some drawbacks, explained in [32]. Usually, a wireless interface causes higher packet loss and thus a higher broadcast rate for OGMs is required in order to quickly detect bad links. Interfaces like the VPN connection

[18] `https://github.com/o11s/open80211s/wiki/HOWTO`

over the Internet lead to a lower packet loss, and those interfaces would benefit from using a lower broadcast rate. Furthermore, it is more important to detect local link changes quickly, than it is to propagate this information within the complete network. This is because a node, which is reachable within a few hops, doesn't need to know about that link change in a timely manner.

Due to those issues and the overhead when using OGMs for estimating and propagating the link qualities, B.A.T.M.A.N. V will have two major modifications. According to [38, bat_v_elp.c,bat_v_ogm.c] and [39], the task of measuring link qualities between direct neighbors will no longer be done by using OGMs. For this, the Echo Location Protocol (ELP) is introduced in [39]. However, OGMs are still transmitted to disseminate routing information, although with a lower regular interval.

In B.A.T.M.A.N. V, every node broadcasts ELP messages on all available Batman interfaces that are using ELP. The interval is set to 500ms, but like the OGM interval it can be configured. A fixed number of two unicast packets [38, bat_v_elp.c] is sent to a given neighbor in order to trigger the throughput estimation. If a packet gets received, the ELP packet is used register this node and to create an entry in the originator list.

As the link estimation is handled by the ELP, the OGM processing explained in Section 2.1 becomes simpler, but new code for the ELP is added. The code to handle ELP messages is partly implemented and available in [38, bat_v.c,bat_v_ogm.c]. If a new OGM was received, it is first of all checked in the same way like in version IV. The next check is about the announced throughput of the node, as ELP uses the throughput of used hardware interfaces to estimate the link quality. This replaces the TQ mechanism that was one of the most important techniques in Batman IV. The throughput is handled in the same way as the TQ metric. It can be directly measured between two neighbors, but the throughput for a complete path needs also to be known. As the hop with the lowest throughput acts like a bottleneck for the complete path, Batman V simply chooses the lowest announced throughput as the path throughput. This replaces the global TQ for a given node. Like in version IV, this metric is used separately for each interface, and it is used to get the next best hop towards a destination. So Batman V triggers the throughput announcement (echo) by using ELP packets (call). However, the development for Batman V is not completely done yet and thus it is not available right now.

Chapter 3

Modeling the Freifunk network

As mentioned in Chapter 2, the interface *bat0* is the entry point to the mesh network. So every packet using this interface is to be handled by Batman-adv. Here the different kinds of packets are created and sent by using the ad-hoc wifi network or the *mesh-vpn* interface. As mentioned, the high overhead is presumed to be caused by OGM broadcasts. As we're just interested in measuring OGMs, we have to filter them. This can be done with network protocol analyzers such as *Wireshark*[19] and *tcpdump*[20]. However, the measurement has to take place on a real device, operating in the Freifunk network. As mentioned in Chapter 2, the router software is an OpenWrt operating system that is extended with special Freifunk properties. The Open Package Manager (Opkg), that is available with OpenWrt, allows for additional software to be installed. After installing *tcpdump*, it is possible to run first tests. Unfortunately, most of the devices in Paderborn are of the type *TP-Link TL-WR841N(D)* which has just 4 MB of flash memory and a 550 MHz CPU[21]. This impedes measurements because of latencies and limited storage on the device. Since version 0.5.3 of the firmware in Paderborn, the device doesn't offer enough space to install tcpdump or other tools. Therefore, the measurements in the real network take place on several *TP-Link TL-WR1043N(D)* or even more powerful devices. To perform measurements, root access on the device is required. The login by using a password is disabled by default, but a device can be accessed by using an ssh connection with a public key authentication. However, to use this login mechanism, the own public key must be added to the router manually. As all of the devices are owned by individuals, it is impossible to access them by using this procedure. However, there is a rather small subset of three routers, where access is possible. The measurements were performed using tcpdump on the router, combined with

[19] `https://www.wireshark.org`
[20] `www.tcpdump.org/`
[21] `http://wiki.openwrt.org/toh/tp-link/tl-wr841nd`

a pipe to transfer the data to another host. A pipe is a special kind of files, which can be used as a unidirectional data channel. So one program writes to the pipe, while another reads from it. By executing the commands in listing 3, a pipe will be created that is listening on the remote interface *mesh-vpn* and transfers the output to */tmp/pipes/capture*:

```
1 mkfifo /tmp/pipes/capture
2 ssh root@fdca:ffee:ff12:132:6666:b3ff:fe41:0b2e "tcpdump -s 0 ↘
      -U -n -w -i mesh-vpn" > /tmp/pipes/capture
```

While *tpcdump* is capturing packets on the Freifunk device, *Wireshark* can be used to read the pipe as shown below.

```
1 wireshark -k -i /tmp/capture
```

By using various filters it is possible to extract the OGM packets. Table 3.1 shows the result for this sample. Table 3.1 shows the small subset of available nodes in the network. By using an average size of 110 Byte for an OGM, this works out to ∼23 GB of data for each month. The overall number of nodes was about 780. An example for the number of clients during the month July is shown in Figure 3.1. There is one outlier around the July, 4th. This is caused by missing data for that period. On the whole, one can argue that the number of clients always behaves similarly on weekdays. At the weekend the number of clients gets smaller. This is because less people are traveling, working or at school. The last week in July shows

Device (name)	Received OGMs per min.	Sent OGMs per min.
thardes2_batman_dev2	4176	684
c3pb-makerspace	3567	756
northalpha-uplink	5439	792

Table 3.1 – OGM capturing on interface *mesh-vpn*.

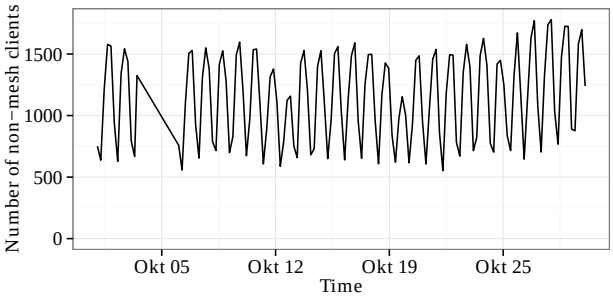


Figure 3.1 – Number of clients in July 2015. The amount of clients is higher on weekdays, as more people are traveling, working or at school. The increase in the last week is due to a big festival in the city.

a higher number of clients, which is because of a big festival in the city. However, the differences between different days are marginal. Basically the number of clients decreases in the night and increases in the morning. The number of clients in the network has a rather small impact on the average size of an OGM, although the OGMs are also used for the client announcement.

3.1 Implementation of B.A.T.M.A.N. IV using a Freifunk network

All necessary parts of the Batman-adv routing protocol were implemented to simulate the Freifunk network in Paderborn. To simulate the Freifunk network I use the simulator OMNeT++, which is a discrete event network simulator. Furthermore, the INET framework is used, which is an open-source model library for the OMNeT++ simulation environment

As the number of OGMs is of most interest, the complete handling of OGMs, the node announcement, link estimation and the route propagation belong to the most important parts. Furthermore, the complete mechanism to announce new non mesh participants is of interest in order to analyze the delays for new clients joining the network. Other parts, like the *distributed ARP Table* feature or the *interface bonding* are completely skipped, as this functionality doesn't have an impact on the OGM processing. My implementation for Batman IV is added to the INET project structure as a new routing protocol.

The simulation contains two parts, where one part is the concrete implementation using C++ and the other one is the network description with the combination of different modules. The overall implementation is close to the one of the real Batman-

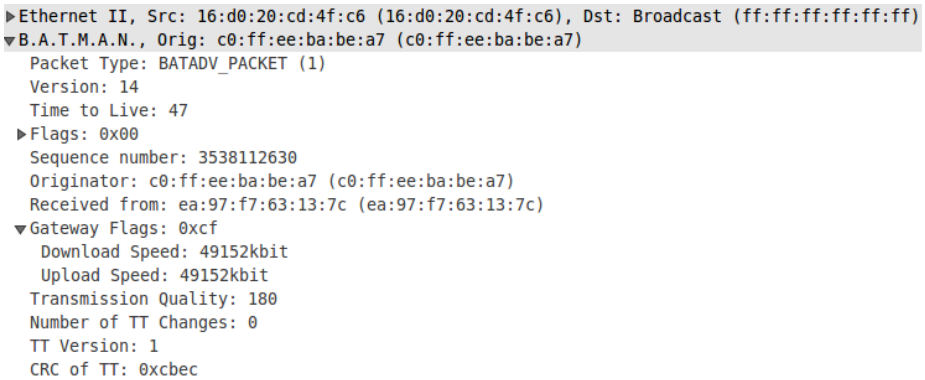


Figure 3.2 – Batman-adv packet in Wireshark. The packet contains all information mentioned in Chapter 2 and also the data that is attached by using the TVLV implementation.

adv code in [15]. However, using the OMNeT++ kernel, it is possible to simplify some components, like a direct interaction with the physical interface. The main module is called *BatmanADVMain* and it inherits the *Ieee80211MgmtBase* class which is part of the INET framework. This class is used as a part of the network in the simulation later on and it is acting on OSI layer 2.

The periodic transmission of own OGMs and the handling of incoming messages, is the most important functionality. Furthermore, the client announcement is related to this feature, as a client gets announced with a new OGM. In order to allow non-mesh clients to participate in the network, the basic functionality of Batman-adv unicast messages is implemented as well. The gateway functionality is also part of the simulation, to enable a regular Freifunk node to discover all gateways in the network and to choose one according to the selected metric. Of course, the *late switch* metric is used in the simulation for this thesis, as it is also used in the real system. All gateways are using the DHCP server functionality which is implemented within the INET framework.

The TVLV feature for OGMs is not exactly implemented, as it can be easily abstracted with the available functionality of OMNeT++. The structure of an OGM is slightly changed for the implementation, to store the TVLV members as well. However, the size of an OGM is calculated according to its content. If, for example, a regular OGM is transmitted and no TVLV data is attached, just the size for the OGM is used for the size of the message. On the other hand, if a new client joins the network and the client information can be attached to an OGM by using the TVLV mechanism, the size for this client including the size for the TVLV header is added to the OGM's size. This results in the same size, as if it would be in the real implementation.

Furthermore, Batman-adv uses several mechanisms to initialize interfaces, that are used for the mesh network. As all interfaces and properties of routers and gateways in the simulation are predefined, all information is gathered during the initialization of the simulation. This compensates the startup process for each interface. The specific configuration for each device is gathered from the omnetpp.ini file.

Batman-adv uses multiple MAC addresses for the mesh, as there are different MAC's for each interface and for the virtual mesh interface. Using multiple MAC's is just a technical detail, as everything gets aggregated into a single one, when the data is made public. To simplify the implementation, all interfaces are operating on the same MAC address during the simulation. As all interfaces are handled individually, this won't lead to collisions or other unexpected behavior.

As described in Chapter 1, a Freifunk network consists of several Freifunk routers, Freifunk gateways and client devices like smartphones or computers. The INET framework already defines different devices for wired and wireless communication.

However, as Batman-adv is a routing protocol and acts on the link layer, the default implementation of the INET framework needs to be changed. To keep the *StandardHost* and the *WirelessHost* implementation of the INET framework untouched, a new module with the name *FreifunkRouter* was created. The module provides radios for wireless transmission and wired interfaces for the connection to the gateways. Furthermore, it contains submodules to support mobility and status changes of nodes. The mobility module is needed in order to place the router at a specific position on the simulation playground. This is important to be able to place non mesh clients next to the node. Otherwise, the client wouldn't be able to establish the connection. The node's status is needed for the delay measurement as the node gets created by the *ScenarioManager* at a specified time. The module to support the gateway functionality is called *FreifunkGateway*, which is similar to the routers description, but it doesn't support wireless communication. The structure of a router is shown in Figure 3.3. In order to use client devices in the network, the default INET module *WirelessHost* is used. The *WirelessHost* is an IPv4 host with TCP, UDP layers and applications which supports one 802.11 wireless card in infrastructure mode. This host is extended with the DHCP client functionality which is also implemented within the INET framework. This allows the client to create and send a DHCP message, after the association process with the node was successful. The DHCP server functionality is used in the *FreifunkGateway* module.

All modules mentioned above are included in the network which is used for the simulation. Since the simulated network should be as close as possible to the real network, extracted data is gathered from the existing network in order to use

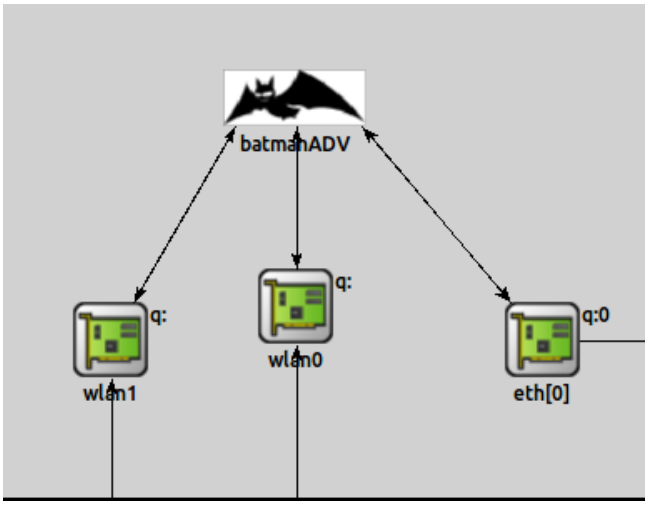


Figure 3.3 – Network structure of the *FreifunkRouter* module. The Batman-adv code receives packets directly from the physical interface. This is similarly implemented in the real world.

them to create a close approximation. Section 3.2 describes this approach and the additional software that is used to solve this task. Using the generated files *omnetpp.ini*, *scenario.xml* and *ffpb_network.ned* it is possible to immediately start the simulation with a correct network configuration.

The INET framework has been changed in some places in order to provide the required functionality. The overall changes are small, but they are required to successfully run the simulation.

To support the creation and deletion of modules using the *ScenarioManager* and the *LifecycleController*, the according code has been changed to create new nodes using the scenario file. The code was primarily ported from the new version 3.0 of the INET framework, which was released during my work on this thesis. As some modules, which are important for my work, were removed and namespaces were changed, I decided to not port my implementation to the new framework as new features are just partly needed.

The second module, that was changed, is the *Ieee80211MgmtBase*, which is an abstract base class for 802.11 infrastructure mode. The original implementation doesn't support modules, which are inactive during the simulation, but are in range of a module that operates using a wireless module to transmit messages. If the disabled module is within communication range and receives a message, the *Ieee80211MgmtBase* crashes in the original implementation.

Furthermore the implementation of *IPv4* has been changed. The original implementation crashes, if a certain module receives a cPacket, which encapsulates an unknown packet type. As Batman-adv uses packets which are unexpected for the IPv4 implementation, the code crashes.

In order to correct those errors, it is enough to simply ignore all of them and forward the packet to the next layer of the module.

3.2 Extraction of network data and generating the model

As the simulation model is used to solve problems in the real world, my thesis requires a model, that is as close as possible to the real world. If the developed model isn't a close approximation of the actual system, the applied changes during the improvements are likely to be divergent and erroneous. Therefore, I use data from the real world to create the network structure and the configuration for the simulation. This data is created by each node in the network and is transported by a user space daemon called the *Almighty Lightweight Fact Remote Exchange Daemon* (A.L.F.R.E.D. or Alfred). Alfred is important, because the daemon is used in Paderborn to share information about the network topology. With this it is possible to visualize

the mesh network by reading the TT and originator table. This information is aggregated, filtered and then distributed within the network. The information, that is transmitted within Alfred, is used by various services like a dashboard [22] or a graphite [23] installation. Both systems are used to create and evaluate statistics. Listing 3.1 shows an example for a JSON file that is transported with Alfred in Paderborn.

