CHALMERS

Evaluation of open source software for mobile ad hoc routing in military tactical networks

Master's thesis within Computer Systems and Networks

Oscar Holmberg

Department of Computer Science and Engineering
Division Networks and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2013

Evaluation of open source software for mobile ad hoc routing in military tactical networks
Master's thesis within Computer Systems and Networks

OSCAR HOLMBERG

Examiner: PETER LUNDIN

Chalmers University of Technology
Department of Computer Science and Engineering
Division Networks and Systems
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden October 2013

Abstract

Evaluations of ad hoc routing protocols have been performed in several studies, both with hardware and in software simulators. However, the network topology plays an important role for the protocol performance. This master thesis evaluates OSPF, OLSR, Babel and Batman-adv in a tactical network set up. A tactical network is not a completely ad hoc scenario which makes it different from other studies. Furthermore, the protocols' settings are varied and they are mainly tested against convergence time and generated overhead traffic. The aim is to find which routing protocol is most suitable in the tactical network scenario that is tested. The tactical network set up, consisting of 36 nodes, is created with virtual machines and software that emulates realistically radio links.

The results show that OSPF can be configured to adapt to a tactical network. OSPF and Babel shows the best performance with respect to convergence time and generated overhead traffic. However, OLSR offers a lot of configuration possibilities which provides the potential to find a more suitable configuration than those that are used in the tests. Both Batman-adv and OLSR generates more overhead traffic than Babel and OSPF for the corresponding convergence time.

Keywords: Ad hoc routing, tactical network, OSPF, Babel, Batman-adv, OLSR.

Acknowledgements

I would like to thank my supervisor at Saab, Anders Gunnar, for the opportunity to perform this master thesis and for his valuable input during the project. I would also like to thank my supervisor at Chalmers, Daniel Cederman, for his proofreading and comments on the report.

Contents

Acknowledgements i
List of figures iv
List of tables v
Abbreviations vii

1 Introduction **1**
1.1 Background 1
1.2 Objective 2
1.3 Delimitations 2
1.4 Methodology 3
1.5 Related work 3
1.6 Report structure 4

2 Literature review **5**
2.1 Routing in MANET:s 5
2.2 Proactive or table-driven routing protocols 5
2.2.1 OSPF 6
2.2.2 DSDV 8
2.2.3 OLSR 10
2.2.4 Batman 12
2.2.5 Babel 13
2.3 Reactive or on-demand-driven routing protocols 13
2.3.1 AODV 14
2.3.2 DSR 14

3 Test environment **17**
3.1 Software 18
3.1.1 VirtualBox 18
3.1.2 Radio Link Emulator (RLE) 19
3.1.3 Quagga 20
3.1.4 Babeld 20
3.1.5 OLSRd 20
3.1.6 Batman-adv 21
3.2 Network set up 21
3.3 Test definition 27

4 Results and Discussion **30**
4.1 Results with default configurations 30
4.2 OSPF 34

4.3 OLSR 35
4.4 Babel 37
4.5 Batman-adv 39

5 Conclusion **42**

Appendix A - Network topology of test environment **46**

Appendix B - Overhead traffic for different configurations, relation between bytes and number of packets **47**

List of Figures

2.1 A mobile ad hoc network where node D moves to a new location. Changes in the routing tables must always be updated. 9
2.2 The messages sent during a neighbour discovery procedure in OLSR. 10
2.3 The difference of using MPR nodes in the flooding procedure. 11
2.4 Example how an error message is created in order to inform the source node that the destination can't be reached 15

3.1 Describes how the communication is allowed in a battalion. The communication is hierarchical and must follow chain of command. . . 17
3.2 Two virtual machines running on a host computer. The virtual machines are connected with an internal network inside VirtualBox. . . . 19
3.3 Overview of the routing nodes and how they are connected in a tactical network set up. 21
3.4 Overview of the routing nodes distributed among different radio networks. 22
3.5 Example of how the RLE and routing nodes are connected together. . 23
3.6 A layered view that shows a small part of the network and how it is running inside VirtualBox. 24
3.7 The three physical computers and how they are connected with the use of VLAN. 25
3.8 A logical view of how the traffic is separated in the three lab computers. 26
3.9 A logical view over the routing nodes where the edges are controlled by the RLEs. The dotted edges have a packet loss of 100 percent at some times during the test. Network traffic is logged at all gray nodes. 27

4.1 The ping ratio for all protocols with their default configurations. . . . 30
4.2 Overhead traffic on intermediate nodes when all links have packet loss from 0 to 25 percent. All protocols have their default configuration. . 32
4.3 Overhead traffic on end nodes when all links have packet loss from 0 to 25 percent. All protocols have their default configuration. 33
4.4 The relation between overhead traffic and convergence time for different settings for OSPF. 34
4.5 The ping ratio for different settings for OSPF. 35
4.6 The relation between overhead traffic and convergence time for different settings for OLSR. 36
4.7 The ping ratio for different settings for OLSR. 37
4.8 The relation between overhead traffic and convergence time for different settings for Babel. 38
4.9 The ping ratio for different settings for Babel. 39

4.10 The relation between overhead traffic and convergence time for different settings for Batman-adv 40
4.11 The ping ratio for different settings for Batman-adv. 41

List of Tables

2.1 Message types of the OSPF protocol 7
2.2 The routing table for node D before moving to the new location. . . . 9
2.3 The routing table for node D after moving to the new location. 9

3.1 How the end nodes send ICMP messages to each other and possible shortest path between the nodes. 28
3.2 Describes the packet loss on the edges for the five test cases. 28

4.1 Convergence times measured at the node N2RLE3 with default configurations for all protocols. 31

Abbreviations

ANSN	*Advertised neighbour sequence number*
AODV	*Ad hoc on-demand distance vector*
Batman	*Better approach to mobile ad hoc networking*
DSDV	*Destination-sequence distance vector*
DSR	*Dynamic source routing*
EIGRP	*Enhanced interior gateway routing protocol*
ICMP	*Internet control message protocol*
IHU	*I hear you*
IP	*Internet protocol*
MANET	*Mobile ad hoc network*
MDR	*Manet designated router*
MPR	*Multi-point relay*
NIC	*Network interface card*
OLSR	*Optimized link state routing*
OLSRd	*Optimized link state routing daemon*
OGM	*Originator message*
OR	*Overlapping relay*
OS	*Operating system*
OSPF	*Open shortest path first*
OSPFd	*Open shortest path first daemon*
RAM	*Random access memory*
RFC	*Request for comments*
RIP	*Routing information protocol*
SSH	*Secure shell*
TC	*Topology control*
TTL	*Time to live*
VLAN	*Virtual local area network*

Chapter 1

Introduction

1.1 Background

There are many different routing protocols that are designed for different intents and purposes. Some protocols are designed to cope well with changes while some are optimized for other objectives. In order to choose the most suitable protocol for your intents, one has to be aware of the network topology properties. For example, today's Internet is a static network with respect to the routers and network infrastructure that create the backbone of the Internet. Even though new nodes are connected and others are removed, the backbone network topology is essentially stable. In contrast, an ad hoc network does not have a fixed network topology and it does not rely on any fixed infrastructure [1]. This poses new requirements on the routing protocols that are being used. For the Internet, all routers in the network contain an up to date routing table in order to select the best route for the traffic. The routing protocol that ensures that this table is up to date is not designed to cope with rapid changes in the network topology. The routing protocols for the Internet are designed for stable and stationary networks. When the routers have found each other once, there is not much change in the routing table which imposes low demands on the update frequency for the routing protocol in use. However, because ad hoc networks does not have a fixed network topology, the nodes in the network must discover the network topology in order to forward information [1]. In a mobile ad hoc network (MANET), the nodes do not only have to discover the network topology once, they have to do so continuously. Because nodes are continuously moving around, this causes the networks topology to change rapidly. When the network topology is no longer persistent, the routing of messages between different nodes in the network gets much more complicated than for a static network. These properties will significantly increase the demands on the update frequency for the ad hoc routing protocol [2]. Furthermore, the nodes in a MANET are usually limited in energy supply, bandwidth and variable connectivity [1]. Therefore, in order to create a reliable network, the routing protocol must be carefully selected to adapt to the characteristics of the network [2].

Mobile ad hoc networks can be applied to miscellaneous concepts in both military and commercial applications. Its concept of building infrastructure-free communication without any planning can be useful in order to maintain the communication among a group of soldiers for tactical operations [2]. Even though radio conditions prevent soldiers to communicate with other nodes directly. The traffic can be for-

warded by other nodes and still reach the whole group. However, it is difficult to find one protocol that will manage all situations well. These types of military networks have nodes that are highly mobile and the communication properties of the radio links are not reliable. The radio link communication is variable in quality and disruption in the communication often occurs. Therefore, in order to assess how different routing protocols manage with packet loss together with a changing network topology, a series of simulations is performed in a tactical network environment.

1.2 Objective

The goal of this master's thesis is to evaluate open source software that implements different routing protocols for mobile ad hoc networks. Furthermore, the project will take in to account the different facets that are of importance for military tactical networks in respect to MANETS. The focus will be on auto configuration of nodes for establishing, maintaining connectivity as well as routing messages between nodes. The maturity of code and if there is a working community around the protocol is also taken into account for the protocols that are selected to be evaluated by simulations. The protocols are evaluated according to the following main questions:

- How resilient is the protocol against disruptions?
- How much traffic overhead is generated in order to maintain the network topology?

This project aims to find a suitable protocol for a mobile ad hoc network that has the properties of a military tactical network.

1.3 Delimitations

There are a number of routing protocols for ad hoc network. However, only a few are implemented as open source software and have a working community which contributes to more stable code. This master thesis includes tests of OSPF, OLSR, Babel and Batman-adv. OSPF is compared as an alternative against the three ad hoc routing protocols. Other well known ad hoc routing protocols are explained in the literature review in order to understand different concepts and problems one face in ad hoc routing. Furthermore, these protocols have many similarities with each other and it improves the overall understanding of the routing protocols that are tested. Nevertheless, these protocols will not be compared against the protocols that are tested since there are no suitable implementations available as open source.

1.4 Methodology

This master thesis consists of three parts. The first part was to perform a literature review in order to decide which routing protocols that would be most suitable for the intended application. Additionally, the most common ad hoc routing protocols were studied to get a more comprehensive understanding of different ad hoc routing techniques. Most of the information was obtained from research papers and RFC documents.