[22] http://stats.paderborn.freifunk.net/stats
[23] https://github.com/graphite-project

```
{
   "uptime":811642,
   "statistics":{
      "traffic":{
         "txbytes":37989228,
         "rxbytes":1565058883
      }
   },
   "macs":[
      "66:6a:b3:41:0b:2e",
      "66:69:b4:41:0b:2e"
   ],
   "aliases":[
      "666ab3410b2e"
   ],
   "hostname":"thardes2",
   "mac":"64:66:b3:41:0b:2e",
   "__UPDATED__":{
      "alfred":1415386385,
      "batadv":1415386386
   },
   "neighbours":[
      "c0:ff:ee:ba:be:04",
      "ea:97:f7:63:13:7c",
      "c0:ff:ee:ba:be:02"
   ],
   "node_id":"6466b3410b2e",
   "location":{
      "latitude":51.737165,
      "longitude":8.677225
   },
   "clientcount":0,
   "clients":[
      "64:66:b3:41:0b:2e"
   ],
   "gateway":"c0:ff:ee:ba:be:02",
   "software":{
      "firmware":"0.6~exp20141005",
      "autoupdater":"testing"
   }
}
```

Listing 3.1 – Example for json file transported by A.L.F.R.E.D.

Alfred is not a part of Batman-adv, but it is a separate daemon. It is based on a client server architecture[24] in the network. There is at least one server (master) node acting as a distributed global storage for the network. All server nodes have to announce their presence. A client (slave) establishes a connection to its selected master in order to periodically transmit its data to the selected server node. The communication is done with IPv6 UDP messages. The current strategy to find a selected server that is chosen randomly from all available servers. The different synchronization mechanisms, different packet types and details about the client data exchange are skipped here, as this functionality is not important for my thesis.

In addition to Alfred, there is an independent project that is developed in Paderborn and called the *Batman / Alfred Transmission Collection, Aggregation & Value Engine (Batcave)*[25]. An instance of a Batcave can be installed on every device that connected to the Freifunk mesh network and runs an Alfred server daemon. The main task of the Batcave is to collect and aggregate data that is provided by the Alfred storage and the running Batman instance on the same system. The aggregated data can be exported as a JSON, CSV or XML file. The raw Alfred data can be accessed without special credentials by using a public URL[26]. The Batcave is not made public and so special permissions are required to reach this system. This is done because of privacy issues, as historical data from the Alfred services is available. For example, it might be possible to create movement patterns for stored MAC addresses by using this data. The Batcave has an HTTP interface, which allows queries with a parametrized URL to get the specific information. Using this mechanism, data for all nodes can be downloaded in different file formats like the CSV or JSON format.

To download and process Alfred data, I developed an additional Java program. This tool downloads all Alfred files for known originator MACs in the network and creates configuration files for the simulation. The Macs are gathered from a git repository which is not made public, as it also contains personal data of device operators. The downloaded files are used to generate the network (ffpb_network.ned), the configuration (omnetpp.ini) and the scenario description (scenario.xml) to create new devices in the simulation. For this, the stored MACs, geographic locations and known neighbors are of most interest, in order to have a simulation model that is as close to the real network as possible.

The mesh network is used by non-mesh participants, who get registered at different nodes in the network to exchange data. In order to create a real world scenario including clients, it is important to know how many non-mesh devices are connected to a certain node in the network. The overall number of clients is also stored in an Alfred data set, but it is not possible to extract detailed information

[24] `http://www.open-mesh.org/projects/alfred/wiki/Alfred_architecture`
[25] `https://git.c3pb.de/freifunk-pb/batcave`
[26] `http://map.paderborn.freifunk.net/nodes.json`

about clients that joined and left a specific node during the last Alfred interval. Alfred data is sent every minute. As a result of this, there could be a scenario within this one minute interval, where one client leaves and another client joins a particular node. The overall number of connected clients would be the same after one minute, but it is not possible to model the leaving and joining clients in the network. Figure 3.4 shows such an example. The TTVN is just changed once per OGM interval and when it gets incremented from version 6 to 7, one client timed out and another client joined the network. The number of clients remains the same and the information about the time out and the new client gets lost. As Alfred also transmits the MAC addresses of all clients, the information can be recovered by comparing the MAC addresses of two records. To collect this data, a new functionality was added to the Batcave. Here, the MACs of non-mesh clients are checked every minute for each node, and the information is temporarily stored to compare both result sets after the next minute has passed. By comparing the client MACs, it is possible to indicate clients that left the node and clients which joined the node during the last interval. The results show two csv files for each node indicating the number of client changes for the *added* and *removed* event. Listing 3.2 shows an example for the structure of such a file.

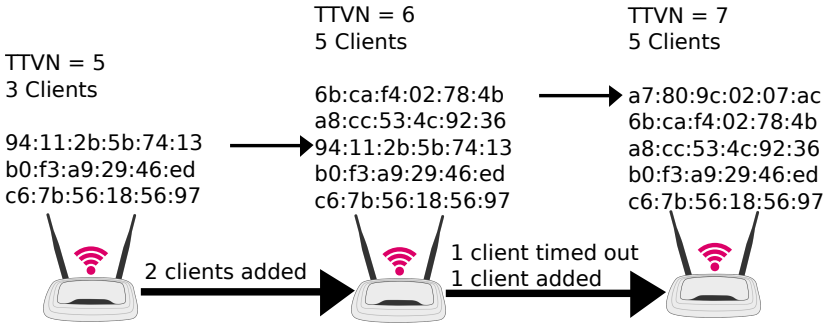


Figure 3.4 – Example for client announcement. As data is sent every minute, there are scenarios where a client event gets lost.

```
1 f8:1a:67:36:4c:8e,2015-06-20 17:17:00,2
2 f8:1a:67:36:4c:8e,2015-06-20 17:18:00,0
3 f8:1a:67:36:4c:8e,2015-06-20 17:19:00,1
```

Listing 3.2 – Example for the result file of clients, that were joining a certain node in the network. The file is kept simple and it just contains the node, a time stamp and the number of clients for that interval. There are two files for each node. One file contains all client that joined the node and the other one contains client that left the node.

With the gathered client events, it is possible to generate a scenario that can be used later in the simulation to dynamically create and destroy non-mesh participants in the network. This leads to a client behavior which is very close to the real world. Gathering exact movement data of client devices is not possible due to data protection reasons. The movement of nodes is not a part of the simulation, as Freifunk nodes are stationary and not moving within the network.

If a new Freifunk node gets installed in the real world, the owner is able to set the position of the node with GPS coordinates. However, setting the GPS position is optional and not required. Furthermore, the coordinates are static values in the node's configuration file and not automatically updated by some GPS sensor or by IP location. Thus it cannot be ensured that the stored position is a correct one, as the node could be moved to another position afterwards or the position was wrong from the beginning. All known GPS positions of available nodes are used for the simulation model. One reason for this is the possibility for wifi mesh connections if two nodes are in range of each other. Furthermore, not every node is directly connected to the gateway backbone structure (see Figure 2.3). In order to take those nodes into account, a neighbor is required, as otherwise they wouldn't be able to participate in the network. Another reason is a higher network load on the wireless channel, as more devices have to share that dedicated one. Having a higher network load leads to more collisions on the channel and the distribution of OGMs from mesh nodes becomes harder. As the position of a node might be wrong, or there might even be no position, a second parameter is taken into account in order to place nodes nearby, like in the real world. The Alfred record contains information about the neighbors of a node. A neighbor could be another node or a non-mesh client. All Freifunk nodes are already known, so this neighbor list can be filtered to just get the real Freifunk nodes, which are connected to this particular node. Using this information, it is possible to place all nodes from the neighbor list within the transmission range of a node. If this neighbor also uses a direct VPN to one of the gateways, this connection is also created. In fact, this doesn't ensure a one hundred percent mapping to the real network. There are also devices without a GPS location and without any direct neighbors. The positions of those devices are completely unknown. Furthermore, there are also devices in the network with special antennas, which allow the node to direct the signal into one direction. Figure 2.1 gives an overview about all available devices. With this, an asymmetric link could be created, which is not covered by this approach. However, the number of directional radio systems are negligible and therefore not taken into account. Since not all nodes in a wifi mesh are mesh only nodes, the neighbor list is also checked for nodes which are connected via a VPN connection. According nodes are connected to the gateway infrastructure. With this, scenarios where VPN and wifi mesh connections are present are also covered.

The number of gateways is static and at the moment eight gateways are present in the Freifunk network. Those gateways are connected to each other in a full mesh topology, meaning each gateway is connected to each other. According to the gateway selection process of Batman-adv, explained in Section 2.4, the number of connected gateways per node should remained more or less the same for each gateway. However, the VPN connections need a high amount of CPU time which is a factor that should be considered. As the hardware equipment of all gateways is different, not every gateway is able to handle the same amount of nodes. Due to this problem with hardware limitations, all gateways are configured using the SaltStack software[27]. With this, there is a router distribution which is predefined and shown in Figure 3.6. The shown distribution was initially used for the simulation model. Due to the new insights gathered from the model validation and the improvement, the gateway distribution was slightly changed. Those changes and the justification are explained later in this chapter.

[27] http://saltstack.com/

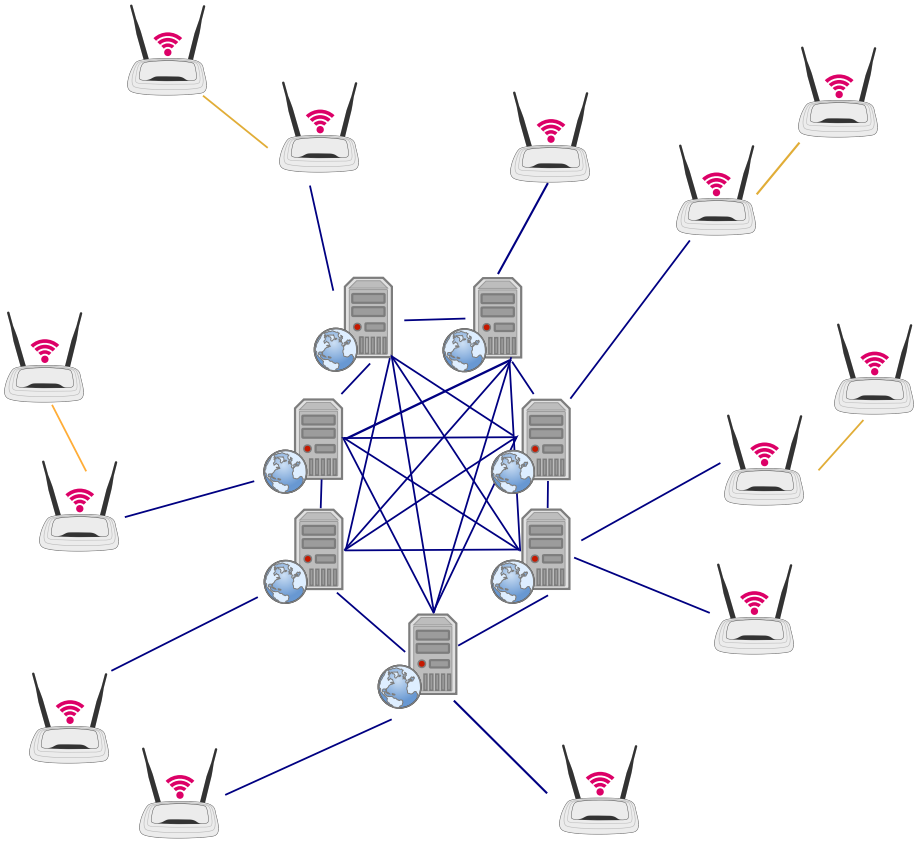


Figure 3.5 – Example for a small Freifunk network

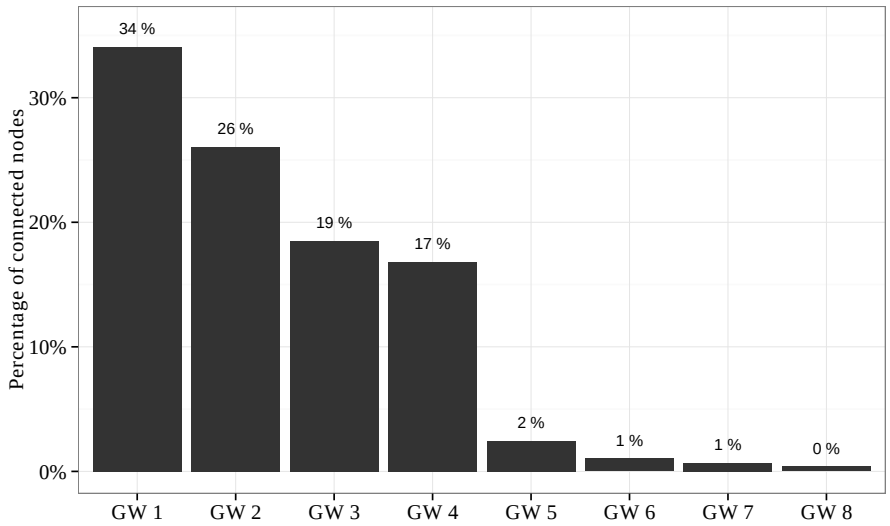


Figure 3.6 – Connected nodes per gateway in Paderborn. By using the SaltStack software, the distribution is predefined.

The developed Java software for this thesis is configurable, as it allows to specify the number of nodes and the number of gateways for the simulation. By using the number of gateways, it is also possible to specify the overall amount of nodes that should be handled by one specific gateway. This feature was added in order to support the initial node distribution and different cases for the optimization. Furthermore, it allows to generate all client data, so the simulation automatically creates non mesh clients according to the chronological sequence gathered from the Batcave data. Moreover, all network and client scenarios for the improvements, which were realized within this master thesis, can be created with this tool by using the start up parameters. The parameters are explained when the application gets started and they are skipped in this document.

3.3 Model validation

The simulation model is used to solve problems in a real network. Hence, if the developed model isn't a close approximation of the actual system, the applied optimizations are likely to be divergent and erroneous. Thus, the first goal of the simulation is to check whether the model behaves like the real network, and in the second step, to check whether specific changes to the protocol lead to a lower routing overhead without violating the required functionality of the protocol or the usability for the users.

The overhead and the usability can be defined by a selection of performance metrics which are measured during the simulation. The following metrics are based on [13, 16–18, 40] and were chosen to be valuable when measuring the performance of Batman-adv in the Freifunk scenario of Paderborn:

Routing overhead: The total number of routing packets transmitted during the simulation.

Packet aggregation: Number of OGMs that are aggregated into one packet.

Delay for node announcement: Delay until a new node is known by all other nodes in the network.

Delay for client announcement: Delay until a new client is known by all other nodes in the network.

The objective of this master thesis is to reduce the routing overhead using Batman-adv in a Freifunk network. Therefore, the overall routing overhead is of most interest. The packet count is measured by each node in the network for the mesh-vpn interface. Here, both sent and received packets are counted separately, as the number of sent packets is assumed to be much lower than the number of received packets.

Batman-adv uses aggregation for OGMs to reduce the overhead by sending one big packet instead of several small ones. This metric is also important, as aggregation is related to the OGM count. So the overall packet count for OGMs could be the same in the simulation and in the real world, but the aggregation could be completely different. Thus, the overall aggregation of packets is also needed to verify the routing overhead as well. However, the aggregation is just interesting for the model validation. As an improvement might complete change this mechanism, it cannot compared with others to draw conclusions from this.

Reducing the OGM packet count affects almost all features of Batman-adv which were explained in Chapter 2. Therefore, it is not enough to simply set the interval of OGMs to a much higher value, as all other features shall remain intact and be unaffected. Changing the OGM behavior might affect the time until a new node is known to all other originators within the network. This is quite critical, as routing is not possible during this time, because paths might not been known and evaluated yet. Due to this, the delay for the node's announcement should approximately be the same like in the model and it shouldn't increase too much during the optimization. The same holds for the announcement of non-mesh participants, like user devices, as smartphones or computers. As an OGM is required to add the client permanently to the global TT, the delay for the announcement of new clients should approximately remain the same. As mentioned in Section 2.2, the information about the new client could be directly attached to the OGM. The second possibility would be just

an incremented TTVN in the OGM without the new client information. In this case, a *TT-Request* is sent to the originator in order to get the knowledge about the new client. Here, the routing of the request is only possible, if the network is fully established. Both delays are measured and collected by an additional module called *StatisticsController* in the simulation. Whenever a node stores a new originator or a new client, this module gets called in order to gather this information for the statistics. To check the information, the MAC address of the new node or the new client is used to distinguish between different participants.

The measured values for the routing overhead are compared to the values gathered from the real world. The delay measurements are compared with theoretical calculations, as those delays are not measured in the real network. With this it is possible to check whether the simulation model behaves in almost the same way as the real implementation, even if the sample from the real world is rather small.

However, there are other potential measurements like the *packet delivery ratio*, which describes a ratio between the number of originated packets and the number of received packets, or the *Path optimality*, which checks whether the optimum route has always been chosen [40]. It is not possible to get information about the *packet delivery ratio* in the real world, as this information is not measured on the real devices at the moment. Of course, the router's firmware could be changed in order to capture this data or at least to approximate the real value, but the cost-effectiveness ratio is quite high and the added value for the simulation is not assumed to be significant. Furthermore, the environment has a high impact on this factor and buildings, and other obstacles are not part of the model. The *Path optimality* is also related to the delay inside the network. If a non optimal path is chosen, the delay of a packet will also increase and this is already measured, as mentioned above. In addition, Batman-adv has the TQ metric to rank different links in the network and it chooses the path based on this metric.

3.3.1 Simulation preparations

To simulate the Freifunk network I use the simulator OMNeT++, which is a discrete event network simulator. Furthermore, the INET framework is used, which is an open-source model library for the OMNeT++ simulation environment. OMNeT++ allows to define multiple modules (.ned file), which can be used in a hierarchical way to build up networks. Combining different modules and connecting them to each other leads to a description of a network. However, the real functionality comes from the logic implementation using C++ code.

The INET framework [28] is a model suite for wired, wireless and mobile networks. It already contains various models and implementations for protocols like TCP, UDP

[28] `https://inet.omnetpp.org/`

or Ethernet, the network stack for IPv4 and IPv6, and furthermore it provides mobility support for objects and nodes in the network.

The implementation for this thesis uses several features of the INET framework, like the support for mobile nodes, wired and wireless communications and implementations for Ethernet, UDP and DHCP. The Batman-adv implementation is added to the INET framework as a new routing protocol, which acts on the link layer. In order to make the INET framework able to support all required functionality, some modules have been changed. Those changes are described in Section 3.1.

OMNeT++ uses three different files to describe the network and all parameters. All three files are generated using the developed Java software (see Section 3.2). The structure of the network slightly depends on the metric that is measured. For example, the network definitions is changed for the node delay measurement. To measure this metric, a new node has to join the network at a certain time, which leads to additional entries in the network and scenario description for the simulation. However, all simulations require to have a fully established network, which means that all originators have to know each other and hence routing is possible. Without a full established network, packets might not be processed and dropped, as some links might not be considered as bidirectional or links are not even known yet, which is not the case in the real world. A warm up time is used in order to avoid this problem.

The measurements are performed after the warm up time is over. The simulation time is set to 60 seconds plus the warm up time. All simulations are repeated for 50 times with different seeds.

Every node works with several random delays in order to make the nodes not to work simultaneously. Otherwise, this would lead to a lot of collisions, as all nodes start to send packets at the same time. Therefore, each node starts to schedule its first OGM with a random delay of up to five seconds. When a new OGM was sent, the next OGM is planned to be sent five seconds in the future according to the OGM interval used in Paderborn. In addition, the real implementation of Batman-adv applies a jitter to each scheduled OGM which is about 40 ms. A further delay is applied when an OGM gets forwarded by another node. Here, the message gets delayed for at most 20 ms.

All measurements take place with 779 nodes in the network if not stated otherwise. Those are nodes using a VPN connection, and of course nodes just using a wifi mesh connection. Whether a node is connected using a VPN is again determined by the Java software. Like in the real world, there is only one wireless channel used for both infrastructure and ad-hoc network. Due to the network data, which was recorded and aggregated for this thesis, detailed client information is available and used for the simulation. Thus, a huge number of clients is created during the warmup time to have a full established network with a number of clients before the real measurement starts. During the measurement, all further clients are created

according to the real network data. Of course, new clients are also announced with OGMs and the TT of all nodes gets updated using the OGMs and the TT messages explained in Section 2.2. However, by using the Batcave, I get data for clients within an interval of one minute. Due to this, there is always a big amount of clients that is added after a simulation time of 60 seconds. Another possibility would be to spread new clients over 60 seconds. Creating all clients in the beginning of a 60 second interval is not realistic. However, the simulations were performed with the first approach and due to constraints in time, the second possibility has not been used.

3.3.2 Routing overhead

The estimation of the routing overhead can be divided into two parts. On one side there is the number of packets which were sent and on the other side there is the number of packets which were received. Furthermore, I use the number of aggregated packets, as this metric is in close relation with the routing overhead. Together, they supply the data for the overall packet count per node. The following plots just consider data from nodes which are connected to some gateway by using a VPN connection. Nodes which are just connected with a wifi link using other nodes are not considered as the number of packets for the VPN connection would be zero. Furthermore, all gateways were removed from the plots. A gateway is hosted in a data center. Therefore I don't consider the traffic at this place.

Like in the real world, the same overall number of connected nodes per gateway is also used in the simulation. Figure 3.7 shows the distribution per node. Unlike Figure 3.6, the distribution within the simulation considers only four gateways for the model validation. This is because 96 % of all nodes are connected to those four gateways. As the measurements could only be performed on three of 779 nodes, the probability to get a node that is connected to a gateway with a low number of connected nodes is low.

Figure 3.8 shows the send count metric in the form of an Empirical distribution function (ECDF). It compares the simulation results with the data gathered from three nodes in the real world. The solid lines show the packet count based on the sample data, that were gathered from the real world by using three different nodes and the mean of running the simulation 50 times. The dashed lines represent 95 % of the measurements which are around the mean. The amount of sent packets in the simulation is below the measured data, but it behaves in a similar way. The steps in the measured data are due to the fact that only three values are present. Thus, the course of the curve is an approximation, as it is created by a rather small sample.

Nevertheless, to mathematically describe the discrepancy between observed values and the values gathered from the model, usually general statistic tests are applied, like the two sample Kolmogorov-Smirnov test (KS-test) or the Mann-Witney

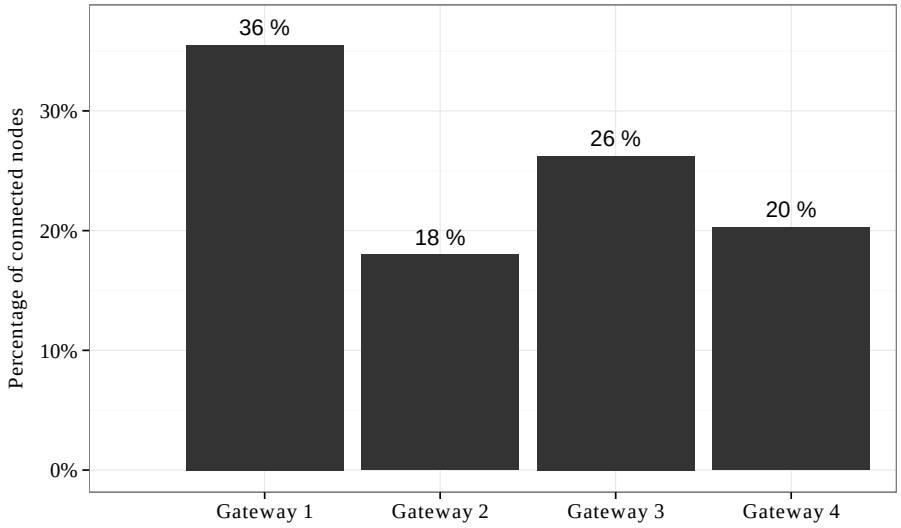


Figure 3.7 – Nodes per gateway during the simulation. Some gateways were removed as they only serve a negligible proportion of nodes.

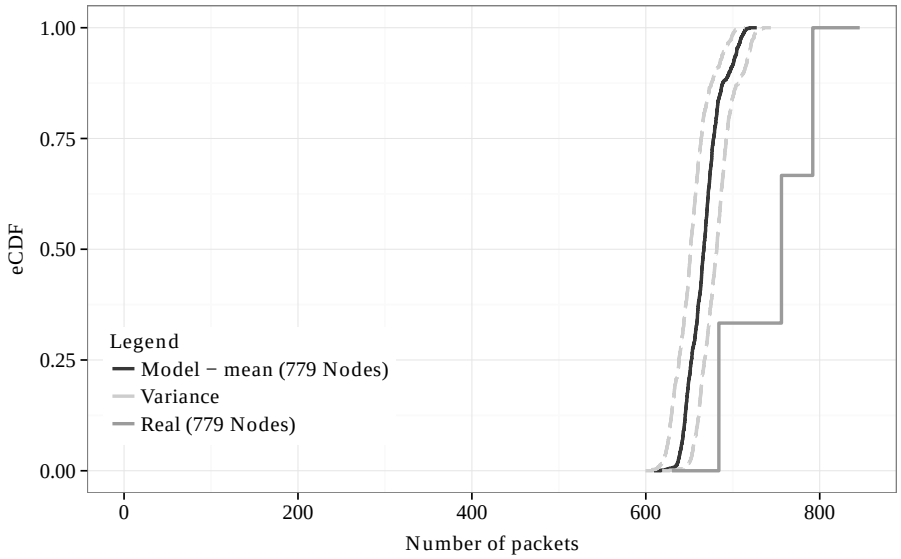


Figure 3.8 – Sent OGMs in the real world and in the simulation as an empirical CDF. The dashed lines represent the interval for a variance of 95 % around the mean.

approach. Those tests belong to the group of Goodness Of Fit (GOF) tests. However, according to *Law* [41], these tests are usually not very accurate for a small sample size. The reason is, that those tests are not very sensitive to subtle disagreements between the data and the fitted distribution. All the tests mentioned above are based on the null hypothesis that the system and the model are somehow the same. As the model is an abstraction and approximates the real world, this assumption is not true. Due to the arguments mentioned above, the author of [41] states that the tests are not useful directly. He suggests to ask whether the differences between the system and the model are significant enough to affect any conclusion derived from the mode. To measure the differences between the system and the model, *Law* suggest to compute numerical statistics from the real world observations and the corresponding statistics from the model output. Both results can be compared afterwards. The problem is the small sample that is used for this validation. In fact only data from 0.12 % of all nodes is known. Based on [41], I'll stick to the graphical approach in order to compare the model with the real system because of the rather small sample size.

The received packet count metric is shown in Figure 3.9. Again, the solid lines represent the real world and the model. The dashed line represents 95 % of the measurements which are around the mean. It is noticeable that there are multiple steps in both lines. Those steps can be justified by taking the gateway architecture into account. Due to this, a gateway node has a huge amount of nodes, that act as a

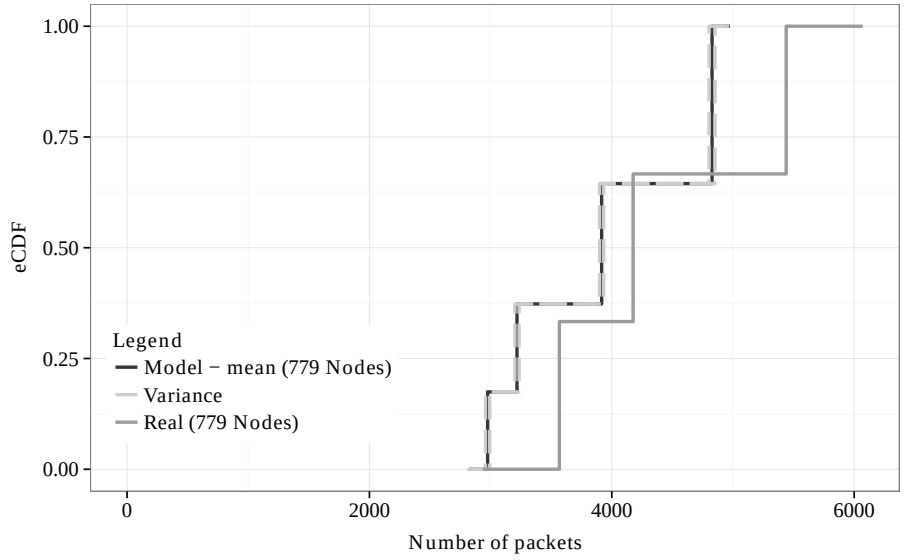


Figure 3.9 – Received OGMs in the real world and in the simulation as an empirical CDF. The dashed lines represent the interval for a variance of 95 % around the mean.

direct neighbor for this particular gateway. Those nodes are reachable within one single hop and based on the Batman routing, all OGMs from a single hop neighbor get forwarded independently from the resulting TQ value. Due to this, a gateway node aggregates and forwards almost all OGMs that were received from direct VPN neighbors. A part of the packets from direct neighbors gets dropped, as those packets could be duplicates or the routing information is outdated and so on. As all gateways are connected to each other, they receive a lot of packets from each other. Here the same situation occurs. If a packet wasn't received from the next best hop towards a destination, the packet simply gets dropped and is not considered any more. As there is only one best next hop for a certain destination, a lot of packets gets dropped at the gateway nodes. Due to this, there are steps in the data which are visible in Figure 3.9. However, as in Figure 3.8, the overall number of packets is slightly below the data from the real world.

Figure 3.10 shows the aggregation metric of OGMs for the sent packets in form of an ECDF. The real worlds data was extracted at the same time as the packet count was measured in order to avoid day specific behavior. There is a high number of packets which were not aggregated. This happens because Batman-adv doesn't aggregate OGMs which were created on the node itself. Furthermore, the curve of the model and of the real world are slightly different. This difference is assumed to be caused by the number of clients, which are present in the network. Due to the interval of the Batcave, which is set to one minute, all clients for the next minute are joining the network at one time. This temporarily increases the amount of data that

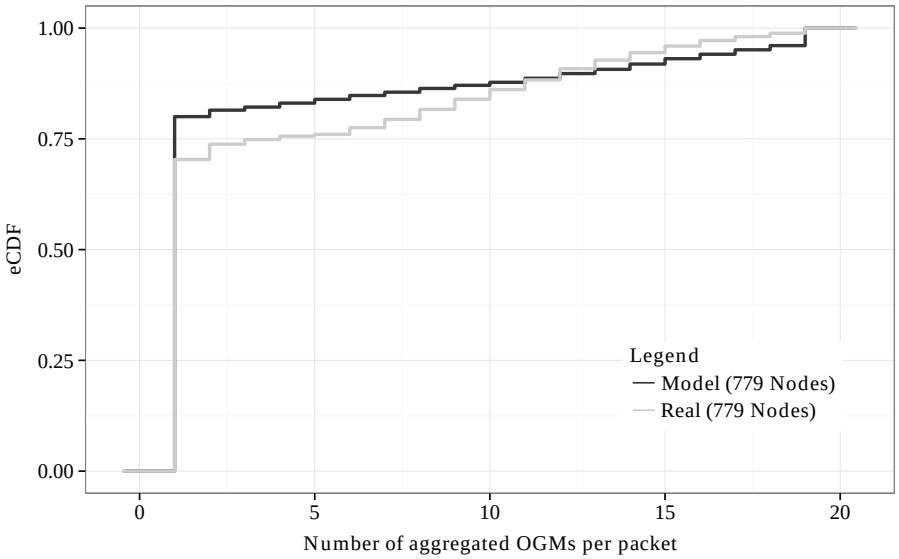


Figure 3.10 – Aggregated OGMs in the real world and in the simulation as an ECDF. The plot shows the aggregation for sent packets.

should be transported with a single OGM. In this scenario, the aggregation differs from the one in the real world.

A node usually receives OGMs from a gateway node. A gateway has to handle a high amount of OGMs and therefore almost all packets get aggregated. Therefore, the plot for the aggregation metric of received packets is skipped.

3.3.3 Delay: Node announcement

The delay to announce a new node is measured by using one new node which joins the network at a certain time in the simulation. The new node is created after the warmup time is reached and the simulation continues until the new node is known to all other originators. This check is based on the MAC address of the new node.