The second part of the project was to build the test environment and define test cases. Saab's internal documentation was used to get an understanding of tactical networks and the Radio Link Emulator (RLE) software. The RLE is a software product developed by Saab which make it possible to emulate realistic radio networks that control delay, varying bandwidth and packet loss. The RLE runs on Linux and make use of Linux traffic control to manage the different traffic streams. In order to create a flexible simulation environment, some modifications were done to the RLE software. The simulation environment was built using two main components which was routing nodes and RLE machines. Together with VirtualBox and VLAN it was possible to create a network with a total of 57 entities that together formed the complete network. The size of the network was partly decided from limitations from the physical computers' RAM which was in total 16GB.

The test cases were designed to provide answer to the problem statement provided in subsection 1.2. Therefore, the focus of the simulation was to test the protocols' resilience against disruption and packet loss but also to compare the amount of overhead traffic that were generated.

The third part of the project was to run and analyse the simulations for the selected routing protocols. The test data that was sent during a simulation was generated with the regular ping command in Linux. To get a full understanding of what was happening during a simulation, the network traffic was logged at almost every node. The logging of network traffic was done with a software called Tcpdump [3]. However, logging all traffic generated a lot of data to be analysed. In order to manage all data a Java program was written using the jNetPcap library [4]. The output from this program was used to create different diagrams in order to evaluate the protocols.

1.5 Related work

There are several papers that evaluate the performance of the protocols that are tested in this thesis. Some of them run simulation software and some are performed on actual hardware. [5] evaluates Batman against OLSR in a network that consists of a 7x7 grid with physical computers inside a 6x12 meter room. The results from [5] clearly states that Batman-adv is the superior protocol for this environment. Another evaluation based on hardware nodes is done is [6] which compares Batman, Babel and OLSR. Main conclusions is that batman has best packet delivery ratio, Babel offers best bandwidth and fastest convergence time while OLSR does not perform very well compared to the other two. However, none of these tests were done in a tactical network set up. Moreover, [7] is an evaluation of OLSR in a military tactical network. The conclusion from [7] is that OLSR is well suited for this type

of network and perform relatively well.

1.6 Report structure

Chapter 2 present the literature review which describes different ad hoc routing protocols. This chapter first introduce the concept of routing in MANETs and it is further divided into two main categories of routing protocols. The categories are proactive and reactive routing protocols with the focus on the proactive protocols.

Chapter 3 describes the test environment which includes the network set up, the test definition and all software that is used. The chapter begins with an overview of a scenario based on a military tactical network. Furthermore, this chapter explains how the tests is carried out and what protocols implementations that are used.

The results from the tests are presented and discussed in chapter 4. The main focus is on overhead traffic and convergence time for various settings for respective protocol. Moreover, the report ends with a discussion and a conclusion based on the results.

Chapter 2

Literature review

2.1 Routing in MANET:s

Routing protocols for MANETs have higher requirements than protocols designed for wired communication [8]. This is mainly due to more limited resources in combination with a continuous topology change. Therefore, a number of MANET protocols have been developed in order to save resources such as bandwidth or energy consumption [2]. Depending on the application, it is important to choose the most suitable MANET routing protocol in order to achieve better performance.

The proactive or table-driven protocols are based on the same concept as routing protocols that are used in wired communication [2]. These routing protocols contain an up to date table describing the connections in the whole network topology [2]. Ideally, at any point of time, all nodes should have the same table showing the real time topology of the network. The main advantage with proactive routing protocols is shorter delay times when two nodes start to communicate. This is because there should always exist a path to all other nodes in the network.

The other type is the reactive or on-demand-driven routing protocols. In comparison with the proactive routing protocols, these protocols do not keep information about the network topology [2]. When one node wants to communicate with another node, it broadcasts a request to find the best route to that node in the network [2]. When they are no communication between the nodes, the path is lost and is not recalculated until further communication is initiated. This saves a lot of overhead traffic but have a negative effect on the delay time.

There are also some protocols which take advantage of both proactive and reactive techniques. These protocols are called hybrid protocols. However, it is not always obvious how a MANET protocol should be categorized. Moreover, in order to select a suitable protocol it is much depending on the type of network and which resources are most critical to be frugal with. The following sections describe the techniques that are used in proactive and reactive routing protocols and the different tradeoffs that must be considered.

2.2 Proactive or table-driven routing protocols

Proactive routing protocols maintain an active route to all other nodes in the network. When a node wants to send data, the path is calculated based on information

in the table. The table is updated frequently, regardless of how often the information is used. However, the benefit comes with short delay times when the data is sent because of the updated path [8]. The path is calculated using a distance vector or a link state technique. The distance-vector technique is often based on the Bellman-Ford algorithm [9] for calculating the shortest path [8]. The Bellman-Ford algorithm has a dynamic programming approach which means all possible paths are considered in the optimization. Furthermore, the distance vector information is exchanged to all the nodes neighbours so the distance information will propagate throughout the network [8]. In comparison with the distance vector approach, the link state technique only sends information about the connections to neighbour nodes. In order to distribute this information to all other nodes, flooding is used [8]. Every node then calculate the shortest path using Dijkstra's algorithm [10] or some other shortest path algorithm. Dijkstra's algorithm is a greedy algorithm which makes it not as versatile as the Bellman-ford algorithm.

The following sections will describe how different proactive routing protocols work. The most important part of the protocols, for example how neighbour discovery and how information is propagated in the network, are explained. The first protocol is OSPF, which is not developed with ad hoc networks as primary are of use. However, with its many configuration options it is still possible to adjust OSPF to better support an ad hoc environment.

2.2.1 OSPF

Open shortest path first (OSPF) is a link-state routing protocol that is often used in the Internet [11]. The protocol is developed by the Internet Engineering Task Force (IETF) and is designed to quickly adapt to new topology changes and still send a minimum amount of routing overhead [11]. However, the protocol is developed to work in a wired network environment and not a wireless mobile ad hoc network which result in longer delay times. A wired, more static, network does not need to adapt to new topologies as often as an ad hoc network. Therefore, it takes longer time for OSPF to propagate topology changes and it also generates less overhead traffic. Nevertheless, in this master thesis, the OSPF protocol is used as an alternative against different ad hoc routing protocols in order to determine if there are significant performance improvements.

In a link-state protocol, all routers in the network contain a link-state database that describes the complete network topology. Furthermore, at any point in time, all routers should have the same link-state database in order to decide how to route the traffic. Every router collects information about its closest neighbours and floods this information to the rest of the routers. This is how topology changes propagate in the network and keep the link-state database up to date. The routing table in every router is built by information from the link-state database. Furthermore, the routing table contains the destination to every router in the network together with the distance and which next hop router the data should be forwarded to in order to reach the destination. It is always the shortest path that is saved in the routing table. It is calculated using Dijkstra's algorithm [10]. OSPF has five types of messages that are listed in table 2.1. These five messages describe the overall functionality of the OSPF protocol.

Message type	Description
1. Hello message	For discover and maintain neighbours
2. Database description	Description of database and adjacent nodes
3. Link state request	Request update from neighbours database
4. Link state update	Receive update from neighbours database
5. Link state acknowledgement	Acknowledgement of the update

Table 2.1: *Message types of the OSPF protocol*

The hello message is used to discover and maintain neighbours and are frequently sent out to all neighbours. However, this process is a bit different depending on which type of network that is used. In a regular Ethernet network, the broadcast network type is most commonly used. In a broadcast network, the hello protocol selects a designated router and a backup designated router in order to minimize the routing overhead traffic. Moreover, the overhead traffic is only sent to the designated router that can collect information from several nodes and distribute the information to all other routers [11].

In a wireless network, there is usually a point-to-multipoint connection between the nodes. An OSPF network which has a point-to-multipoint configuration does not select a designated router. Therefore, there is more overhead traffic on this type of network in comparison with a broadcast network. Moreover, the connections between the nodes in a point-to-multipoint network are managed as a set of point-to-point connections. The hello message is frequently sent to all neighbours and it contains a router priority number, hello interval, router dead interval and a list of all neighbour routers. The router priority number is used for selecting the designated router, so for a point-to-multipoint configuration this is not relevant. Furthermore, the hello interval and the router dead interval specifies how often a hello packet is sent out and how long time before a node is no longer considered to be active.

The database description packet, message type 2 in Table 2.1, is sent after two nodes have recognized each other from the hello message. The database description packet contains information of the database and the adjacent nodes. The two nodes might need to exchange several database description packets in order to send all information. When a node has got information of the database it might notice that some parts are out of date. This is what the link state packets are used for. The node request an update for a specific part of the database and the other node answer with a link state update for that part. However, the link state update packet is flooded in the network which makes is useful for all nodes. The nodes that receive the information will answer with an acknowledgement either as a multicast or unicast.

There are several attempts to adapt OSPF in order to improve the performance in ad hoc networks. Three protocols that are an extension of OSPF are briefly mentioned below. However, none of these protocols seems to have a solid implementation with a working community and are not considered in the evaluation.

2.2.1.1 OSPF-MPR

Multi-point relay (MPR) is a concept taken from OLSR in order to create a more effective flooding procedure [12]. Each node maintains a set called multi point relay set which contains a subset of all 1-hop neighbours. These nodes are selected so

that all 2-hop neighbours can be reached. In this way, the routing overhead data is minimized in comparison to regular OSPF [12].

2.2.1.2 OSPF-OR

Overlapping relay (OR) is also based on the concept of OLSR but with some further modifications [13]. The goal for this approach is that a node should only receive the same data packet once or as few times as possible depending on the network complexity. If the nodes share information of which node they have in the multi point relay set they can find overlapping relays and decide that only one node should keep the overlapping node in the set. In this way the flooding procedure will be more efficient [13].

2.2.1.3 OSPF-MDR

Manet designated router (MDR) is based on the designated router concept in OSPF with some modified functionality to make it to work for a MANET [14]. In regular OSPF an adjacency is formed between two routers only if one of them is the designated or backup designated router. This is a problem in a point-to-multipoint wireless network because of the many connections between the nodes. However, this could be solved by using a spanning tree and let the edges be adjacency and the non leaf nodes, the nodes that have more than one connection in the spanning tree, be Manet Designated Router (MDR) nodes. Every router decides if it should be a designated router based on local information received from hello messages. This means that a spanning tree is not always created globally but rather several local spanning trees are created which leads to non-optimal flooding between some regions in the network [14].