As mentioned in Chapter 2, knowledge about a node gets propagated by using OGMs, and based on the collected information, the OGM gets forwarded or dropped. Hence, it is required to collect enough information before a node is considered to be a potential router. As mentioned in Section 2.3, this estimation is based on the TQ metric and on neighborhood information. Figure 3.12 shows an example flow for a new node which joins the network. However, an OGM only gets forwarded, if certain tests are successful. One of those tests is a check for the TQ value, which needs to be greater than zero. Based on Equation (2.2) for the EQ and RQ values, the TQ will be zero after the first OGM of a new node was received as the EQ becomes zero and hence $TQ_{local} = \frac{EQ}{RQ} = \frac{0}{>0} = 0$. However, every OGM from a direct neighbor gets forwarded, independent from the resulting TQ. Nevertheless, the OGM will be received at the next hop with a $TQ = 0$ and the packet gets dropped during the router checks. Irrespective of whether the OGM gets dropped at this place, the new node still gets registered at those nodes. This is done during the preliminary router checks, mentioned in Section 2.3.

Figure 3.11 shows the announcement procedure of a new node as an ECDF. The plot shows the amount of nodes that have successfully stored the new node in its originator table. The new node joins the network at simulation time T=50 and starts to send its first OGM with a delay of at most 5 seconds. This first OGM is received at a connected gateway which performs, all checks and finally aggregates and rebroadcasts the OGM as it has been received from a direct neighbor. Before the OGM gets forwarded, there is a small delay of at most 20 ms which can be ignored here. As a gateway has a huge amount of nodes that are connected by using a VPN connection, there is a first big rise of the graph at simulation time T=∼55 seconds. However, due to the calculation mentioned above, the new TQ value is 0 after this hop and thus the OGM gets dropped at the next hop and is not forwarded. From here, an OGM only gets forwarded, if the TQ value becomes greater 0 and if the according routing information for the originator are stored. Based on Equation (2.3),

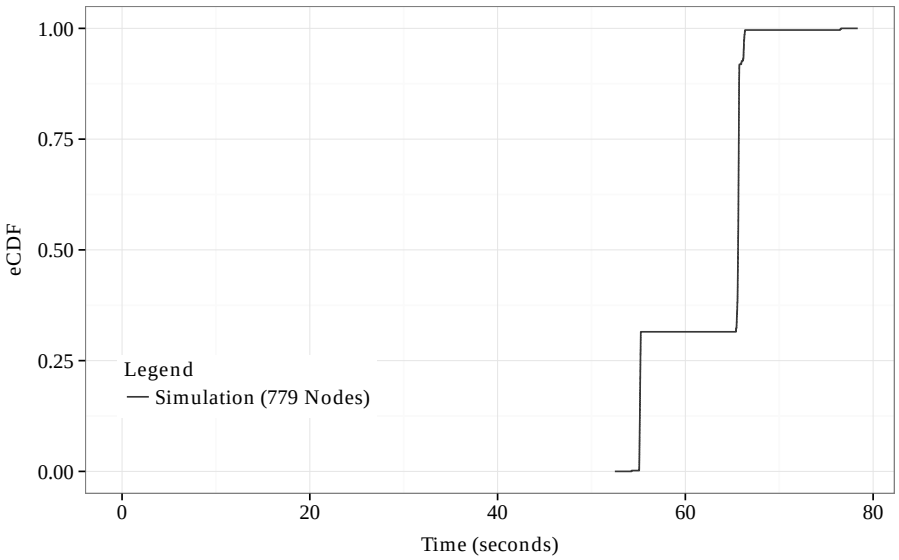


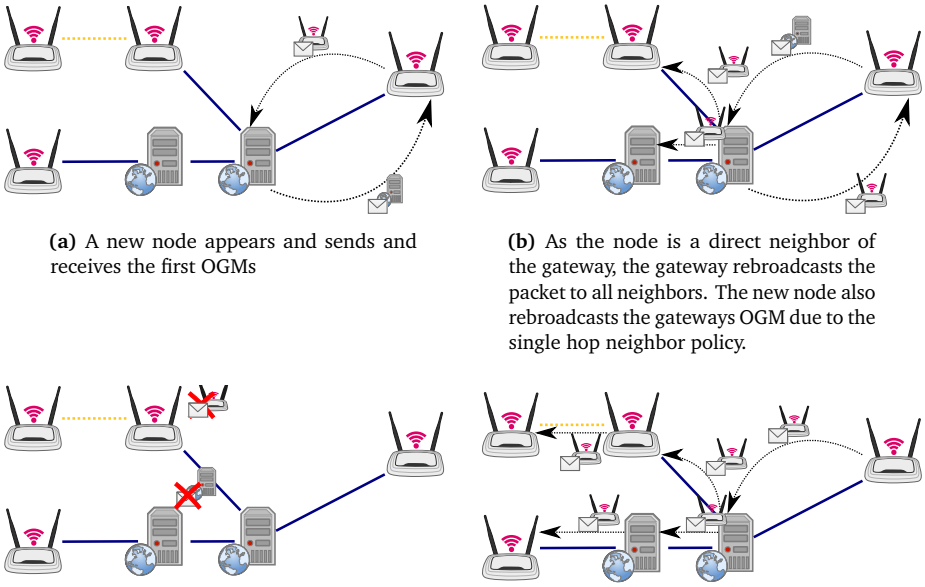
Figure 3.11 – Delay for the announcement of a new node as an ECDF. A random sample out of 50 runs is shown.

a direct TQ value can only be computed for one-hop neighbors. Thus, the global TQ value is important at this place, which is the TQ value stored in the received OGM and the TQ for the neighbor like shown in Equation (2.4). As the new node joins the network at a time where the complete network has established, all paths are fully rated and known to all nodes. For this reason, the neighbors TQ is already greater than 0. Hence it is enough to receive a TQ value greater than 0 to result in $TQ_{local} \times TQ_{received} = TQ_{global} > 0$. Due to the OGM interval of five seconds, it takes up to ten seconds until the first neighbor has enough information to result in an TQ greater than 0. This is because an OGM needs to be rebroadcast to get an EQ value greater 0. The last increase in Figure 3.11 are all other nodes, which are connected by an additional wifi link. Figure 3.12a and Figure 3.12b show an example for this scenario. In Figure 3.12d, the new nodes sends its own OGM to its one hop neighbors, which is just a gateway in this scenario.

Based on the explanations above, the measured values are within the theoretical variations of the delay for a new node that joins the network.

3.3.4 Delay: Client announcement

The previous subsection described and justified the delay that occurs, when a new node joins the network. A device like a smartphone is not part of the mesh and hence it doesn't send OGMs, as it just connects to a node. Afterwards its the node's task to announce the client's presence in the network using the TT mechanism, which was



(a) A new node appears and sends and receives the first OGMs

(b) As the node is a direct neighbor of the gateway, the gateway rebroadcasts the packet to all neighbors. The new node also rebroadcasts the gateways OGM due to the single hop neighbor policy.

(c) The devices which are connected to the gateway are not rebroadcasting the OGM as the originator is not direct neighbor and the TQ value is zero. Thus it takes approximately 10 seconds to gather enough information in order to calculate a $TQ > 0$

(d) After the TQ gets a value greater zero, the OGM gets forwarded.

Figure 3.12 – Delay for nodes in an small network.

described in Section 2.2. Thus, the effect with the node delay doesn't apply and the propagation should be much faster. Using the network scenario shown in Figure 3.5, it is possible to approximate the delay for a new client.

Figure 3.14 shows the ECDF plot for the client propagation delay when 779 nodes are present in the network. The node appears after the simulation has passed the warm up phase, which is at simulation time T=55 s. The client appears next to some predefined router in the network and starts the association process which takes about 1.6 seconds. Batman-adv assumes that a new node in the network starts to broadcast a packet like a DHCP discovery message to reach a DHCP server. With this, the new node gets added to the local TT of the first node. To summarize, the client gets registered at the local node after approximately 2 seconds. Changing the local TT leads to an incrementation of the TTVN, which is announced with the next OGM interval. The new client gets registered at the other nodes in the network. Here, the process of creating and scheduling OGMs is important. A simplified flowchart is shown in Figure 3.13. When a new OGM gets scheduled, the complete packet is created and added to the queue of scheduled packets. With this mechanism, the TVLV data gets attached to the OGM about five seconds before the actual packet is sent on the physical interface. This means that the attached information has already an

age of approximately five seconds according to the OGM interval of this simulation. So the incremented TTVN value is not sent with the next OGM, but together with the packet after that, as the next OGM is already prepared and scheduled. This leads to an additional delay of about five seconds due to this mechanism. As the OGM interval is set to five seconds in the real network, it might take up to 5.6 seconds in theory until the OGM is sent. This is because of the random delays when sending and forwarding OGMs. With the next OGM that gets scheduled, the client is announced at the direct neighbors. Those are usually gateways and potentially wifi mesh neighbors. Since the network is fully established, the OGM gets forwarded, as all checks are successful and the TQ is greater than zero. The forward delay is ~20 milliseconds. All gateways are connected to each other and each gateway has a number of nodes which are connected directly to them. Due to this, the plots course is increasing, as all connected nodes and all gateways receive the knowledge about the new client at approximately the same time. This procedure is repeated and as all gateways are rebroadcasting the OGM then, all directly connected clients receive the information about the new client. In a last step, all nodes which are just connected via a wifi connection obtain knowledge about the new client and thus the network is flooded with the new client information in a short time. The overall delay is negligible, after the first OGM has been sent.

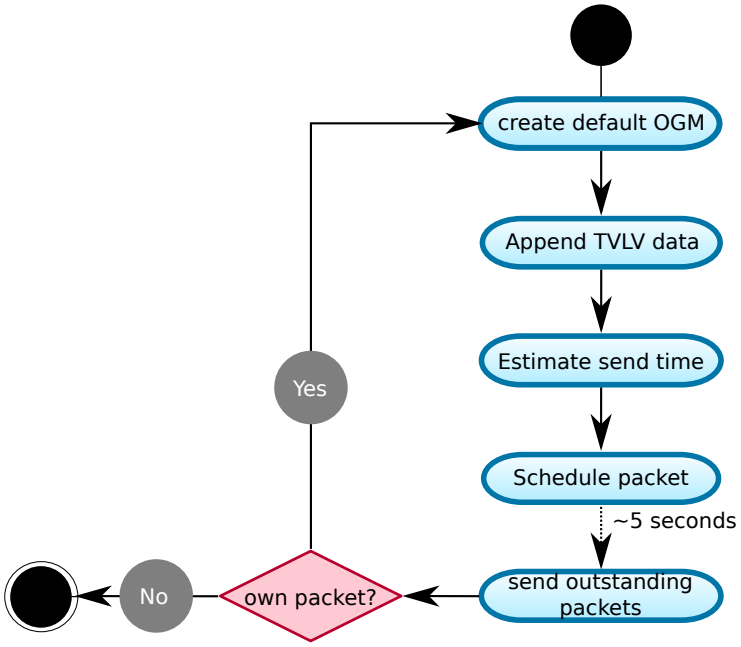


Figure 3.13 – Flowchart for the scheduling of a new OGM. The TVLV data gets attached before the OGM gets scheduled with the OGM interval.

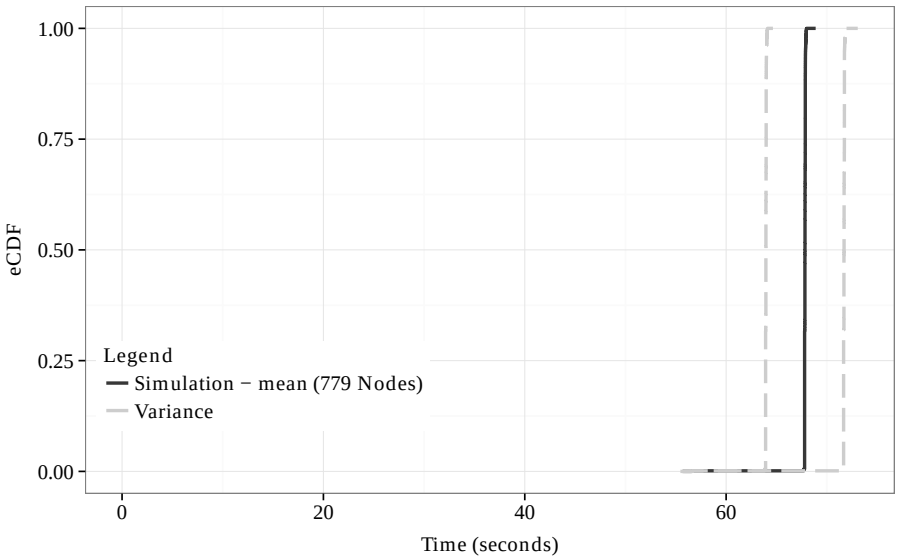


Figure 3.14 – Delay for the announcement of a new non-mesh client as an ECDF. Due to delay when forwarding OGMs, the new client information gets propagated very fast.

Based on the explanations above, it takes about 5 seconds until the next scheduled OGM is sent to the nodes' one hop neighbors. The forward delay at each node is about 20 ms and therefore, the delay of the measurements above are within the band.

Chapter 4

Improvements for B.A.T.M.A.N. IV in Paderborn

The task is to develop different improvements for Batman-adv in the Freifunk scenario of Paderborn. Those improvements focus on the overall amount of exchanged OGMs for each node. All changes to the algorithm and to the network are applied without an impact on other metrics, like the node and client, delay. This thesis includes five different possibilities to improve the OGM overhead in the Freifunk scenario of Paderborn.

The first idea is to split the network into multiple subnetworks (clusters) with a gateway infrastructure for each of them. With this, an OGM just gets forwarded within one subnetwork, but it is not forwarded to gateways of another network. It is supposed that the number of OGMs declines, as less nodes are part of each network and therefore less OGMs get rebroadcast. Furthermore, this approach provides an opportunity to create new communities, by moving the field of responsibility for certain parts of the network, to other Freifunk participants.

Based on the results of the cluster approach, another change has been applied. It is striking that the amount of OGMs depends on the overall amount of nodes that are connected to a certain gateway in the network. Therefore, the distribution of nodes across all gateways has been changed, to have them evenly distributed. This network change demonstrates, that a wrong clustering approach may even lead to a worsening of the OGM overhead.

The third optimizations changes the behavior of gateway nodes. A gateway becomes responsible for the complete routing in the network. With this, a node has knowledge about it's local wireless environment and about the gateway infrastructure, but it doesn't know about other nodes in the network. Hence, the forwarding mechanism of OGMs is changed, which leads to a major reduction of OGMs.

The fourth optimization changes the number of VPN connections that is used by a single node to be connected to the gateway infrastructure. The amount of VPN connections is reduced from two to one. By using more than one VPN connection, a lot of packets get dropped, as they are duplicates. Setting the VPN connection number to one should lead to a lower packet count without changing the other metrics. This optimization has also been implemented in the real world. New measurements from this optimization is compared to the simulation results.

It turns out, that setting the VPN connections to one leads to a major reduction of OGMs. As the first improvement not only reduces the OGM overhead, but also results in other advantages caused by multiple smaller networks, both improvements are combined in form of a fifth improvement.

4.1 Improvement: Network splitting and gateway distribution

As explained in Chapter 2.1, Batman-adv has local knowledge about each node in the network, and based on the information contained in an OGM, the next best hop towards this destination is evaluated by using the TQ metric. As each OGM is transmitted every five seconds and afterwards rebroadcast, the network is flooded with a high amount of OGMs.

The current implementation uses several gateway nodes, which are connected to each other using a Batman-adv link. Therefore, OGMs are exchanged at this point and the complete network is fully connected and there are no gaps. The optimization takes place at this point. However, all subnetworks are still connected to each other by using the gateway backbone, but those links are not used for the transmission of OGMs any more.

It is not possible to just split the network in arbitrary subnetworks in order to reduce the OGM overhead. It necessary to keep the functionality of the routing, node and client advertisement and other features, like the client roaming, untouched. Due to the client roaming, it is not possible to move two nodes into different subnetworks, if they are connected to each other using a wifi link, as they won't understand each other. In such a scenario, the client connection gets lost, if it travels within the network. This is because of the firmware configuration that was explained in Chapter 2. Here, each mesh network has a unique mesh ID, that is used to identify the network the node belongs to. Thus, if there are nodes with different mesh ID's, roaming is not possible any more.

Looking into the future, there might also be other nodes which are not there today. So there might be a gap in the wifi area today, that might be filled after a while. Thus, even nodes that don't have a direct wifi connection, but which are so

close to each other that this wifi gap might be filled at some time, are not allowed to be placed in different networks. So the task is to build clusters in a way that the scenario mentioned above will never happen. To solve this task, it is important to know about the physical positions of all nodes. Based on Chapter 3, this information is already partly present due to the Alfred data. Furthermore, it is necessary to know about the geographical location of Paderborn and its surroundings. Usually, there is less civilization between two villages, as those areas are usually used for agriculture and so the probability for new Freifunk nodes is quite low in this area. Figure 4.1 shows the distribution of nodes across Paderborn. There are also differences in the altitude as for example Büren is below Paderborn and the surrounding area is quite hilly, so a wifi link between those two locations is very hard to realize and not planned to be established today. Section 1.1 describes the Freifunk project and its intension to build community controlled and decentralized networks. As there are

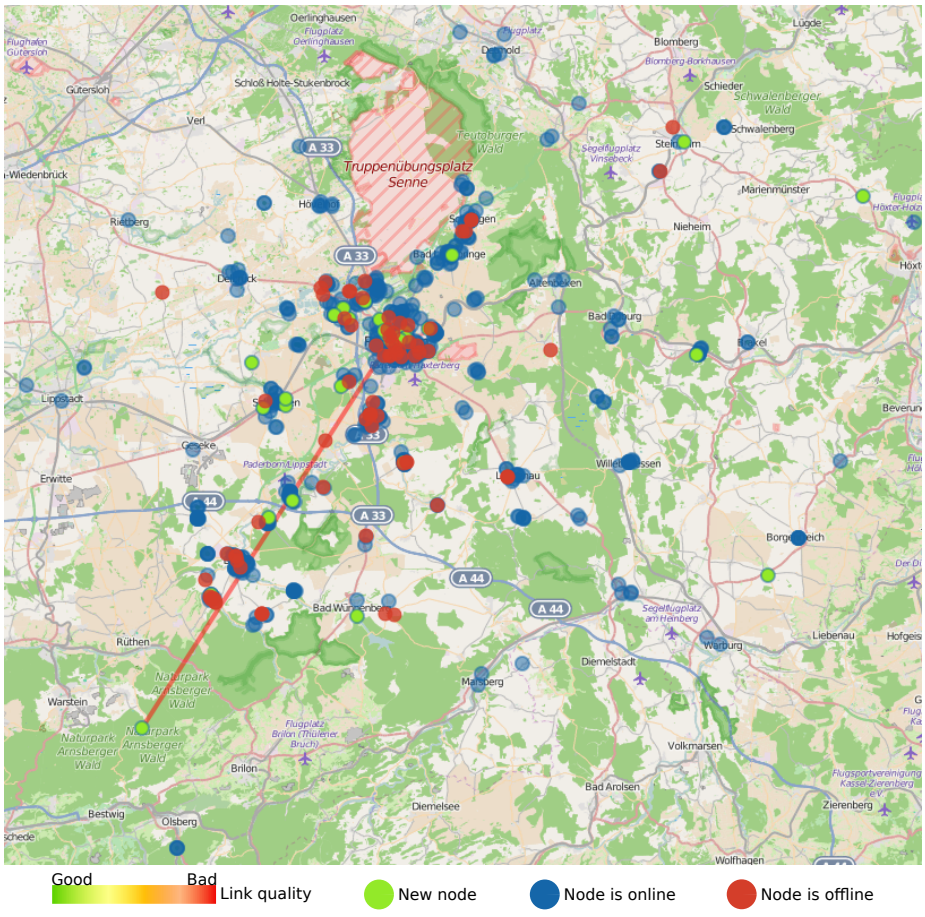


Figure 4.1 – Nodes distributed across Paderborn - ©OpenStreetMap contributors

already large collections of nodes visible, which are not a direct part of Paderborn, but located in Büren, Schlangen, Elsen or Schloß Neuhaus, there are possibilities to split the network in a way, that a new community can be founded by smaller cities or villages.

The first changes take place at the gateway infrastructure. Right now, all gateways are connected by using fastd connections between them. Those fastd connections are used to transfer the Batman-adv traffic, and they get removed, as a certain gateway is just responsible for a subnetwork in the new architecture. However, all gateways are still connected to each other, as all of them are still a part of the Freifunk Paderborn community. Therefore, some services might be located in a different part of the network, which should still be reachable by all nodes. Services like `http://highscore.ffpb` or `http://start.ffpb/` are identified by using their URL, so this traffic can be easily routed by using the gateway nodes and there is no need to apply further changes.

A cluster cannot be generated automatically, as the calculation of the geographical layout is not possible due to missing data. Even if this data would be available, the implementation wouldn't be possible at reasonable cost. So the clusters were created manually by using my personal experience and knowledge about the geographic layout of the network. Furthermore, the plans for new network installations are also important, as they might change the current situation and violate the conditions mentioned above. In the following, I will define the different clusters and give reasons for these decisions.

Paderborn / Elsen / Schloß Neuhaus (Paderborn): The inner city of Paderborn covers most of the clients and nodes. Here, roaming of client devices is possible, and due to this, there are no possibilities to change this part of the network, as important functionality would be lost in this case. The geographic positions of Elsen and Schloß Neuhaus are very close to Paderborn, and there are no big gaps between those three. Today, there are no direct connections between the different parts, but for future reasons, they are all sticked together. This cluster is called *Paderborn* for the remainder of this document.

Büren: Büren is located south of Paderborn. The number of nodes in Büren rises strongly and there are also first plans to install local gateways, which are connected to the network using directed wifi links. This could be the first step to create a new community that is independent from Paderborn. As the beeline between Paderborn and Büren is more than 20 km, and Büren is, geographically speaking, below Paderborn, a wifi link would be very expensive and complex and so, Büren will be another cluster.

Bad Lippspringe / Schlangen/ Hövelhof (Schlangen): The network in Bad Lippspringe and Schlangen is promoted by local companies, as they would like to

offer access to the Freifunk network for their customers and tourists in those cities. There are a some plans for bigger installations in both cities and a high amount of new nodes is expected. The amount of nodes in Hövelhof is not that high right now, and there are no bigger installations planned. However, the beeline between Bad Lippspringe and Hövelhof is about 12 km, but the area between both cities is covered by a military area or a preserved area in the future. Due to this, there won't be any new nodes and so Hövelhof is included in this cluster. This cluster is called *Schlangen* for the remainder of this document.

Remainder: Nodes are distributed within a big area and there are places with a few nodes, but not enough to put them in a new cluster. So there is a fourth cluster, which contains all other nodes that don't fit to a cluster mentioned above.

All clusters contain both wifi-only and VPN nodes. Figure 4.2 shows the distribution for both kinds. The bars for the group *Model* give a reference for both types of nodes in the traditional implementation.

To support clusters in the network, changes to the omnetpp.ini and to the network file are mandatory, as connections to gateways were changed and every node needs another mesh ID (see Chapter 2) in order to distinguish between packets of its own mesh network and the others. Again, the generation of the .ini, .ned and .xml file takes place by using the Java software. It requires some startup parameters to

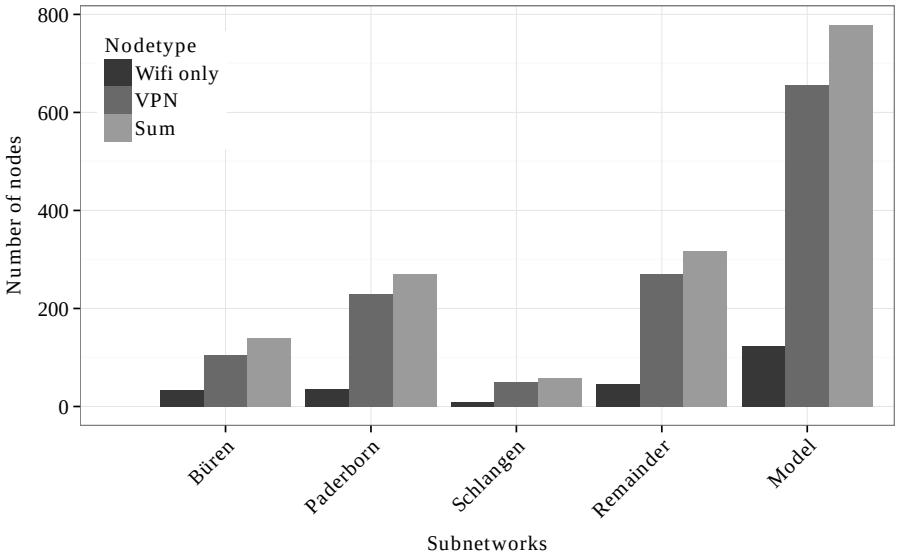


Figure 4.2 – Wifi-only and VPN nodes

generate the files according to both scenarios, namely a network with clusters and one network without clusters.

The area for one cluster gets specified by the longitude and latitude. All nodes which fits to such an area are mapped to the specific clusters. All nodes that not fit to a cluster are collected in the cluster "remainder". Of course, all nodes without GPS coordinates, but acting as a neighbor of some node with a well known location, are also considered for the according cluster.

The changes in the Batman-adv code are negligible, as just the used gate between all gateways has been changed in the simulation's network file.

The preparations and metrics are the same as described in Section 3.3.

4.1.1 Routing overhead

Figure 4.3 shows the send count metric in the form of an ECDF plot. The data from the model validation is added again to have a reference to the traditional model. As expected, the number of packets is lower than in the original model. However, the course of the cluster graphs are slightly different than the one of the original model.

Upon closer look, the plots of the clusters *Paderborn* and *Remainder* shows a temporary flattening of the curve. A more detailed analysis showed that this behavior is connected to the number of nodes which are connected to a certain node by using a wireless link. This is because of the OGM forwarding mechanism of Batman-adv, which is explained in Figure 4.4. Node A in Figure 4.4 has two neighbors B

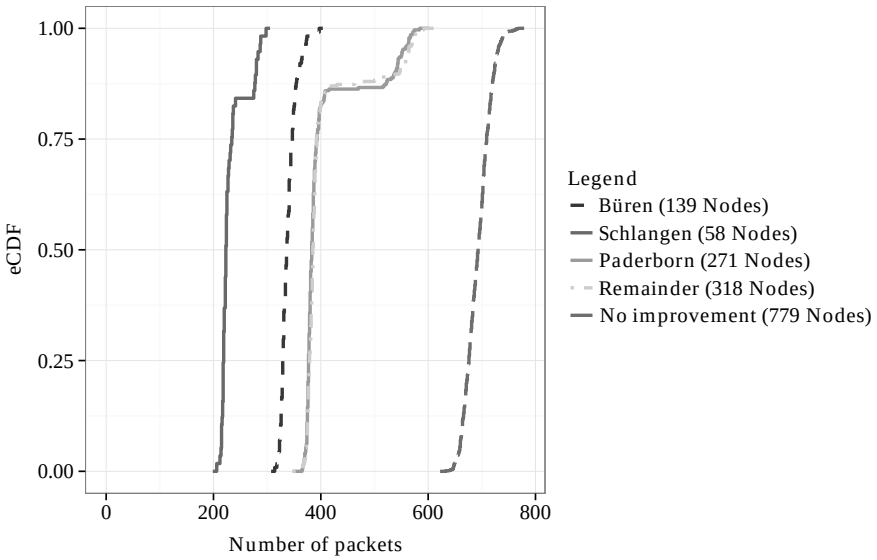


Figure 4.3 – ECDF for sent OGMs in the cluster scenario and in the current implementation.

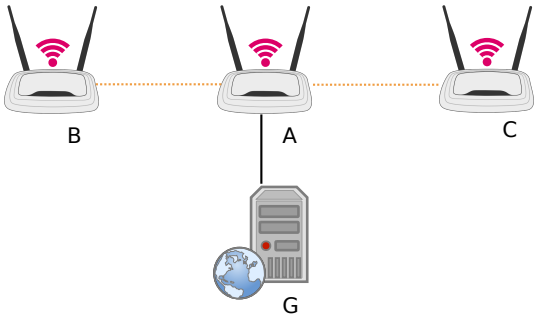


Figure 4.4 – The number of sent packets is higher for a node with certain wifi neighbors, as Batman-adv always forwards OGMs from one-hop neighbors.

and C which are connected by using a wireless link. Due to this, they become a direct neighbor of A. Node A receives OGMs from both B and C. Those OGMs get processed by A and then forwarded as Batman-adv always forwards OGMs from direct neighbors. Furthermore, node A has to rebroadcast all forwarded OGMs from B and C, except those OGM, that get dropped during the checks. This effect is not directly visible within the data from the simulation of the model validation. The number of nodes is much higher, and the effect gets blurred.

The data for the received packet count is shown in Figure 4.5. The assumption was a behavior similar to the course of the sent packet metric. Having a look at the data, a different outcome can be noticed. Both, the cluster for Paderborn and for the Remainder show a received packet count which is similarly high as the one from the model validation, although those clusters have a much lower number of nodes. In particular, Paderborn covers 271 nodes and the Remainder covers 318, but the simulation with 779 nodes leads to a similar amount of packets. A more thorough evaluation showed, that the effect is related to the gateway infrastructure. A further analysis has shown, that a different distribution of nodes across all gateways can lead to a lower packet count for the received metric. This improvement is discussed in Section 4.2. The aggregation metric is not taken into account here, as it is not meaningful. The aggregation is related to the number of OGMs which were sent and received. As the variety, in relation to the number of nodes, is high, the aggregation count differs a lot and thus is not helpful in order to estimate this part of the network change.

4.1.2 Delay: Node announcement

Figure 4.6 shows the delay for the node announcement in the cluster scenario as an ECDF. As four clusters are used, the plot contains one graph for each and one additional graph for the old implementation explained in Section 3.3. The behavior is the same as in the old implementation, but with one new node for each cluster. The

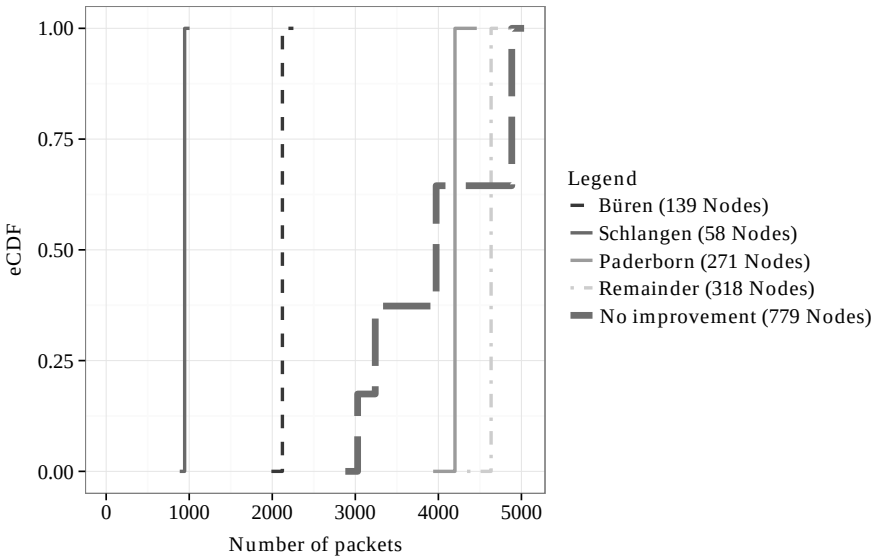


Figure 4.5 – ECDF for received OGMs in the cluster scenario and in the current implementation.