2.2.2 DSDV

Destination-sequence distance vector (DSDV) protocol was developed by[15] in 1994. DSDV is based on RIP [16] which is a routing protocol designed for wired networks [15]. In order to adapt DSDV to an ad hoc network the developer had to deal with the looping problem that comes with using the Bellman-Ford algorithm [9] in an ad hoc network environment. The design goal was therefore to preserve the simplicity of RIP but to solve the looping problem with the Bellman-Ford algorithm [15]. In order to do this, [15] introduced sequence numbers on every entry in the routing table which made it possible for the nodes to determine stale routes. When the distance vector information is exchanged the receiving node will always know if the table should be updated or if the information is outdated. If all nodes have an up to date routing table, there should not be any loops. All nodes must frequently broadcast the table entries to its neighbours in order to propagate changes in the network topology. When a node receives an update containing the distance information and a sequence number, it compares the sequence number in its own table in order to decide what to do with the information. If the sequence number in the received information is higher than the one already saved in the table, the new information will replace the old one in the table. If the received sequence number are lower the information will be ignored [15]. However, if they are equal the node must compare the distance information and the shortest route will be saved [15].

To explain how this works Figure 2.1 shows an example ad hoc network. The initial routing table for node D is shown in Table 2.2.

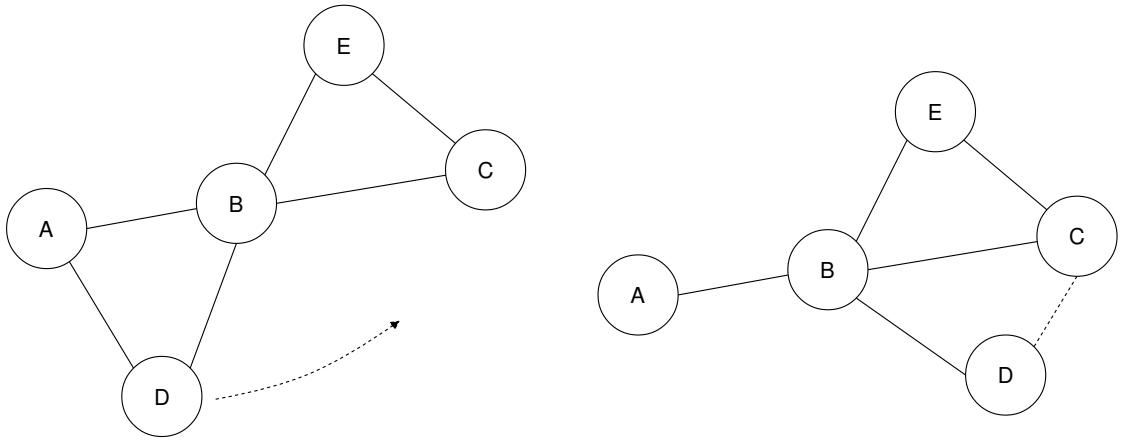


Figure 2.1: *A mobile ad hoc network where node D moves to a new location. Changes in the routing tables must always be updated.*

Destination	Next hop	Distance	Sequence number
A	A	1	A-103
B	B	1	B-235
C	B	2	C-87
E	B	2	E-201
D	D	1	D-196

Table 2.2: *The routing table for node D before moving to the new location.*

When node D detects that the link between node D and node A has failed, node D will update the distance information to node A to infinity and increment the sequence number by one. The information will be broadcasted to the other neighbours in order to propagate the information through the network. Moreover, that means that no node will have a route to A that goes through D. The link to node A will be down until some node (node D itself) will send out a message with higher sequence number that do not have infinity as destination length. As soon as node D has reached its new location, node A will eventually receive an update message containing a higher sequence number which means node A will update its routing table and replace the previous infinity value regarding node D. When all these are done, the routing table for node D will be as in Table 2.3.

Destination	Next hop	Distance	Sequence number
A	A	2	A-104
B	B	1	B-235
C	B	1	C-88
E	B	2	E-201
D	D	1	D-198

Table 2.3: *The routing table for node D after moving to the new location.*

DSDV suffers from a fluctuation problem, that is when one node have to change the table information several times successively because the new information is delivered from several nodes in a non-optimal order [17]. However, in order to damp the fluctuations, the node does not immediately retransmit the new changes in the routing table. If that was the case, the same problem would propagate through the network and cause the same problem everywhere. Instead, the node waits to see if it will receive more possible routes, to the same destination, that are shorter than the one previously added. This behaviour is good with respect to the traffic overlay that otherwise would be created. However, the network might suffer from the delay times that will arrive from the waiting.

2.2.3 OLSR

Optimized Link State Routing (OLSR) is another proactive routing protocol where every node uses the complete network topology in order to calculate the shortest path to the destination. OLSR is an improvement on the OSPF [11] routing protocol that is used for wired networks. However, the distribution of routing information in OLSR is different compared to OSPF. In order to minimize the overhead flooding in the same region, the flooding is only done by some special nodes called multipoint relays (MPR:s) [18]. This will reduce the number of retransmissions that are needed in order to flood a message to all nodes in the network. The nodes that are selected to be MPR nodes must keep track of which nodes that have chosen it to be an MPR node. Furthermore, this set of nodes are called selector nodes and must be continuously updated by the MPR node [18]. This is the minimal information that is needed in order to determine the topology of the network [18]. However, to explain the OLSR protocol more thoroughly, the protocol can be divided in three core functionalities. These are; link state flooding, multipoint relaying and neighbour discovery. The neighbour discovery process is described in Figure 2.2 where node A has just moved in the area of node B.

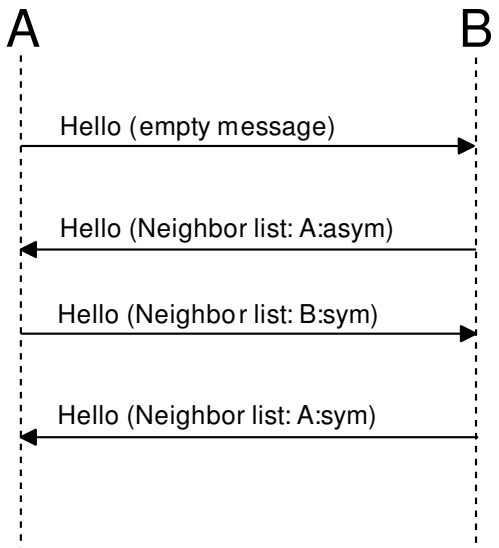


Figure 2.2: *The messages sent during a neighbour discovery procedure in OLSR.*

Node A and B frequently sends out hello packets to discover if there are some other nodes within reach [18]. When node B receives the first hello packet it does not contain any information about other nodes than A. Therefore, node B will announce itself as a neighbour to A and send a hello packet containing this information [18]. When node A receives the packet it will notice that its address is included in the message and will therefore set B to a neighbour [18]. From this point the hello packets sent from A will include an address to B specifying that B is a neighbour to A.

The multipoint relay process starts with that each node selects a set of MPR nodes that are in its 1-hop neighbourhood. The nodes that are selected should together reach all 2-hop neighbours from the initial node. Now it is only this MPR set that will retransmit a broadcast from the initial node. The other nodes that are in range from the node will only receive the broadcast and use the information, but not retransmit. This is illustrated in Figure 2.3 where the comparison between having a MRP set and using a regular flooding approach.

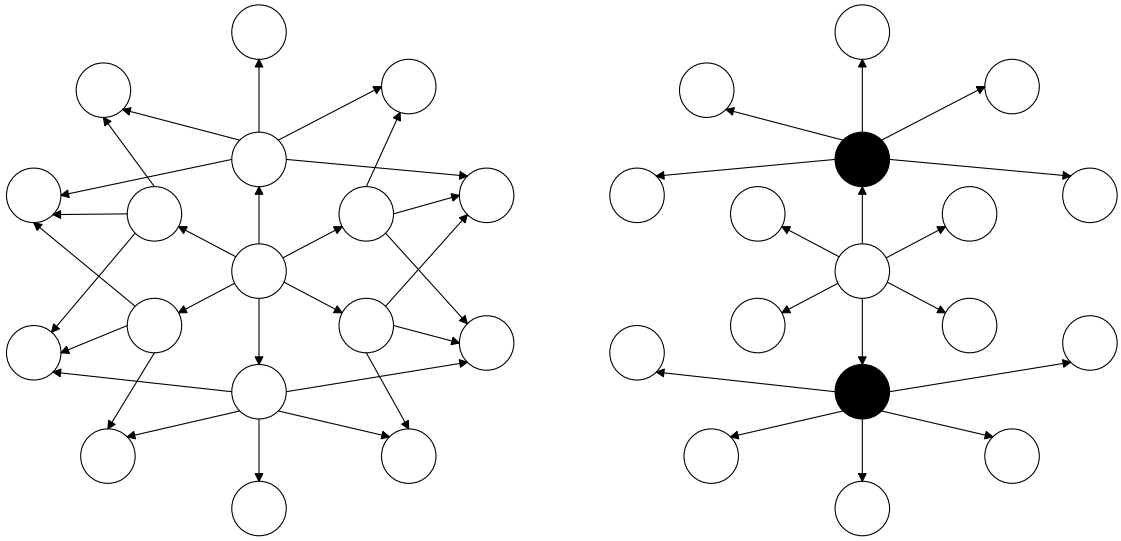


Figure 2.3: *The difference of using MPR nodes in the flooding procedure.*

Figure 2.3 shows that the number of transmissions is significantly less in the use of a MPR set. In order for a node to keep track of which other nodes that have selected it in a MRP set, all nodes maintain a Multipoint Relay Selector set (MPR selector set) that contains this information [18]. Furthermore, a hello message is used to tell the node that is has been selected in a MPR set. The hello message is sent frequently and is also used to for neighbour discovery as previously described [18].

As OLSR is a link state protocol it will frequently flood the network with information about the links. However, this will be done according to the MPR optimization. Every node maintains a set that contains information about links to its neighbours, this set is called the link set. The link set information is distributed to other nodes in a topology control (TC) messages [18]. The TC messages contain the addresses to all neighbours of the node together with a advertised neighbour sequence number (ANSN) [18]. The ANSN is a sequence number generated by the node that sent the TC message [18]. It will not be increased for every TC message but instead only when the data is changed in the message [18].

When a node receives a TC message the following will be done. If there is no entry for the node that first created the TC message this will be created and all the addresses in the TC message will be added with a validity time together with the ANSN. If it already exist an entry, the first thing to be done is to compare the ANSN to determine if the data should be replaced or not. It is only when the ANSN is higher than the already stored ANSN that the addresses are replaced with the new data. However, if the ANSN is equal to the stored ANSN the data is not changed. In this case the validity time, that determines for how long time a link is valid, is increased.

Even though OLSR got much functionality to make it efficient, the implementation of OLSR differs from the RFC [18]. Some of the proposed solutions does not seem to work as well as expected. Therefore, the developers at *olsr.org* have been working on improvements in order to create a fully function ad hoc routing protocol [19]. Some of these improvements are explained in the software section where the implementation from *olsr.org* is explained.

2.2.4 Batman

The Batman protocol [20] (better approach to mobile ad hoc networking) is another proactive routing protocol that is developed for mobile ad hoc networks. In Batman, the nodes do not keep track of the complete network topology as in OLSR. The nodes only know the best next hop to all other nodes in the network. In this way, the best end-to-end path should be used when two nodes are communicating. The batman-adv algorithm makes use of only one message type called originator message (OGM). This message is used for both neighbour discovery and to determine the best next hop node to a specific destination. A node first broadcast an OGM to all its neighbours which will then be aware of the node existence. The OGM contains the address of the originator (the node that first created the message), the address of the sender, TTL value and a sequence number. After the neighbours have received the OGM they will rebroadcast the message to their neighbours. This time the address of the sender is changed and the TTL value are decreased. The originator address and the sequence number remain unchanged. Because every node is retransmitting the message, the OGM will be flooded to all nodes in the network. In order not to starve the network, the node makes use of the sequence number to be sure that the OGM is only broadcasted once.

In order to choose the best next hop neighbour, the node will simply choose the first OGM that arrives from the originator node. This is based on the assumption that other OGM that may arrive later have travelled on unreliable links and might have been exposed to delays or packet loss. If a new OGM from the same initiator arrives from another neighbour and this OGM has a higher sequence number, the current next hop neighbour will be replaced by the new one. In this way the current best next hop neighbour will always be updated if there are any changes in the local topology. Note that this information is not propagated to all nodes in the network which saves a lot of overhead traffic.

2.2.5 Babel

The Babel protocol [21] is a proactive routing protocol that is developed with properties from DSDV, AODV and EIGRP. Therefore, Babel is usually described as a loop-avoiding distance-vector protocol. Babel is designed to work well in both wired networks as well as wireless mesh networks. The distance-vector approach is obtained from DSDV with the Bellman-Ford algorithm. However, some additional properties are implemented in order to increase performance. The loop avoiding properties is obtained from EIGRP and the diffusing update algorithm [22]. This algorithm make use of feasibility condition that must be met before a new route is added to the routing table. Therefore, it can guarantee that only loop-free routes are selected. A detailed description of this algorithm is explained in [22].

In order to find neighbours, Babel periodically sends out hello packets. When a neighbour receives a hello packet it will answer with an IHU (I Heard You) packet. This will confirm that the two nodes now know about each other. The hello and IHU packet are only transmitted to the closest neighbours and are never forwarded. In order to distribute the network topology information every node sends out its routing table information to all its neighbours. These update messages are sent out periodically but they can also be triggered. If a node has made changes to its routing table that other nodes should know about a triggered update should be sent out. In order to be sure that the update is received by the neighbours, the node can require acknowledgements from the neighbours. However, this is only a good choice if the number of neighbours is fairly small. Otherwise, the best option could be to resend the same packet a reasonable number of times to make it likely to reach the neighbours without sending to much traffic on the network.

When an update message is received, all new routes are checked against routing loops before they are accepted. If there are significant changes to the routing table, the node must also send an update message to its neighbours. This is how topology changes propagate throughout the network. A node can also send a request message for a specific node. If the node that receives the request has the specific node in its routing table and the sequence number is higher than the one sent in the request it must send an update.

2.3 Reactive or on-demand-driven routing protocols

Reactive or on demand routing protocols have a very different approach to find the routing path in comparison with proactive or table driven protocols. Because routes are only produced on demand, and only maintained as long as requested, this would let us ignore non used links completely. If there are no request of communication between nodes there should be no overhead traffic sent in the network. This will result in both minimizing energy consumption and traffic overhead compared to a proactive routing protocol [2]. However, the lack of communication also result in a disadvantage against proactive routing protocol with longer delay times as a result [1]. If the network is large, the long waiting times might preclude the use of a reactive routing protocol.

2.3.1 AODV

Ad hoc on-demand distance vector (AODV) is a reactive protocol which means it will only find routes on demand [23]. However, AODV has many similarities with the previously mentioned proactive protocol DSDV. Both protocols use the distance vector technique to calculate the shortest path to a node. Furthermore, AODV makes use of sequence number, in order to avoid the looping problem, in the same way as DSDV does.

AODV has three main types of messages in order to deal with route calculations, these three types are; route request, route reply and route error [23]. AODV only maintains routes that are in use, all other routes will eventually be deleted in the routing table [23]. If a node request a new route, that node will broadcast a route request packet to all neighbours that will retransmit the broadcast [23]. A node will only retransmit the same route request once in order not to flood the network several times. All nodes on the way to the destination will add the source node in a new table entry in order to be able to answer with a route reply that might be sent later. If the source node already exists in a table it might be updated depending on the sequence number that is included in the route request packet. The table entry is updated if the sequence number in the route request packet is higher or equal than the already saved sequence number and the hop count to the source is less than the previously stored value. Before retransmitting the route request, the hop count field is increased by one in order to keep track of how many nodes the packet has passed.

Once the request packet is received by the destination node, or a node that has an up to date route to the destination, a route can be established. This is done by sending a route reply packet back to the initial sender node. The route reply is a unicast reply and will only traverse through the shortest path in the network. When a node receives the route request packet it knows it is now part of an established route that has to be maintained until it is no longer in use. If there are no packets travelling along that path it will time out and the table entry will be deleted. The time out value is included in the route reply packet.

If a node discovers that a neighbour is no longer in reach it will create a list of all unreachable destinations that comes as a result of the lost neighbour. A route error message will be transmitted to all neighbours that are part of a route that are affected of any unreachable destination from the created list. These neighbours will retransmit the route error message to the effected neighbours making the error message to eventually inform the source node about that the rout is no longer valid.

A node can share routing table information with its neighbours by sending periodical hello messages. When a neighbour node receives a hello message it must create an active route to that node in order to maintain its neighbour set. The hello message is actually a route reply message with the destination IP and destination sequence number from the node itself. The message has a time to live (TTL) value set to 1 in order to only send it to the 1 hop neighbours. Furthermore, by sending these messages frequently, the neighbour nodes can determine when the connection to the node is lost and should then broadcast an error message as previously described.

2.3.2 DSR

Dynamic source routing (DSR) [24] protocol is another reactive or on demand routing protocol for mobile ad hoc networks. There are two main parts of the protocol,

route discovery and route maintenance. In contrast to AODV, that sends frequent hello messages in order to keep track of close by neighbours, DSR does not send anything when it is not requested. Therefore, when no links are maintained and no new links are requested, the traffic overhead is zero [24].

Every node maintains source cash with all known destinations together with the complete path. If a node wants to send data to a destination that it does not already know the path to, the discovery process is invoked. The route discovery process starts with the node sending a request message to all its neighbours containing the destination address, the source address, an ID of the request message and a list with all intermediate nodes between the source and destination. All receivers of this message will add itself to the list in the request message in order to form a route and then retransmit a broadcast. The nodes will only retransmit the same request message once because of the request ID that is included in the message. Once a node finds itself as the destination address, the destination is reached. If this is the first time the node receives this message it will answer with a request reply message to the initial source node. If the node does not already have a route to the initiator in the node source cash, the node will perform its own route discovery. In order to not create an infinite route discovery loop between these two nodes, the route reply will be encapsulated in the route request message. When the initiator has received the route reply the route will be added to the source cash and can now be used for communication.

However, the network topology will change so there is no guarantee that the route will be valid when the node wants to send data. Therefore, every node along the path must confirm that they have managed to forward the data that is sent from the initiating node. In Figure 2.4 there is an example where one node along the route is no longer in reach.

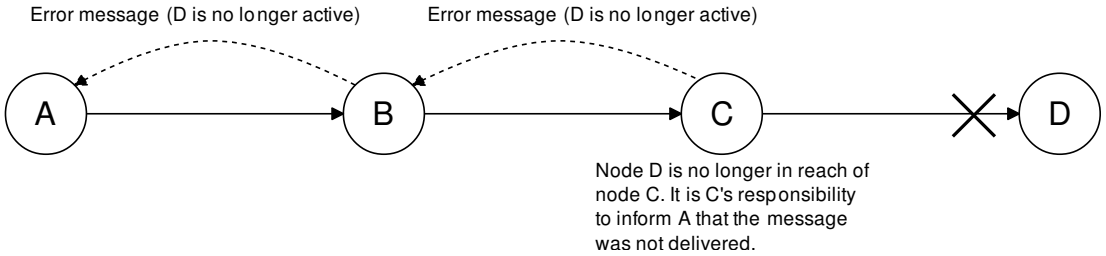


Figure 2.4: *Example how an error message is created in order to inform the source node that the destination can't be reached*

Node A wants to send data to node D using the source route produced by the route discovery procedure. The route is A->B->C->D, but node D is no longer active. Node C is the one responsible for the delivery to node D. When C does not get a response it must inform node A about the broken link with a route error message so that A can change its source cash regarding this route.

Any link can collect information to its source cash by listening to the traffic that is received, even though the traffic is not intended for that node. It could be route requests or route replies that give the node additional information about the network. Moreover, a node is allowed to reply to a route request packet if it has a route to the destination in the source cash. But before sending a route reply to

the initiator, the node must verify that there are no duplicates in the route list. If that was the case, it would create an unnecessary long route where the traffic was forwarded by the same node twice. Furthermore, it can create even more serious problems when answering to someone else's route request. Imagine the case when there are several close nodes together and all of them know about the destination, and therefore all of them want to answer the route request. If all nodes were to answer at the same time, it would create collisions and some packets might be lost. In order to solve this problem, the DSR protocol does not allow every node to answer at the same time. This is achieved by a randomized function that calculates a waiting time before the node sends the reply. Moreover, this function is based on the number of hops from the sending node which means that the close by nodes will always answer first. Overall, a major disadvantage with DSR is that the header size is not fixed. It grows with the number of hops because of the list that describes the path.

Chapter 3

Test environment

The network set up is based on tactical network scenario. Therefore, in contrast with a complete ad hoc scenario, all nodes are not able to communicate with all other nodes directly. Even if they are close enough so it is technically possible, the traffic is limited to a few predetermined routes. The hierarchical structure in a military battalion corresponds to the allowed flow of traffic. A battalion is a military unit that consists of several companies. The number of companies varies depending on country, but it is usually between five and ten. Figure 3.1 describes a part of a battalion and how the different units are allowed to communicate with each other.