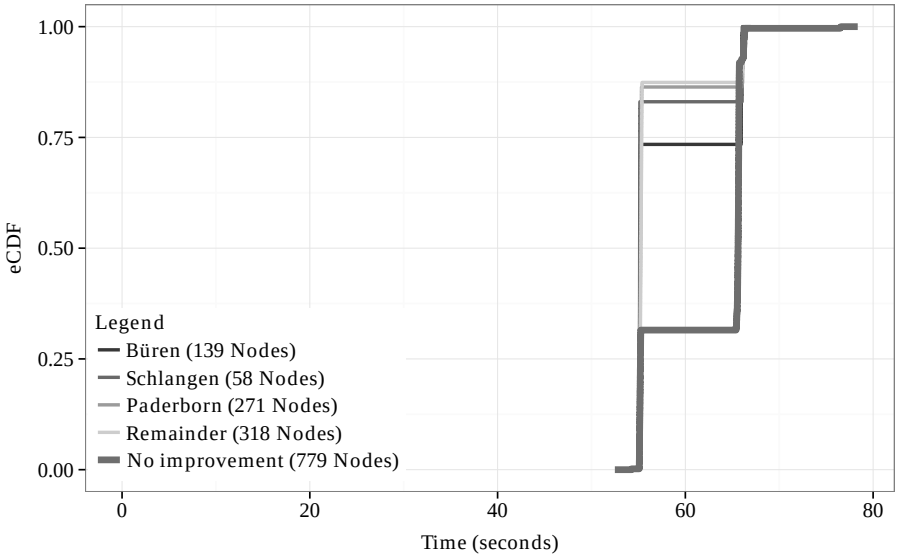


Figure 4.6 – Delay for the announcement of a new node in the cluster scenario as an empirical CDF.

only part that differs, are the connections between all gateways. As those connections are not used any more to transmit OGMs, a node is only announced in its cluster network. Due to the rebroadcast mechanism of Batman-adv, a connected gateway rebroadcasts the OGM directly to all neighbors. All nodes in the cluster, which are using a VPN connection to get the knowledge about the node after a gateway was reached. Due to this, there is a first high increase, as a lot of nodes are connected by a VPN connection. In the original scenario, all further hops take more time as the OGM gets directly dropped at the second hop because of the TQ, which is zero. So it takes at most about 10 seconds to reach the next hop. As evident in the old implementation, the second increase in the graph happens at a later time, due to different delays.

On the whole, the overall delay to announce a new node decreases just a little bit. This is due to the simplified architecture with less hops than in the original one. Furthermore, all cluster graphs have a higher increase in the first movement of the graphs. This is because the old implementation does not have that many direct neighbors per gateway, relative to a gateway in the cluster scenario.

To summarize, the improvement for the node delay metric is negligible.

4.1.3 Delay: client announcement

Figure 4.7 shows the delay for the announcement of a new non-mesh client. As in the original simulation, a new client joins the network at simulation time T=55 s.

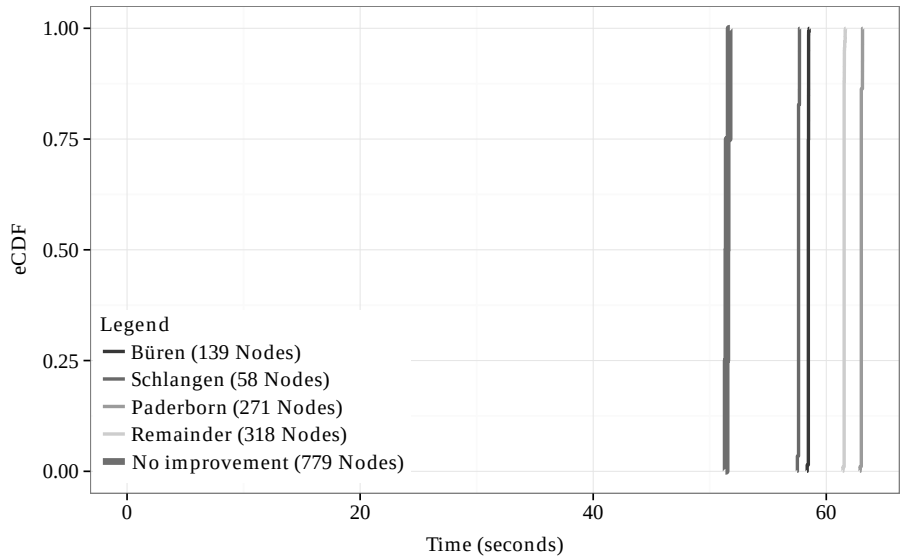


Figure 4.7 – Delay for the announcement of a new non-mesh client in the cluster scenario as an ECDF.

Opposed to the model validation, there is one client for each cluster in the network. The reason for this is the same as for the node announcement, since a new client is just announced within its cluster.

As a new client gets announced within the next scheduled OGM of the connected node, it takes some time until the client gets associated with the node. After that, it takes up to five seconds, until the next OGM with an incremented TTVN and the attached client data is scheduled. Due to the fact, that a fully established network with all routing information is given, the announcement of the node is as quick as in the old implementation. However, based on Figure 4.7, it seems as the client announcement delay got worse than before. This is based on the delay a node uses before it's first OGM is sent. Therefore, these variations are within the theoretical tolerances. Based on the used delays when forwarding an OGM, the OGM gets aggregated and forwarded in a short-term manner, and the client gets announced immediately.

However, the overall delay to announce a new client is negligible. Again, to improve the complete time until the new client gets announced, the implementation could be changed, so that the TVLV gets only attached right before sending the OGM. Thus, it doesn't contain information that is already known for some time. Of course this improvement would affect the original model and the optimization, but the effect would be the same in both scenarios.

4.1.4 Open issues

In order to realize different mesh networks and individual gateway infrastructures per cluster, the firmware of all nodes has to be changed. This is for several reasons. First of all, the mesh ID of each node changes. Furthermore, all gateways are registered in a node's firmware. As gateway nodes are dedicated to a certain cluster right now, this gateway list now depends on the cluster a node belongs to. However, the automatic deployment of a new firmware is already supported at the moment, and a first suggestion for clustering is already given with this thesis.

The first issue is the handling of nodes, that are connected to the network by using a wireless link and no VPN connection. Figure 4.8 shows a scenario where node A is connected to a gateway and there are two further nodes which do not have a direct connection to gateway G. To update all nodes, it is required to start with node C, as the mesh ID changes with the update. So if node B gets updated before C, C wouldn't be able to communicate with B any more and C would need a manual update. So the update procedure would be C, B and then A.

If the clustering is implemented and the new firmware gets installed by using the automatic update mechanism, each node is mapped to exactly one cluster. With the first firmware upgrade, this can be done automatically. If a new node gets installed,

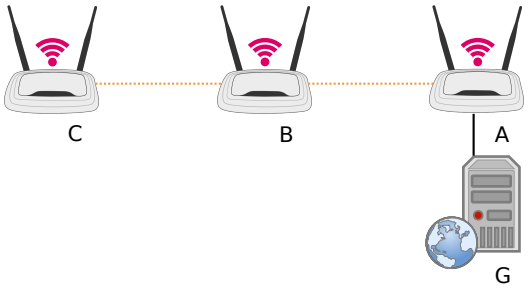


Figure 4.8 – Firmware update scenario. If the mesh ID changes and node B receives the update before node C, node C needs a manual update.

the affiliation of a certain cluster must be configured somehow, and right now, the only way to do this is with a manual configuration during the installation of the node. However, a significant amount of node operators are not able to reconfigure nodes on their own. So if some operator moves a configured node to another place without changing the network affiliation, the node might not be able to participate in the mesh by using a wifi link, and the operator is not able to locate and fix this issue. Thus, some kind of emergency procedure might be needed in order to automatically repair the router's configuration, or some different mechanism that allows a node to exchange data with other nodes, if the configuration is wrong.

4.2 Improvement: Node distribution across available gateways

Section 4.1 discusses an improvement that splits the network into multiple subnetworks. This approach shows a behavior, which is unexpected. As shown in Figure 4.5, the clusters for Paderborn and for the Remainder show a received packet count that is similar to the old implementation without any improvement, although those clusters have a much lower number of nodes.

As explained in Section 4.1, each node uses two gateways, like in the original model. Again, the effect can be explained with the forwarding mechanism of Batman-adv. As each node is connected to the gateways, they are considered as one hop neighbors and thus an OGM gets aggregated and forwarded. Considering the node distribution from Figure 3.7 in the model scenario, gateway one serves about 36 % of all nodes in the network which is about 280 nodes. The cluster for Paderborn covers 271 nodes and the data is placed around the center of the model's data in the plot, which matches this explanation. To prove this assumption, I performed the simulation without the clusters, but with an even distribution of nodes across all gateways. The result is shown in Figure 4.9.

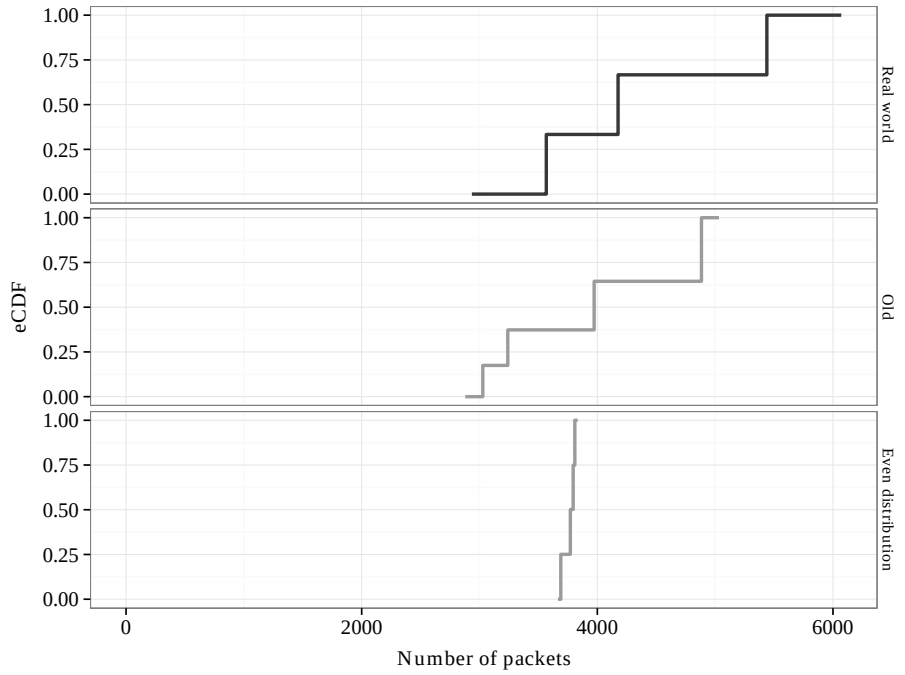


Figure 4.9 – Received OGMs in all different scenarios. The first plot shows the measurements from the network in the real world. The second plot shows the simulation results when using the same amount of nodes per gateway like in the real world. The third plot uses four different gateways, where each gateway serves about 25 % of all nodes in the network.

I performed the experiment with four gateways, where each gateway serves about 25 % of all nodes. The results is a curve which shows a few steps according to the forwarding behavior of Batman-adv. Based on the results, all nodes have approximately the same receive packet count, when using the even distribution of nodes across all gateways. Therefore, changing the node distribution among all gateway nodes is another possibility to influence the overall OGM count of nodes in the network. In the simulation results of Section 4.1, the OGM count can even get worse, if a lot of nodes are connected to one certain gateway in the network. This happens with the cluster of Paderborn and the Remainder. Therefore, it is advisable to distribute all nodes as evenly as possible over all available gateway nodes.

As explained, the amount of nodes for a certain gateway is predefined by an individual configuration for each gateway. This is required, as the fastd VPN connections require computational power. As gateways with varying hardware equipment are used, the overall number of possible fastd connections per gateway is limited by the hardware. This is why the suggested improvement cannot be implemented at the moment. To make an implementation possible, more powerful hardware is needed, resulting in higher operating costs.

4.3 Improvement: Termination at gateway nodes

The complete Freifunk network should be based on a number of nodes which are of the same computational power. This means that a certain node in the network doesn't have more control of the network than any other node. However, this paradigm is already violated, as the network needs at least one gateway node to keep the network fully connected. Furthermore, a gateway offers different services like DHCP, DNS and access to the Internet. As there is no mechanism to ensure authenticity or integrity of the provided information from other mesh participants, a gateway could easily forge its information. For example, the TQ values of OGMs could be changed in order to change the route selection of nodes. This could be easily used to focus all traffic on a certain node in order to perform a denial of service attack. In order to reduce the amount of OGMs even more, the role of all gateway nodes could be made even more powerful.

In the current scenario, gateways are used to keep the network fully connected and to offer services like access to the Internet or special Freifunk services. A gateway handles most of the traffic, because it acts like a hub in the network, forwarding OGMs to all VPN nodes. Without gateways, a node would just have knowledge about its local environment, which is reachable with the wifi signal. The idea is to change the complete fundamental principles of a decentralized mesh network and to outsource the complete routing of all traffic to the gateway nodes. With this, there is no need any more to forward OGMs to other VPN nodes. A node has just knowledge about its local environment that is reachable within the node's radio signal. However, it is still necessary to ensure that a gateway knows all nodes and all routes in the network. For this reason, it must be ensured that the OGMs from nodes without a VPN connection get successfully forwarded to a gateway. Furthermore, a node needs to know all gateway nodes in the network in order to perform the gateway selection of Batman-adv.

In the Freifunk scenario, there are three different cases where an OGM gets forwarded. All different scenarios are illustrated in Figure 4.10.

The first scenario is the communication between a node and a gateway by using a VPN connection. This is the most interesting part, as the gateway needs to know about the node and its neighbors, but the node itself just needs to know about all gateway nodes. Therefore, the behavior of the node remains the same like in the traditional implementation. The functionality of all gateways is changed. Here, the VPN connections are just used to transmit OGMs, which contain gateway information. This can easily be checked, as gateway information are attached to each OGM. With this mechanism, a node just receives OGMs of every gateway in the network, but not of other nodes. This ensures that a node has full knowledge about the gateway

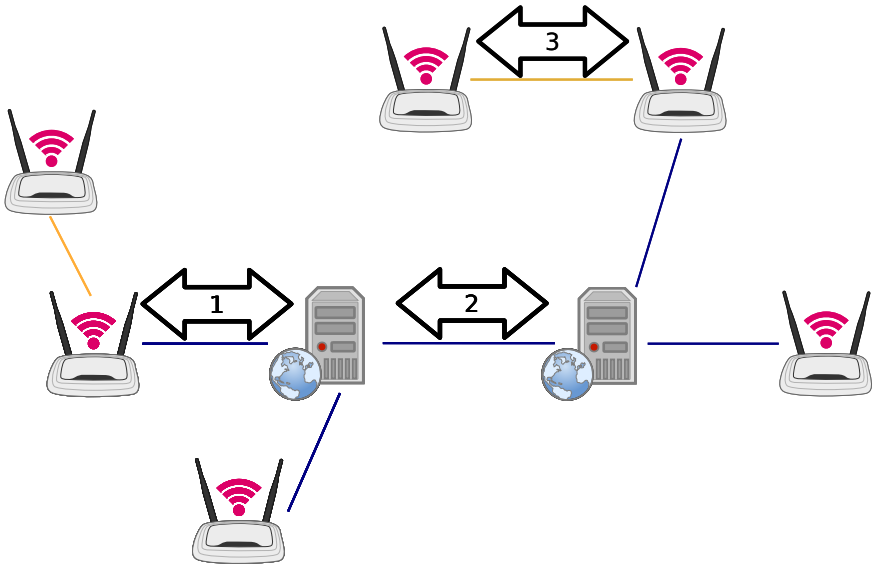


Figure 4.10 – Simplified network architecture of a Freifunk network. Different communication scenarios are marked with arrows.

structure, so it can change to another gateway, by using the configured gateway selection criteria.

The second scenario is the communication between all gateways. This part remains the same, as every gateway needs full knowledge about the network. Thus, all OGMs are exchanged at this point.

The third scenario is the wifi mesh connection between nodes. As there are nodes in the network without a VPN connection, the OGMs from those nodes need to be forwarded to the gateways. Therefore, all OGMs received via the wireless interface are also processed and forwarded like in the traditional implementation.