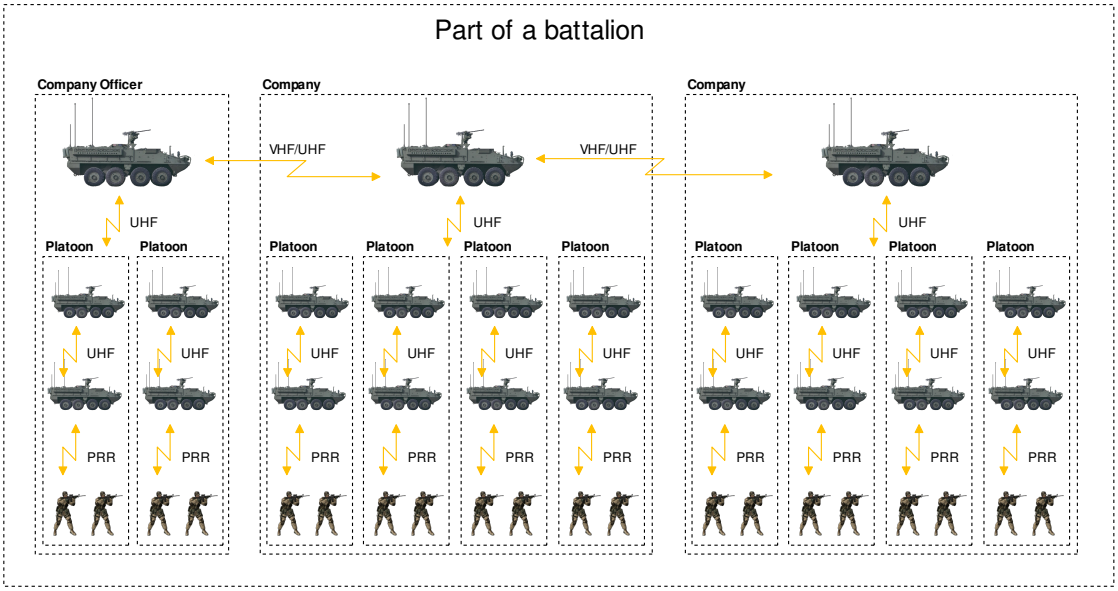


Figure 3.1: *Describes how the communication is allowed in a battalion. The communication is hierarchical and must follow chain of command.*

The three largest vehicles in Figure 3.1 are the highest ranking officer in each company. All information that comes in or is sent out from the company must pass through this vehicle. The chain of command must not be broken which makes a full ad hoc scenario irrelevant.

The company vehicles represented by the three largest vehicles in Figure 3.1, share one common radio network and can therefore communicate with each other.

Furthermore, these vehicles have another radio network which can communicate with the platoon leaders. The platoon leaders are the first vehicle in every platoon in Figure 3.1. In the same way, the platoon leaders are connected to two radio networks, one that includes the other platoon leaders and the company leader and one that includes the rest of the platoon. A vehicle in a platoon can also communicate with the soldiers in the same platoon. This is done via a personal role radio (PRR) which all soldiers are equipped with. However, this master thesis is not considering the soldiers and the communication with PRR because of the increasing number of nodes.

The test environment is based on a battalion with four companies that each has eight vehicles. The allowed communication routes are the same as Figure 3.1 describes. The ad hoc scenarios are created between the platoon leaders where traffic can be routed within the same company. It is also created between the company leaders which are using a common radio network. A more detailed description of the test environment is describes in the network set up section.

3.1 Software

This sections describes which software products that is used in the network set up. First, the virtualization software that is used to create the virtual network is explained. Second is the RLE software that is used to emulate the characteristics of a radio network. Then, there is a overview description of the selected software that implements the routing protocols that are used in the test.

3.1.1 VirtualBox

VirtualBox is an open source software product which can be used for creating virtual machines (VM) [25]. The first step is to create a virtual machine image where the new operating system can be installed. The new operating system that is running inside the virtual machine is called a guest operating system (guest OS) while the operating system running outside VirtualBox is referred to as the host OS. If one need to have several virtual machines that looks the same, one can create a clone once the guest OS is installed on the first virtual machine. There is two possible ways to make a clone, either a full clone or a linked clone. A full clone is a complete copy of the original virtual image, so there will be two identical virtual machines. After the clone is created, the new clone is not dependent on the original image in any way. The other possibility is to create a linked clone which means that the clone will use the parent virtual machine image as a base point when it creates the new machine [25]. This will require less disk space than creating two identical copies. If one need to have several machines that are going to use the same operating system, the best solution is to create linked clones in order to save computer resources. As any regular computer, the virtual machine can be connected to one or several networks. To illustrate how this works Figure 3.2 shows two VM:s running inside VirtualBox on a host computer.

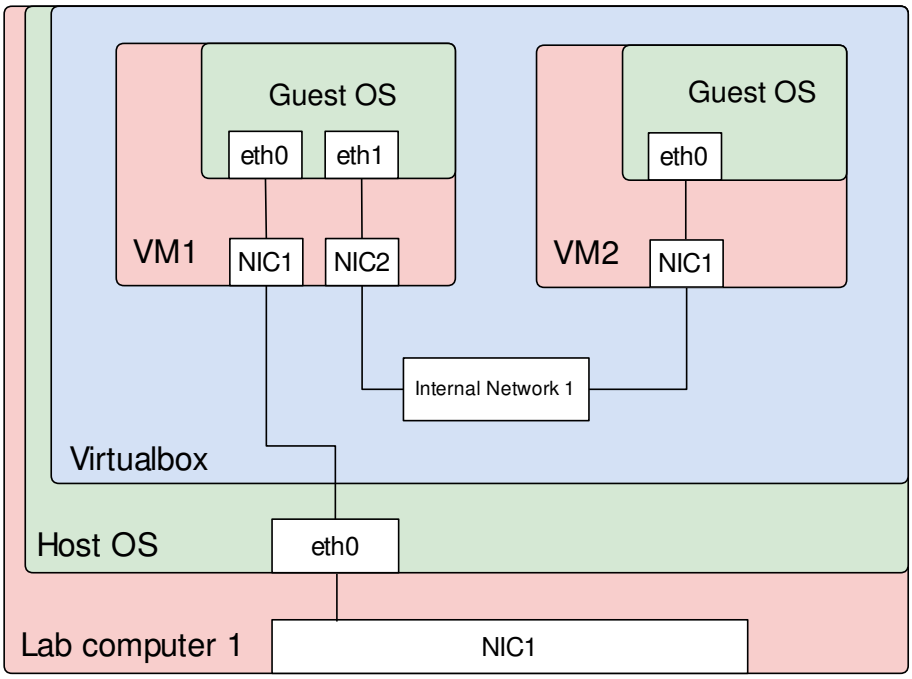


Figure 3.2: *Two virtual machines running on a host computer. The virtual machines are connected with an internal network inside VirtualBox.*

Moreover, the virtual machine has several virtual network interface cards (NIC:s) that can be configured in different ways depending on the user needs. In figure 3.2, VM1 has two NIC:s and VM2 has one NIC. NIC1 on VM1 is configured as a bridged connection to the host OS. This means that VM1 and the host OS can operate on the same subnet. Furthermore, this allows other computers to access the guest OS on VM1 in the same way as they access the host OS.

There is one network option in VirtualBox that is essential to this master thesis, and that is the possibility to create internal networks. An internal network works in a very similar way as a bridged network, the main difference is that the created networks are only accessible inside VirtualBox and not from the outside. In Figure 3.2, NIC2 on VM1 are connected to an internal network with the name "Internal Network 1". In the same way, NIC1 on VM2 is also connected to an internal network with the name "internal Network 1". It is possible for different virtual machines to access the same internal network if they enter the same name of the internal network. Therefore, one is able to connect different virtual machines to each other and create a large virtual network consisting of virtual machines.

3.1.2 Radio Link Emulator (RLE)

To be able to simulate a radio network it is necessary to use a software product that can handle all the natural events that can occur in wireless communications. The software that is used for the simulation in this thesis is called Radio Link Emulator (RLE) and is developed by Saab. It is developed in order to emulate characteristics in radio networks such as delay, varying bandwidth and packet loss. The RLE is running in a separate virtual machine and is connected to the other nodes using VirtualBox internal networks. This makes it very flexible because it can make any

traffic stream get the characteristics of a radio network. The RLE is based on Linux and its main functionality makes use of Linux traffic control. Linux traffic control consist of a queuing system where packets gets into different queues depending on if they are marked with a specific token [26]. In the queue it is possible to drop, rearrange and also determine the dequeue rate of the packets [26]. This makes it possible to create different types of traffic flow on different interfaces. It is important to add that the RLE does not provide any routing; the traffic will only be forwarded from one interface to one or several others according to the specification of the user.

3.1.3 Quagga

Quagga is an open source routing platform that supports various routing protocol. The core component in Quagga is the zebra daemon which communicates with the Unix kernel. The zebra daemon also works as an interface to the implemented routing protocol. For example, OSPF is implemented in the OSPF daemon called ospfd. In order to run ospfd, the zebra daemon must be running and configured with the proper settings.

The Quagga version that is used in this thesis is version 0.99.20.1. Although it supports implementations of Babel, this implementation is not used. According to the Babel homepage, the Babel implementation in Quagga suffers from many bugs. OLSR support is under development for Quagga but in this stage it seems to be best to go for the standalone daemons instead which are independent from Quagga. Thus, for the tests, Quagga is used only for running the OSPF daemon.

3.1.4 Babeld

The Babel implementation that is used in the tests is the standalone daemon version 1.3.0-1. The Babel daemon is easy to configure and run. However, the software suffers from some bugs that appeared during testing and configuration. One explanation to this could be that Babel is still a quite new protocol and the community working with the implementation has still thing to improve. However, the community seems to be active and releases new versions of Babel regularly.

3.1.5 OLSRd

The implementation of OLSR that is used in the tests is olsrd which is a standalone daemon developed by the community at olsr.org [19]. The version of olsrd that is used is 0.6.1-5. Olsrd differs in some ways from the suggested implementation described in the RFC. For example, in order to decrease the overhead traffic a new feature called fish eye was introduced. It is based on limiting the impact of a sent TC message. A close by neighbour should get a TC message more frequent than a node that is more distant from the initiating node. To make this happen, the fish eye mechanism change the time to live (TTL) value for the TC message. TC messages with a small TTL are sent more frequent than TC messages with a high TTL. In this way, the most up to date information are only sent to the close by nodes. However, if the TC message is sent less frequent, the probability of routing loops may increase if the topology changes. In order to prevent routing loops, the TC messages must be sent more frequent than the hello messages.

3.1.6 Batman-adv

There are two main implementations of the Batman protocol, one which is working on the network layer and another working on the link layer [27]. The one working on the link layer (OSI layer 2) is called Batman-adv and it is this protocol that is used in this master thesis. The other implementation, Batmand (OSI layer 3), is not developed any further and is therefore not that interesting for this project. The version of Batman-adv that is used is 2011.4.0.

Routing on the link layer differs quite a lot from routing on the network layer. One thing is that the protocol cannot make use of the IP address and the kernel routing table because these are both layer 3 entities. Instead, Batman-adv has to make use of a kernel module to directly handle the Ethernet frames on layer 2. This comes with both advantages and disadvantages. The main advantage is performance improvements. It is much less costly to process a frame on layer 2 than a packet on layer 3. Furthermore, because the routing is done on layer 2, one can run any other protocol above batman-adv on layer 3 such as IPv4, IPv6 or some other protocol. However, there are some disadvantages as well. Because of the kernel module that handles the layer 2 frames there is no routing information forwarded up to layer 3. This means that outside of the kernel module, one is unaware of the network topology. From the OS view, all computers in the network are link local and can be accessed within one hop. This might not be a problem, but one should be aware of that the kernel module handles everything in terms of routing and the only information that goes up to layer 3 is the protocol running above batman-adv.

3.2 Network set up

In order to conduct the tests in a credible environment a tactical network set up is used [28]. The tactical network is created with the use of several radio networks where each node is limited to one or two radio networks. Figure 3.3 describes the logical structure between all 36 nodes that are used in the tests. The 36 nodes are together forming a battalion with 4 identical companies.

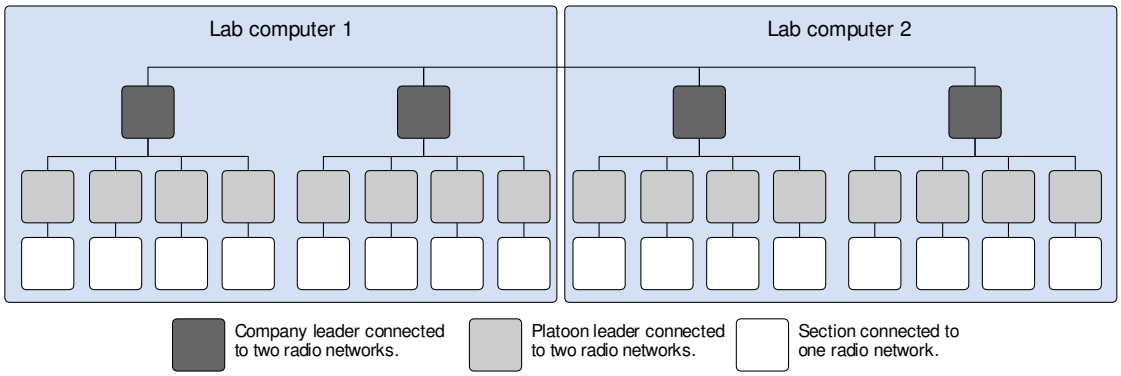


Figure 3.3: *Overview of the routing nodes and how they are connected in a tactical network set up.*

Each company has a leader which is represented by the dark grey nodes. These 4 nodes are sharing one radio network with each other. This means the traffic can

be routed in any way within these four nodes depending on the current connectivity of the network. Each company consists of 4 platoons which consist of 2 sections. Each section has a platoon leader, represented by the light gray nodes. The platoon leader is connected to two radio networks. One radio network with the rest of the platoon and one radio network in order to communicate with the company leader. All platoon leaders in the same company are sharing one radio network. That means this radio network include five nodes where traffic can be routed through each other. A clear representation of the different radio networks and which nodes that are sharing the same radio network are presented in Figure 3.4. Furthermore, each platoon is forming a radio network that connects the two sections together. Moreover, the traffic that is sent by the sections represented by the white nodes in Figure 3.3 must go through the platoon leader nodes in order to reach the rest of the network. This becomes clearer in figure 3.4 where on node in a red radio network only has one communication path.

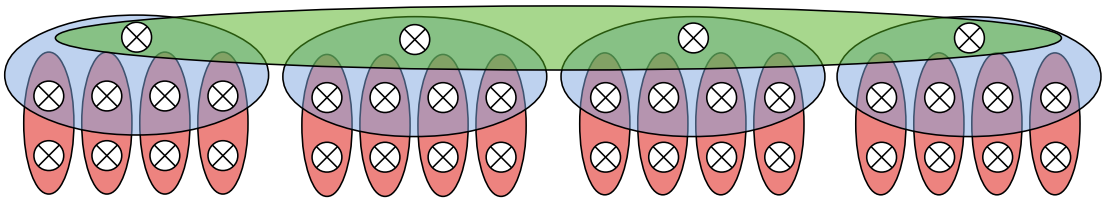


Figure 3.4: *Overview of the routing nodes distributed among different radio networks.*

Every node in the network is running on a separate virtual machine where the routing protocols are configured. The network set up makes use of three physical computers which are referred to as lab computer 1, 2 and 3. Lab computer 1 and 2 are illustrated in Figure 3.3 which shows the distribution among all routing nodes. However, it is not only the routing nodes that are running in Virtualbox. The RLE software which simulates the radio networks are also running in different virtual machines, one virtual machine for each radio network. All RLE are running on lab computer 1 and 2 except for one that is running on lab computer 3. Lab computer 3 is used to connect these two computers together by running the RLE that are connecting the four company lenders together. Furthermore, this RLE is not running in Virtualbox but directly in the host operating system. How this is done will be described in more detail later.

Because of the need of a credible test environment, all links between all nodes must be controlled in manageable way. Therefore, one RLE is used in order to emulate one radio network. As previously described, the network set up in Figure 3.4 consists of 36 nodes and 21 radio networks. Thus, there are a total of 56 virtual machines running on lab computer 1 and 2 plus one RLE running on lab computer 3 in order to connect the computers together. To describe how the RLE machines works, Figure 3.5 shows the network topology of one company consisting of 9 nodes and 5 RLE machines. The nodes are named according to which RLE they are connected to. Thus, the node that is connected between RLE1 and RLE5 is named N1_RLE3_RLE5 as showed in Figure 3.5. In order to be able to control the RLE from the physical computer, all RLE machines have a bridge on eth0 through VirutalBox. This makes it possible to connect to the RLE machines and change the link's connectivity. However, it is important to underline that eth0 in the RLE is

not part of the RLE bridge that forms the connection between the nodes. If that was the case, traffic might be routed in an undesirable manner. In the same way, all routing nodes have a bridged connection to the physical computer. This makes it possible to access the routing directly from the physical computer. A full network topology is provided in Appendix A.

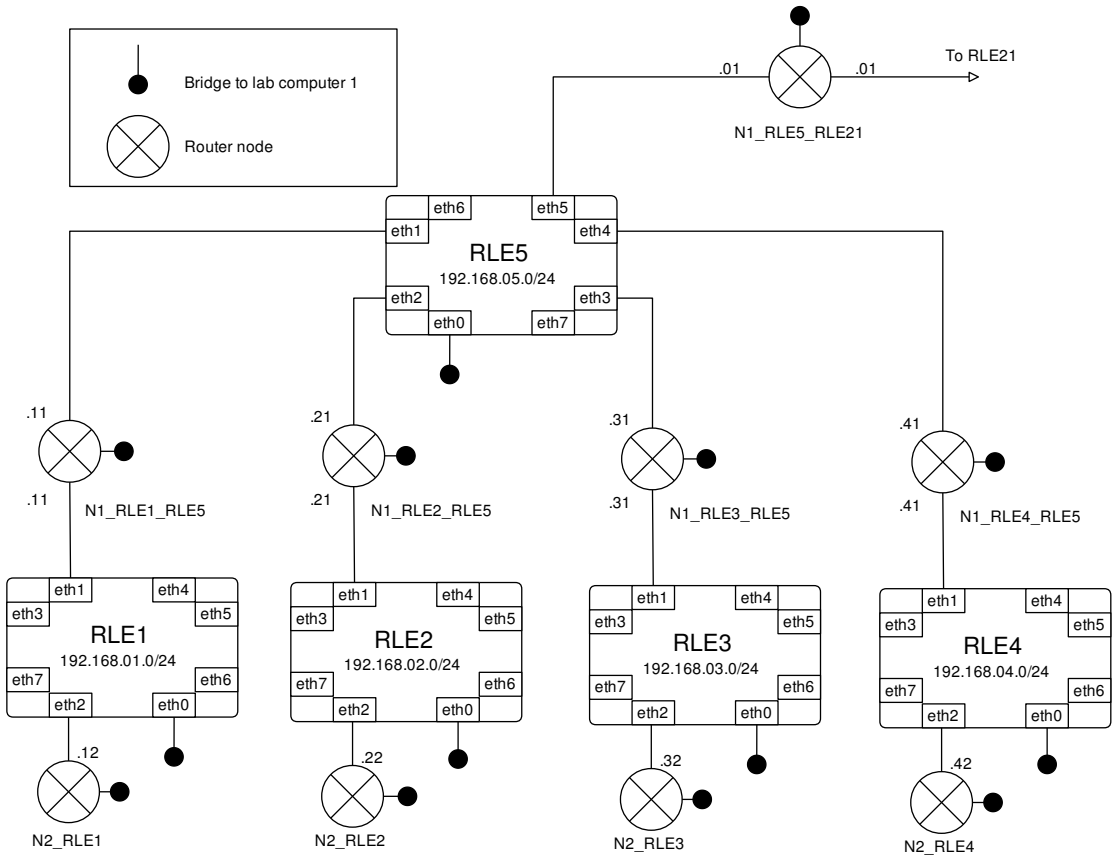


Figure 3.5: *Example of how the RLE and routing nodes are connected together.*

Figure 3.5 shows 5 RLE machines and 9 nodes which are forming one fourth of the complete network. Each RLE is configured with its own subnet. Therefore, the nodes that are connected to two RLEs are also connected to two subnets (two different radio networks as previously mentioned). In order to define an understandable IP-address distribution among the nodes, the last 8 bits are always the same for one specific node. For example, node N1_RLE3_RLE5 that are connected to both RLE3 and RLE5 have two interfaces, one to each radio network. The IP-addresses for these interfaces are 192.168.05.31/24 for the RLE5 subnet and 192.168.03.31/24 for the RLE3 subnet. Bit 17-24 describes which RLE the address belong to. The last 8 bits is in this case the decimal number 31, this will be the same for both interfaces. Because this node is node nr 1 to connect to RLE 3, the IP-address ends with a 1. In the same way, node number 2 that are connected to RLE3 has the IP-address 192.168.03.32/24.