Furthermore, the OGMs from gateways need a rebroadcast in all scenarios. This is necessary, as the TQ metric depends on rebroadcasts to result in a value greater 0 for neighbors. A node is not able to make individual decisions any more. To ensure valid routing, a node just sends everything, that can not be handled with the local knowledge, to the preferred gateway.

Figure 4.11 shows the send count metric in the form of an ECDF plot. As one might logically have expected, the overall number of OGMs gets down to a much lower count. Again, the differences are caused by a different number of wifi mesh neighbors. As those neighbors are direct neighbors of the mesh VPN nodes, all OGMs get forwarded.

The analysis of the different nodes reveals, that a node just knows about its local environment and about all gateways. Thus, a node is still able to choose a preferred gateway, based on the selected metric. Of course, a gateway's OGM gets forwarded

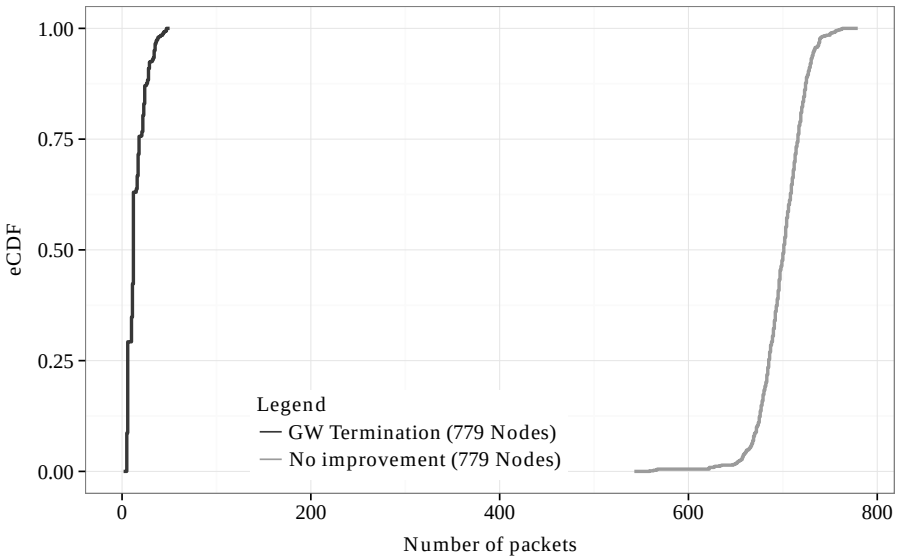


Figure 4.11 – ECDF for sent OGMs with the termination of OGMs at a gateway node.

to all nodes that are just using the wifi mesh. Furthermore, all gateways are able to create an originator table, covering all available nodes in the network. Thus, routing is still possible at this point.

A graphical representation for the received OGM count is skipped at this point. As a gateway just forwards OGMs of other gateways, the same amount OGMs is received on the VPN interface by every node. The overall amount is smaller than 100 and very low compared to the data shown in Figure 3.9.

The aggregation mechanism is no longer involved, as the amount and the frequency of OGMs is not high enough. Thus, the analysis of the aggregation metric has shown that almost all OGMs are sent as a single packet.

The delay measurements are also the same as in the model validation. Of course, the announcement is terminated, if a gateway node is reached. Thus, the delay for the node announcement gets lower.

This approach violates the fundamental idea of a decentralized network. The current approach of all Freifunk communities already tries to avoid the centralized gateway infrastructure. However, those gateway nodes are still needed at the moment, as there is no other way to solve the problems of various services, and to keep the complete network connected. Regardless of legal restrictions for the Internet access, a gateway is always needed in order to provide services like DHCP and so on. However, in order to avoid hosting solutions in big data centers, the current efforts focus on a wireless backbone infrastructure in Freifunk cities. This wireless backbone infrastructure allows to place gateways locally in the city of a community.

If a big subset of nodes is connected to such a gateway by using the wireless links, the fastd connections using the owners Internet connection becomes redundant, as the wifi backbone would be sufficient. In order to reduce the power of gateway nodes even more, services like DHCP could be outsourced to multiple smaller and cheaper devices like EdgeRouter[29] devices. Tests in the real world have already shown that this approach is usable. By eliminating the German Störerhaftung problem, access to the Internet can directly be provided by the nodes without using expensive gateway nodes.

The current plans for backbone links in Paderborn are not complete at the moment, but there are already a few links established and planned. Those links are used to connect local gateways, which are directly hosted in Paderborn, to different locations. It is utopian to reach all nodes in a city by using local gateways and a local wireless backbone infrastructure. Figure 4.12 shows a small example for the wireless backbone in Berlin. As shown in Figure 4.12, there are a lot of wifi links in

[29] `https://www.ubnt.com/edgemax/edgerouter/`

[30] `http://openwifimap.net/map.html#bbox=52.45600939264076,13.263416290283203,52.55871643879347,13.593006134033203`

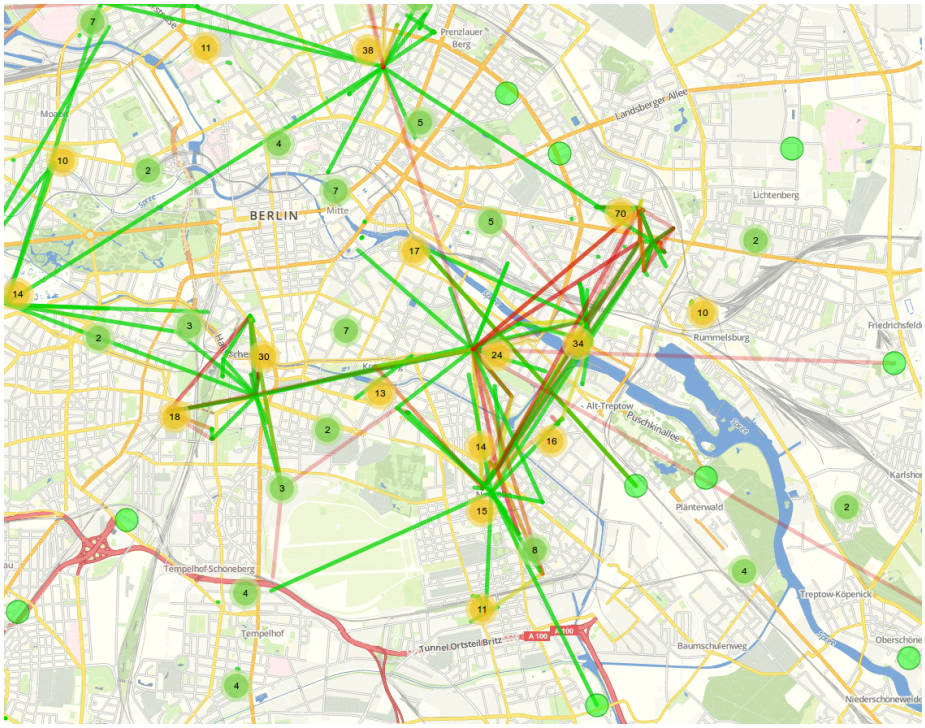


Figure 4.12 – An example of the wireless backbone network in Berlin[30]. Every dot is another location. The number inside a dot indicates the number of nodes, which are connected to this wireless link. The lines represent a wireless link. ©OpenStreetMap contributors

the city, but there are still a lot of locations, that are not connected to the backbone infrastructure. Therefore, it is utopian to connect the complete network by using the backbone infrastructure and central gateways in data centers with a broadband access are still required. The complete central gateway infrastructure cannot be fully replaced, but the amount of data that needs to be transmitted by using the Internet connection of residential ISP customers can be reduced.

4.4 Improvement: One fastd connection

Right now, a node uses two fastd connections to two different gateways. Those connections are chosen by the Batman-adv gateway selection and by the Gluon firmware. In fact, a random mechanism of Gluon chooses both gateway nodes that should be used afterwards for the fastd connections. As a node needs some connection to the gateway infrastructure in order to gather knowledge about all gateways, the Batman-adv gateway selection is performed afterwards. However, the fastd connection is independent from the Batman-adv gateway selection. So there could be a different gateway selected by Batman-adv than by the random mechanism of Gluon.

A node establishes two different connections in order to react quickly in case of a failure. This might be the case, if the primary gateway node needs to reboot or simply becomes unavailable due to another reason. If a fastd connection becomes unusable, the node can switch to the second one to send and receive packets. The gateway selection of Batman-adv is not affected by this.

In the current implementation, a node receives almost every packet twice. Due to the broadcast of OGMs, a packet is received from the first gateway and shortly after from the second one. Instead of receiving a packet twice, OGMs from the node are just transmitted by using one fastd connection, namely the connection to the primary fastd connection. This is an implementation detail of Gluon.

The idea is to remove one of the fastd connections in order to reduce the amount of OGMs that are received. The changes are only related to the generation process of the network file, used by OMNeT++. Here, the number of gateway connections is simply reduced to one instead of two. Changes to the Batman-adv algorithm are not necessary.

Applying this change to the real network is also possible without investing a lot of work and putting the complete network at risk. Gluon provides a parameter that allows to change the amount of gateways, a node uses for the fastd connections. Changing the amount of gateways doesn't have a direct influence on the network performance. The throughput and other metrics are also unaffected, as only one connection is used to send and receive at a time. The second fastd connection is

just used, if broadcast packets are received. Therefore, changing this parameter can quickly be implemented and reverted in case of problems. Because of the argumentation mentioned above, this change was also performed in the real world, to compare the simulation results with measurements from the real network. The firmware has been adjusted by the firmware developer of the Freifunk community in Paderborn. All measurements have been performed after the firmware was available on all nodes with enabled auto update feature.

Figure 4.13 shows the sent packet count metric as an ECDF. As mentioned above, the plot shows the simulation result of the improvement and the measurement in the real world when using one fastd connection instead of two. To compare the results of the improvement, the data from the old implementation of the model is added as well. The number of sent packets is similar to the measured value in the model validation, as it was bound to be. This is simply because only one fastd connection is used to send packets, independently from the overall number of established connections. The overall number of nodes in the network has changed. Thus, the packet count with 820 nodes is higher than with 779 nodes.

Figure 4.14 shows the amount of received packets as an ECDF. The data shows a good matching between the simulation result and the data from the real world. As mentioned in the model validation, a rather small sample from the real world is used in order to compare both result sets. Therefore, the results are compared by using the graphical approach. However, the simulation data shows four different steps whereas the data from the real world shows only three. Of course, the steps in the

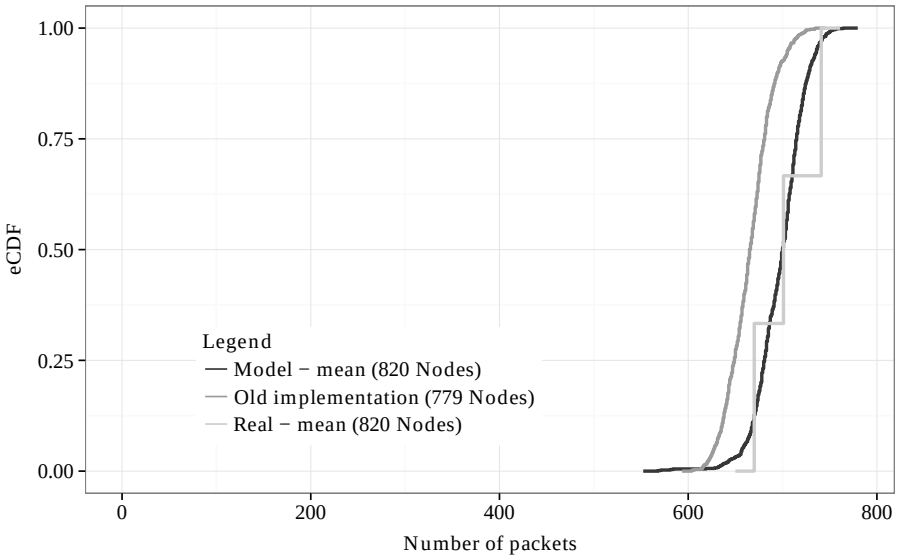


Figure 4.13 – Sent OGMs by using one fastd connection instead of two as an empirical CDF.

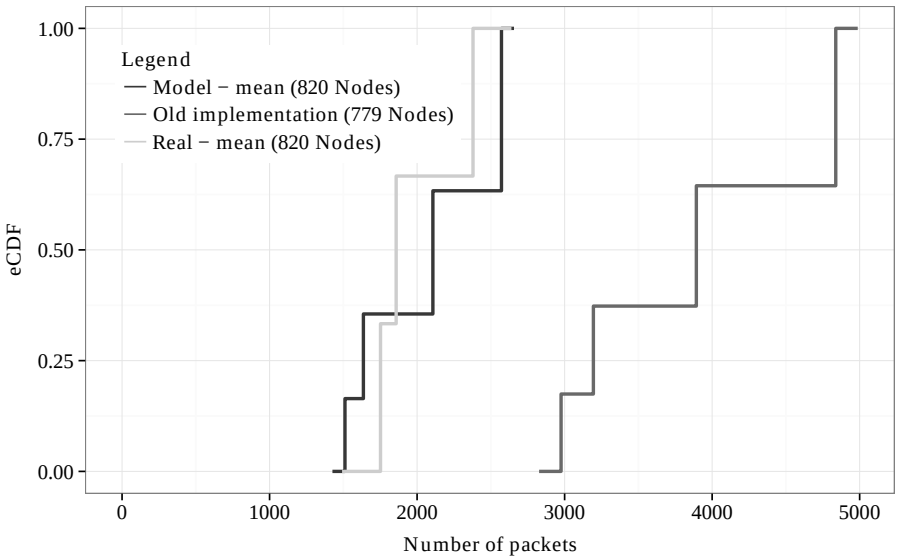


Figure 4.14 – Received OGMs by using one fastd connection instead of two as an empirical CDF.

real world measurement are due to the fact that only three nodes were used for the measurements. The four steps in the simulation are due to four different gateway that are used. It should be noted that the amount of packets has nearly halved. That was to be expected. First of all, one connection is skipped and furthermore, the number of duplicated packets is reduced.

Plots for the delay measurements are skipped at this point, as the behavior is the same like in Section 3.3.3 and 3.3.4. This was to be expected as well. Due to the sequence number checks, each node drops the OGM from the second fastd connection immediately after the preliminary router checks. Thus, even in the previous version, only the first OGM was used in order to announce a node or a client, as the second one was also dropped in that scenario.

Based on the graphical validation, the changed simulation reflects the behavior of the implementation in the real world. By using an average size of 100 Byte for an OGM, this works out to ∼10 GB of data for each node per month for the transmission of OGMs. This is a reduction of ∼46 % compared to the old implementation.

Two fastd connections were used in the old implementation to ensure reliability, if gateway needs to reboot and the connection gets lost. However, the uptimes of the gateway nodes in Paderborn vary between 100 and 1 days. The major number of gateways has an average uptime which is lower than ten days, and therefore it is not uncommon that a gateway needs to reboot. This is because sometimes Batman-adv causes a kernel panic. Therefore, a reboot is required to temporarily solve this issue. The reason for this is not known as of today. However, a survey in the real network

with two fastd connections revealed, that the delay in case of a gateway failure is between 120 seconds and 150 seconds. After the number of fastd connections has been set to one, the delay was between 180 seconds and 200 seconds. Therefore, using one fastd connections leads to an increase of about one minute, which is ∼33-50 % compared to the old implementation. However, even if a gateway reboots once a day, an additional failure time of one minute is acceptable.

4.5 Improvement: Network splitting and one fastd connection

Both improvements in Section 4.1 and 4.4 lead to a reduction of OGMs without an impact to other metrics. Therefore, the clustering approach can be used with only one fastd connection in order to use the advantage of both improvements. In addition to the reduction of OGMs, a smaller network might also lead to a lower overhead in terms of IPv6 and ARP. Therefore, this approach first of all improves the OGM overhead and second, all advantages mentioned in Section 4.1 are achieved.