In order to connect a node to the correct RLE, VirtualBox internal networks are used. As described in Section 3.1.1, if several virtual machines connect to the same internal network, they can communicate. However, two routing nodes will

never be connected to the same internal network because the traffic must always go through a RLE first. Otherwise the ability to emulate a realistic radio network is lost. Figure 3.6 describes how RLE1 and N1_RLE3_RLE5 are connected together on lab computer 1. Note that this is just 2 of 28 virtual machines that are running in VirtualBox on lab computer 1.

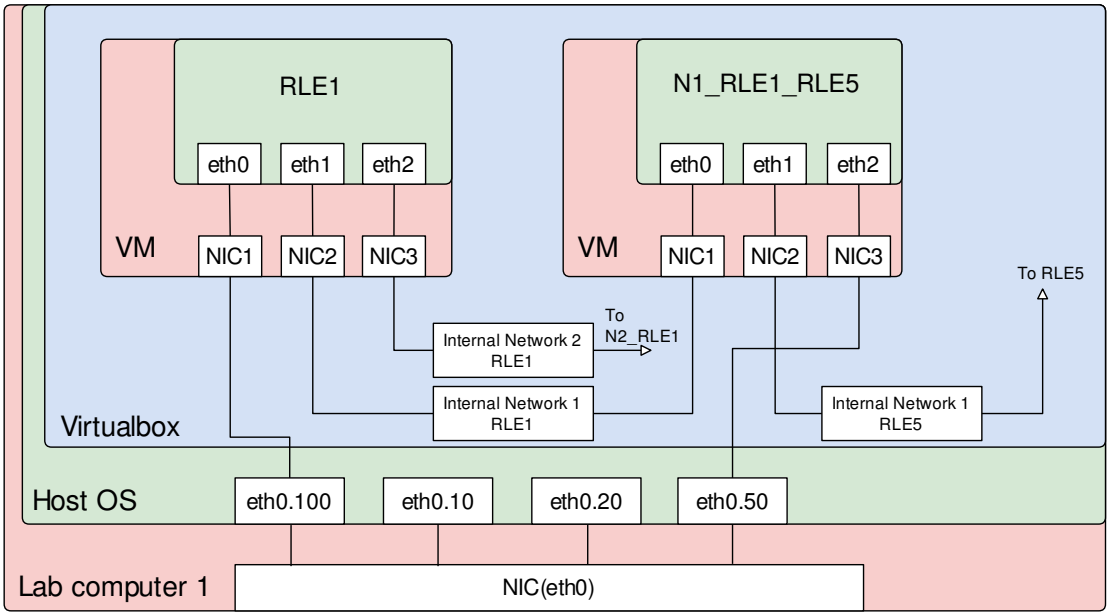


Figure 3.6: *A layered view that shows a small part of the network and how it is running inside VirtualBox.*

RLE1 has 3 network interfaces, the first of which is the admin interface (eth0) that is connected to the host OS interface eth0.100. The eth0 interface on RLE1 is not part of the RLE bridge, so there can never be a connection between eth0 and one of either eth1 or eth2. However, eth1 and eth2 are part of the RLE bridge which means that all traffic that goes through eth1 and eth2 can be modified in terms of packet loss and longer delay times. It also means that the traffic can be redirected to other interfaces that are part of the RLE bridge.

In the settings for the virtual machine, which RLE1 is running in, NIC2 is designated to be connected to "Internal Network 1 RLE1" which is the name of one of the internal networks. In the same way, NIC1 on N1_RLE3_RLE5 is designated to be connected to the same internal network as described in Figure 3.6. This simply corresponds to connect two physical computers with an Ethernet cable. In the same way, NIC3 on RLE1 is connected to internal network 2 RLE1 which goes on to NIC1 on N2_RLE1 (N2_RLE1 is not included in the figure).

As RLE1, every other RLE on lab computer 1 has its NIC1 connected to eth0.100. This is the admin network for the RLE machines where commands to the RLEs will be sent during simulation. The connection in VirtualBox between NIC1 and eth0.100 is a bridge connection. In the same way, the routing nodes have a bridged connection to the host OS eth0.50 interface. This is the admin network for the routing nodes. It is used in order to change the configuration between different routing protocols, configure the nodes before a simulation and download network data after a simulation is run.

If one now look back to Figure 3.5, one can understand how VirtualBox internal network helps building up the network environment. Every line in figure 3.5 is an internal network in VirtualBox. All internal networks connected to RLE5 will have a names like "Internal Network X RLE5" where x goes from 1 to 5, hence there are five internal networks connected to RLE5.

There is one radio network which is a bit different in the configuration and that is the one connecting the four companies together. The four nodes that are coloured dark grey in Figure 3.3. One different thing about this radio network is that it exists on all three physical computers. In lab computer 1 and 2, there are two routing nodes in each computer that are connected to the RLE that runs in lab computer 3. For example, in Figure 3.5, there is the routing node N1_RLE5_RLE21 which is connected to RLE5 and RLE21. RLE21 is running in lab computer 3 so the traffic has to go from lab computer 1 to 3. Because there are three more connections just like this the traffic must be separated in a logical way since all traffic is sent in the same cable. This is why VLAN is used for all traffic that is sent between the physical computers. For example, to make it possible to know which traffic that is sent from node N1_RLE5_RLE21, one of its interfaces has a bridged connection to the host OS interface eth0.10. This interface is also illustrated in Figure 3.6. So all traffic sent from this node to RLE21 are marked as VLAN10 traffic. In the same way, one interface on lab computer 3 is configured for VLAN10 which will receive the traffic. This interface is added to the RLE bridge so that the traffic can be changed according to the simulation. The fact that the RLE is running directly in the host OS on lab computer 3 makes it easier to add the incoming interface from another physical computer to the RLE bridge.

Figure 3.7 describes the physical network set up with the three lab computers. It also describes all VLAN interfaces and bridges that are configured on respective computer. Note that these are just the configurations in the host OS and not in Virtualbox. One important part of this interconnection is that lab computer 3 has two physical network cards which make this set up possible.

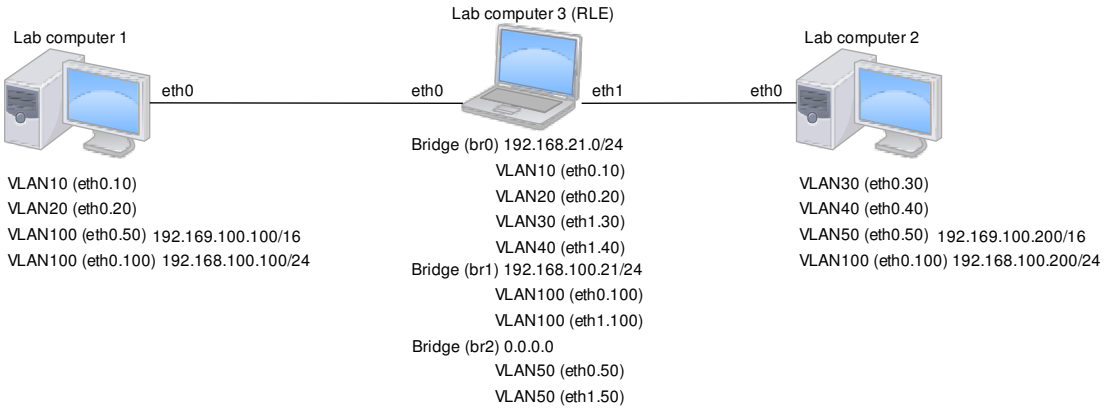


Figure 3.7: *The three physical computers and how they are connected with the use of VLAN.*

As previously said it is very important to be able to separate the data traffic with VLAN in order know what to do whit the traffic. The two admin networks, VLAN50 for the routing nodes and VLAN100 for the RLE machines are running over all three physical computers. This traffic must not be affected by the RLE that is running on

lab computer 3, even though all traffic is going through lab computer 3. That is why there are three different bridges configured on lab computer 3. Bridge br0 is the RLE bridge which means the RLE controls the traffic between all included interfaces. Br1 and Br2 is where the admin network interfaces are included. Therefore, the traffic is clearly separated and to illustrate this even further Figure 3.8 describes the traffic flow and how it is separated on each physical computer.

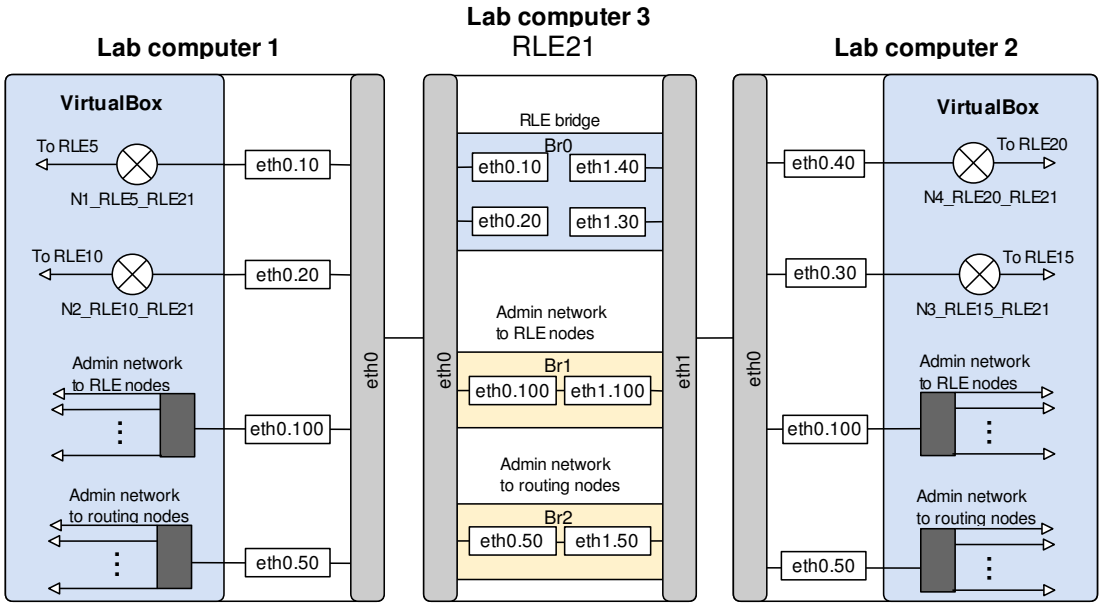


Figure 3.8: *A logical view of how the traffic is separated in the three lab computers.*

In Figure 3.8 one can see the four routing nodes that are connected through a common radio network. All of these nodes are connected, through VirtualBox, to its own VLAN in order to separate the traffic on the physical Ethernet cable between the computers. When the traffic reaches lab computer 3 it is divided according to the VLAN tag. For example, if there is a connection between eth0.10 and eth1.30 in the RLE on lab computer 3, then N1_RLE5_RLE21 and N3_RLE15_RLE21 are able to communicate with each other. The configurations for Br1 and Br2 on lab computer 3 are always static which means that the connectivity on the admin networks should be consistent. The simulations do not affect the traffic on these two networks. Because of the admin networks, it is possible to access all routing nodes and all RLE machines from the host OS from any of the lab computers. Furthermore, because of all nodes and all RLE machines has a SSH server installed, this makes it easy to write scripts that can do everything that is needed for the simulation. The source code for the scripts is not included in this report but the network environment can be totally automated in terms of loading different routing protocols on all nodes, change the configurations for these protocols on all nodes, set up the nodes for a routing simulation scenario, collect network data on preselected nodes and perform a simulation scenario by sending various commands to all RLE machines. This makes it easy to test different scenarios with different routing protocols and configurations.