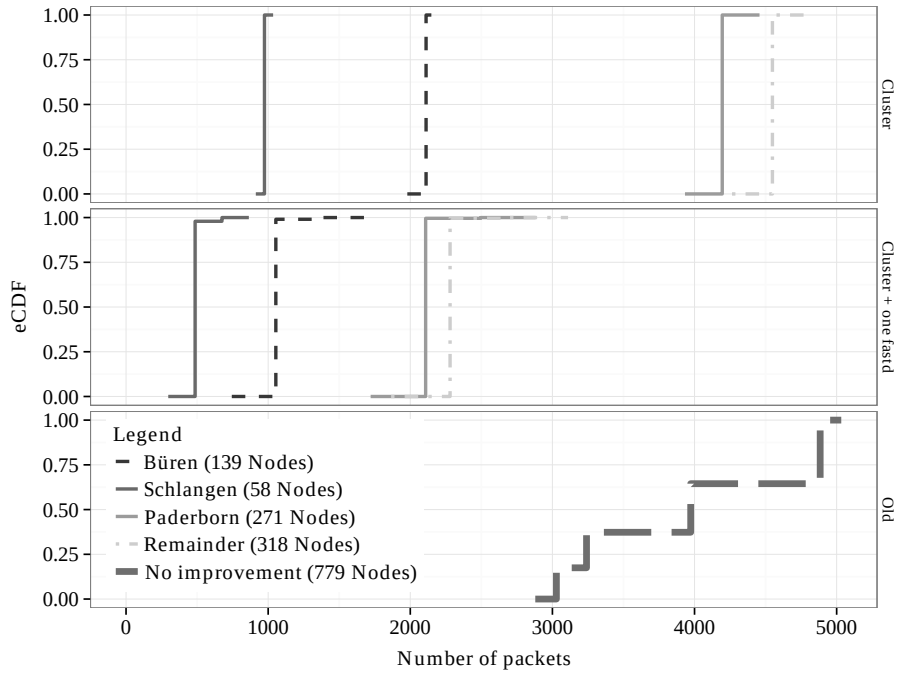


Figure 4.15 – Received OGMs in all different scenarios. The first plot shows the simulation results, if only the clustering is used. The second plot shows the results for the combination of clustering and one fastd connection. The third plot shows the old implementation without any improvement as a reference.

Figure 4.15 shows the received packet count metric as an ECDF. The figure compares three different scenarios. The first graph shows the simulation results of the cluster scenario and the second one shows the cluster scenario, when only one fastd connection is used. The third one shows the received packet metric when no improvement is used.

The behavior is similar to the results if only one fastd connection is used. The overall amount of received packets is nearly halved for each cluster, as was to be expected. In addition to this, the clustering leads to a reduction of the overall OGM count.

The plot for the sent packet count metric is omitted here. As already shown in Section 4.4, one fastd connection only has a small impact on this metric. Therefore, the sent packet count is only slightly changed in this scenario.

Both delays for nodes and clients are the same as in Section 4.1, as one fastd connection doesn't have an impact on those metrics. This was also shown in Section 4.4.

The analysis shows, that a combination of both improvements can lead to a received OGM reduction of ∼54 % to ∼90 % compared to the model without any improvement. The differences are due to the different sizes of the used clusters.

To use this approach in the real network of Paderborn, the same issues as in Section 4.1.4 need to be solved. The reduction to one fastd connection is already implemented.

Chapter 5

Conclusion

This master thesis gives an overview of the technology of mesh networks and the idea of the Freifunk project. It describes the fundamentals of the Batman routing algorithm that is used in the Freifunk network of Paderborn, and explains the problem, which is related to the high number of Originator Messages (OGMs) that are used to establish the network.

Further, important aspects of Batman version IV are presented and explained with various examples. This version has been distinguished from other routing protocols, previous versions and future developments of Batman itself.

I discuss the approach, that is currently implemented in the real world and the model process for the simulation by using a lot of data from the real world. This data has been gathered and aggregated by using an additional software. I developed this software to automatically create all configuration and network files for the OMNeT++ simulation. The model has been compared to the real Freifunk network by comparing the simulation results with a sample from the network. The analysis was performed by using different metrics, that were chosen to be meaningful. Due to the small sample from the real world, the validation is based on theoretical assumptions and on a graphical analysis of the sample and the simulation results.

The developed model was used to improve the overall number of transmitted OGMs in the network. This thesis contains five different approaches. The first one splits the network into multiple subnetworks whereas the second one uses an equal distribution of all nodes across all gateways. The third approach changes the node hierarchy in order to avoid forwarding OGMs to regular nodes. The fourth approach sets the overall number of fastd connections per node to one instead of two. This approach has been implemented in the real network, and the simulation results have been compared with new measurements from the real network using this improvement. The last approach uses one fastd connection and combines this with the subnetwork approach. It has been shown, that all approaches lead to a lower

amount of OGMs to establish the network, without impairing other metrics like the delays. However, all approaches require different changes in the real implementation, which are also discussed with each improvement.

Based on the simulation results, I suggest a combination of two approaches, namely to split the network into multiple subnetworks and to reduce the number of fastd connections to one. Setting the number of fastd connections to one doesn't have any negative influence on the networks stability and it improves the OGM overhead by over 46 %. A combination with the network splitting approach leads to a reduction of received OGM that is up to 90 %. Further to the problem of the amount of OGMs, there is also an overhead due to the use of IPv6 and ARP. Both additional overheads are also related to the network's size, and it can be reasonably assumed that this overhead also gets lower, as the network becomes smaller. Furthermore, this approach provides the possibility to create new Freifunk communities.

In a future work, it would be useful to extend the simulation to analyze the behavior of ARP together with Batman-adv. As mentioned above, the ARP mechanism probably leads to a high overhead as well. In order to investigate this behavior, the simulation can be used as well. With this thesis, the complete Freifunk network of Paderborn is available within the OMNeT++ simulation and the model validation has shown that the network structure is correct. Furthermore, there is the Java software which allows the download of all required data and parameters to generate the according network and client description for the simulation. Therefore, the Batman algorithm can easily be replaced by some other algorithm. With this, different algorithms can be compared in the specific Freifunk scenario of Paderborn in order to further improve the performance and stability of the complete network.

List of Abbreviations

batmand	B.A.T.M.A.N. daemon
BSSID	Basic Service Set ID
CRC	Cyclic Redundancy Check
DHCP	Dynamic Host Configuration Protocol
DSDV	Destination-Sequenced Distance Vector routing
ECDF	Empirical distribution function
ELP	Echo Location Protocol
EQ	Echo Link Quality
ETX	Expected Transmission Count
fastd	Fast and Secure Tunneling Daemon
GOF	Goodness Of Fit
GSoC	Google Summer of Code
HNA	Host Network Announcement
ICMP	Internet Control Message Protocol
ISP	Internet Service Provider
KS-test	Kolmogorov-Smirnov test
LQ	Link Quality
MPR	multipoint relay
MTU	Maximum transmission unit
NHDP	Neighborhood Discovery Protocol
NIC	Network Interface Card
OGM	Originator Message
OLSR	Optimized Link State Routing
Opkg	Open Package Manager
RQ	Received Link Quality
SQ	sequence number
SSID	Service Set ID
TC	Topology Control
TLV	Type-Length-Value
TQ	Transmit Link Quality

TT	Translation Table
TTL	Time to live
TTVN	Translation Table Version Number
TVLV	Type-Version-Length-Value

List of Figures

List of Tables

Bibliography

[1] M. Hundebøll and J. Ledet-Pedersen, "Inter-Flow Network Coding for Wireless Mesh Networks," Master's Thesis, Aalborg University, 2011.

[2] M. Eslami, O. Karimi, and T. Khodadadi, "A survey on wireless mesh networks: Architecture, specifications and challenges," in *IEEE 5th Control and System Graduate Research Colloquium (ICSGRC)*. Shah Alam, Malaysia: IEEE, August 2014, pp. 219–222.

[3] M. L. Sichitiu, "Wireless mesh networks: Opportunities and challenges," in *IEEE symposium on computers and communications*. Cartagena, Spain: IEEE, June 2005.

[4] H. Narra, Y. Cheng, E. K. Çetinkaya, J. P. Rohrer, and J. P. Sterbenz, "Destination-sequenced Distance Vector (DSDV) Routing Protocol Implementation in Ns-3," in *4th ACM/ICST International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2011): 3rd International Workshop on ns-3*. Brussels, Belgium: ICST, 2011, pp. 439 – 446.

[5] C. Sommer and F. Dressler, *Vehicular Networking*. Cambridge University Press, November 2014.

[6] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, "IEEE 802.11s: The WLAN Mesh Standard," *IEEE Wireless Communications*, vol. 17, no. 1, pp. 104 – 111, February 2010.

[7] E. Chissungo, E. Blake, and H. Le, "Investigation into Batmand-0.3.2 Protocol Performance in an Indoor Mesh Potato Testbed," in *26th IEEE International Conference on Advanced Information Networking and Applications (AINA 2012), Workshops*. Fukuoka, Japan: IEEE, March 2012, pp. 526–532.

[8] A. Hartmann, *Unterlassungsansprüche im Internet*. C. H. Beck Münchem, 2009, no. 75.

[9] A. G. Bowitz, "Simulation of a Secure Ad Hoc Network Routing Protocol," Master's Thesis, Norwegian University of Science and Technology, 2011.

[10] D. Furlan, “Improving B.A.T.M.A.N. routing stability and performance,” Master’s thesis, University of Trento, 2011.

[11] J. Klein, “Implementation of an ad-hoc routing module for an experimental network,” Master’s thesis, Universidat Politecnica de Catalunya, September 2005.

[12] K. H. Isaac, “Situation-aware routing for wireless mesh networks with mobile nodes,” Master’s Thesis, University of the Western Cape, March 2012.

[13] F. Oehlmann, “Simulation of the "Better Approach to Mobile Adhoc Networking" Protocol,” Bachelor’s Thesis, Technische Universität München, September 2011.

[14] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich, “Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.) - draft-wunderlich-openmesh-manet-routing-00,” IETF, Internet-Draft (Experimental), February 2008. [Online]. Available: https://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00

[15] Schiffer, Matthias and Schneider, Nils and Litza, Jan-Phillip, “Batman-adv-Legacy,” git checkout a85427747b93164876c8e9a8c6630b8848d1bfa2. [Online]. Available: https://github.com/freifunk-gluon/gluon.git

[16] L. Barolli, M. Ikeda, G. De Marco, A. Durresi, and F. Xhafa, “Performance Analysis of OLSR and BATMAN Protocols Considering Link Quality Parameter,” in *23rd IEEE International Conference on Advanced Information Networking and Applications (AINA 2009)*. Bradford, UK: IEEE, May 2009, pp. 307–314.

[17] E. Kulla, M. Hiyama, M. Ikeda, and L. Barolli, “Performance Comparison of OLSR and BATMAN Routing Protocols by a MANET Testbed in Stairs Environment,” *Computers & Mathematics with Applications*, vol. 63, no. 2, pp. 339 – 349, January 2012.

[18] H. B. Abolhasan, M. and J. C.-P. Wang, “Real-world performance of current proactive multi-hop mesh protocols,” in *Asia-Pacific Conference on Communications (APCC 2009)*. Shanghai, China: IEEE, October 2009, pp. 44–47.

[19] A. Neumann, C. E. Aichele, and M. Lindner. (2007, June) B.A.T.M.A.N. Status Report. [Online]. Available: http://downloads.open-mesh.org/batman/papers/batman-status.pdf

[20] T. Clausen and P. Jacquet, “Optimized Link State Routing Protocol (OLSR),” IETF, Internet-Draft (work in progress) 3626, October 2003.

[21] T. Clausen, C. Dearlove, P. Jacquet, and U. Herberg, "The Optimized Link State Routing Protocol version 2," IETF, Internet-Draft (work in progress) draft-ietf-manet-olsrv2-19, March 2013.

[22] S. Javed, F. ul Islam, and A. A. Pirzada, "Performance analysis of OLSR protocol in a Mobile Ad hoc wireless Network," in *2nd International Conference on Computer, Control and Communication (IC4 2009)*. Karachi, Pakistan: IEEE, February 2009.

[23] J. Wu, F. Dai, X. Lin, J. Cao, and W. Jia, "An extended fault-tolerant link-state routing protocol in the Internet," *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 1298 – 1311, October 2003.

[24] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morri, "A High-throughput Path Metric for Multi-hop Wireless Routing," in *9th ACM International Conference on Mobile Computing and Networking (MobiCom 2003)*. San Diego, CA: ACM, September 2003, pp. 134 – 146.

[25] C. E. Aichele. (2005, December) Link Quality Fish Eye Mechanism. [Online]. Available: http://wwwiuk.informatik.uni-rostock.de/fileadmin/wwwiuk/download/external/README-Link-Quality-Fish-Eye.txt

[26] M. Lindner, S. Wunderlich, S. Eckelmann, L. Lüssing, and A. Quartulli, "batman-adv - Git," git checkout e017251099df603ec0857dfe306bed2434d7b487. [Online]. Available: http://git.open-mesh.org/batman-adv.git/

[27] D. Johnson, N. Ntlatlapa, and C. Aichele, "Simple pragmatic approach to mesh routing using BATMAN," Pretoria, South Africa, October 2008.

[28] M. Schiffer, N. Schneider, and J.-P. Litza, "freifunk-gluon," git checkout 3f80b6585647dce960325c3df46b4407a00bca7f. [Online]. Available: https://github.com/freifunk-gluon/batman-adv-legacy.git

[29] Helge Jung, Tobias Hardes, Stefan Klöpping, "Freifunk Paderborn - PR," git checkout 17851f6c117f2e075be7d79022018e824ab2e00d. [Online]. Available: https://git.c3pb.de/freifunk-pb/pr.git

[30] M. Lindner. (2012, September) Gsoc 2012: Spyros gasteratos' final report. [Online]. Available: http://www.open-mesh.org/projects/open-mesh/wiki/2012-10-01-GSoC-2012-Spyros-Gasteros-Final-Report

[31] A. Quartulli and R. Lo Cigno, "Client announcement and Fast roaming in a Layer-2 mesh network," University of Trento, Tech. Rep. DISI-11-472, October 2011.

[32] A. Klein, L. Braun, and F. Oehlmann, "Performance study of the Better Approach to Mobile Adhoc Networking (B.A.T.M.A.N.) protocol in the context of asymmetric links," in *13th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2012)*. San Francisco, CA: IEEE, June 2012.

[33] D. Seither, A. König, and M. Hollick, "Routing performance of Wireless Mesh Networks: A practical evaluation of BATMAN advanced," in *36th IEEE Conference on Local Computer Networks (LCN 2011)*. Bonn, Germany: IEEE, October 2011, pp. 897 – 904.

[34] D. Kotz, C. Newport, and C. Elliott, "The mistaken axioms of wireless-network research," Dartmouth Computer Science, Tech. Rep. TR2003-467, July 2003. [Online]. Available: http://www.cs.dartmouth.edu/~dfk/papers/kotz-axioms-tr.pdf

[35] M. Z. n. Zamalloa and B. Krishnamachari, "An Analysis of Unreliability and Asymmetry in Low-power Wireless Links," *ACM Transactions on Sensor Networks*, vol. 3, no. 2, June 2007.

[36] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys '03. New York, NY, USA: ACM, 2003, pp. 14–27.

[37] S. Wunderlich, M. Lindner, and A. Lunn. batctl - B.A.T.M.A.N. advanced control and management tool. [Online]. Available: http://manpages.ubuntu.com/manpages/trusty/man8/batctl.8.html

[38] M. Lindner, "batman-adv - Head: marek/batman_v," git checkout 3dcc91d0e63114b1fb13bf03539156a9bf52a396. [Online]. Available: http://git.open-mesh.org/batman-adv.git/shortlog/refs/heads/marek/batman_v

[39] M. Lindner, S. Wunderlich, S. Eckelmann, L. Lüssing, and A. Quartulli. (2012) Echo location protocol (elp). open-mesh.org. http://www.open-mesh.org/projects/batman-adv/wiki/ELP.

[40] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," in *4th ACM International Conference on Mobile Computing and Networking (MobiCom 1998)*. Dallas, TX: ACM, October 1998.

[41] A. M. Law, *Simulation, Modeling and Analysis*, 4th ed. McGraw-Hill, 2007.