3.3 Test definition

The network set up of 36 routing nodes makes it possible to perform a versatile test with different scenarios. The configured routing protocol has to maintain connectivity and find new routes when the RLE simulates that a link goes down. Moreover, the data traffic that is generated by the routing protocol in order to communicate with the other nodes is referred to as overhead traffic. Besides this overhead traffic there is regular ICMP data traffic that is generated with a ping program in order to measure current connection status.

Figure 3.9 shows a logical connection diagram over all routing nodes in the network. The edges between two nodes represent the connection over the radio network and are therefore controlled by the RLEs. Furthermore, during the test, the packer loss varies on some of the links in order to simulate disruptions.

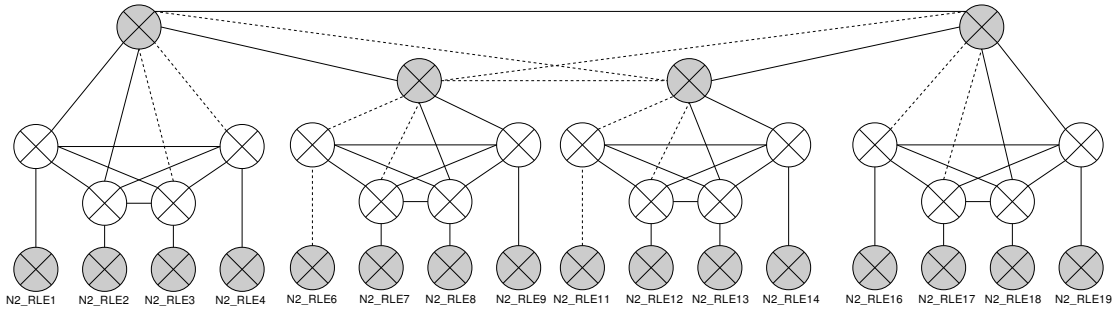


Figure 3.9: *A logical view over the routing nodes where the edges are controlled by the RLEs. The dotted edges have a packet loss of 100 percent at some times during the test. Network traffic is logged at all gray nodes.*

Some of the edges in Figure 3.9 are dotted lines and some are plain. The plain edges have a fixed packet loss value during a complete test case. However, the dotted edges have a packet loss that varies between 0 and 100 percent. Therefore, when these edges have 100 percent packet loss, the connectivity is completely lost which force the protocol to find a new route.

Network traffic is logged by all gray nodes in Figure 3.9. The four gray nodes that are in the middle of the network is referred to as intermediate nodes while the nodes that only have one possible connection is referred to as end nodes. The name of the end nodes are stated in Figure 3.9. Moreover, it is between the end nodes that the ICMP traffic is sent during a test in order to utilize the entire network. To capture different scenarios but still be able to get rid of deviation, Table 3.1 describes how the nodes are paired with source and destination.

Nr	Source	Destination	Shortest path, full connectivity	Shortest path, edges are down	Isolated
1	N2_RLE1	N2_RLE19	5	5	No
2	N2_RLE2	N2_RLE18	5	5	No
3	N2_RLE3	N2_RLE17	5	7	No
4	N2_RLE4	N2_RLE16	5	7	No
5	N2_RLE6	N2_RLE14	5	N/A	Yes
6	N2_RLE7	N2_RLE13	5	8	No
7	N2_RLE8	N2_RLE12	5	8	No
8	N2_RLE9	N2_RLE11	5	N/A	Yes

Table 3.1: *How the end nodes send ICMP messages to each other and possible shortest path between the nodes.*

The first column in Table 3.1 describes the source nodes which sends ICMP traffic to the destination node stated in column two. When the dotted edges in Figure 3.9 are available for communication the shortest number of hops between source and destination is stated in column three. This can be verified from Figure 3.9. Furthermore, when the dotted edges are set to 100 percent packet loss the shortest path might change which is stated in column four. Column five clarifies if the node gets isolated from the rest of the network which means at some point during the simulation the node does not have any neighbours.

Note that the first and second source and destination pair in Table 3.1 always have 5 hops as the shortest path. This is because the dotted lines in Figure 3.9 do not affect the shortest path for these nodes. Furthermore, these two pairs are expected to deliver the same result which is a safety to avoid erroneous deviations during the simulation. In the same way, the other source and destination nodes have been selected in a way that they have one other source and destination pair to compare it to.

There are two nodes which is slightly different from the other nodes. Node N2_RLE6 and N2_RLE11 gets disconnected from the entire network when the dotted edges in Figure 3.9 reach 100 percent packet loss. Therefore, no ICMP traffic is able to be sent or received to these nodes during the connectivity loss.

The plain edges in Figure 3.9 are static during a test case. However, they are different test cases where these edges have different values. One objective is to measure how much overhead traffic that is generated from the different protocols and how that changes with higher packet loss on the links. Therefore, the packet loss for the plain edges is varied from 0 to 25 percent as seen in Table 3.2.

Test	Plain edge	Dotted edge
1	0%	Varies between 0-100%
2	2.5%	Varies between 2.5-100%
3	5%	Varies between 5-100%
4	10%	Varies between 10-100%
5	25%	Varies between 25-100%

Table 3.2: *Describes the packet loss on the edges for the five test cases.*

Table 3.2 describes the five different test cases that are run for each protocol

in order to measure the amount of overhead traffic that is generated. Ping ratio and convergence time is only measured in the first test case where the plain edges have 0 percent packet loss. This is because the ICMP messages that are used to measure these parameters get dropped which complicates the measurement. The convergence time is defined as the time it takes to register and manage an interrupt until the protocol has established a new route. The ping ratio is defined as the ratio between sent and received ping packets on a preselected node.

Each protocol has different parameters that can be modified to make the protocol behave in a different way. When these options are varied the protocol is only tested with test case 1. Test case 1 is designed to only focus on 100 percent disruptions on preselected links. This makes it possible to measure how fast the different protocols change routes and how they behave in different situations.

Chapter 4

Results and Discussion

4.1 Results with default configurations

The protocols are initially tested with their default configuration. In order to compare the protocols against each other, different settings are used for each protocol. Figure 4.1 shows the ping ratio for test case 1 with the protocols configured with their default settings. The x-axis describes the eight nodes that sends ping request according to the test case described in the previous chapter. The y-axis describes the ratio between sent and received ping messages.

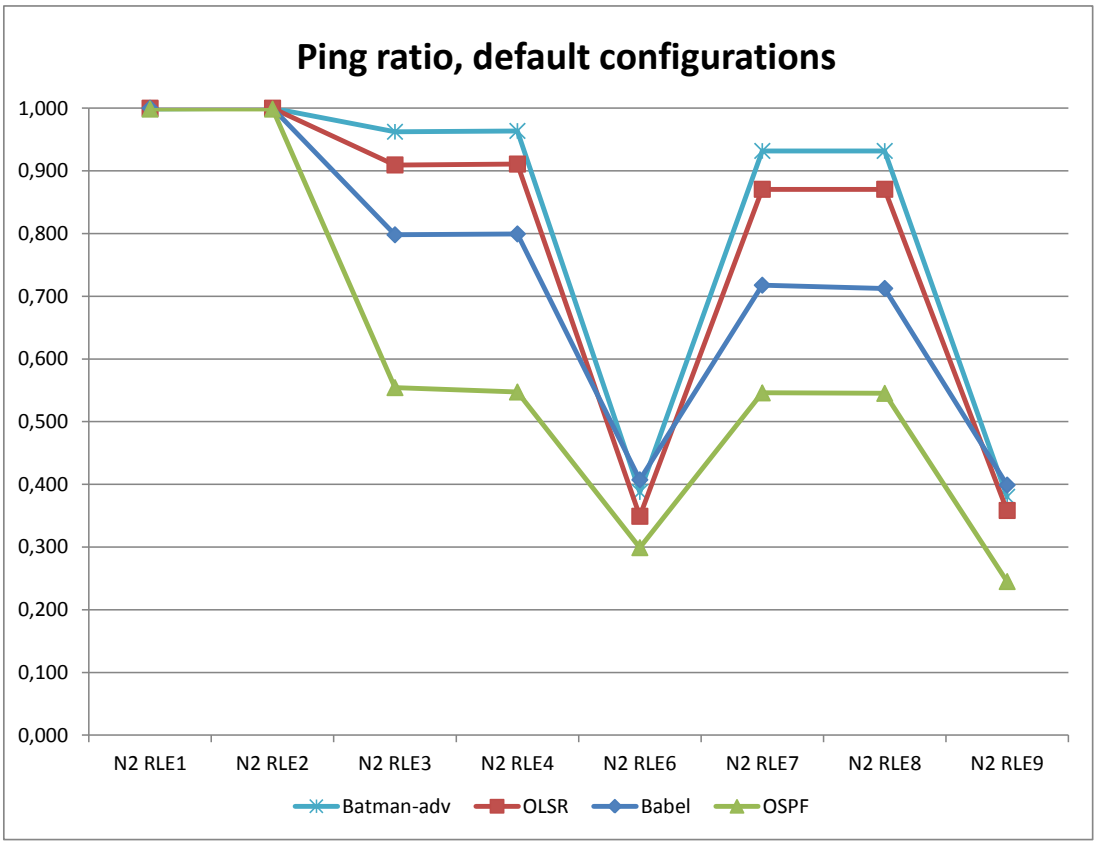


Figure 4.1: *The ping ratio for all protocols with their default configurations.*

In this figure, batman-adv has the best performance because of the highest ratio on most of the nodes. With the default setting, Batman-adv has a convergence time

around 10 seconds for this test case. That is by far the fastest of the four protocols. The convergence times for node N2RLE3 are listed in Table 4.1.

Protocol with default settings	Convergence time on N2RLE3
Batman-adv	10s
OLSR	20.5s
Babel	18.5s
OSPF	38s

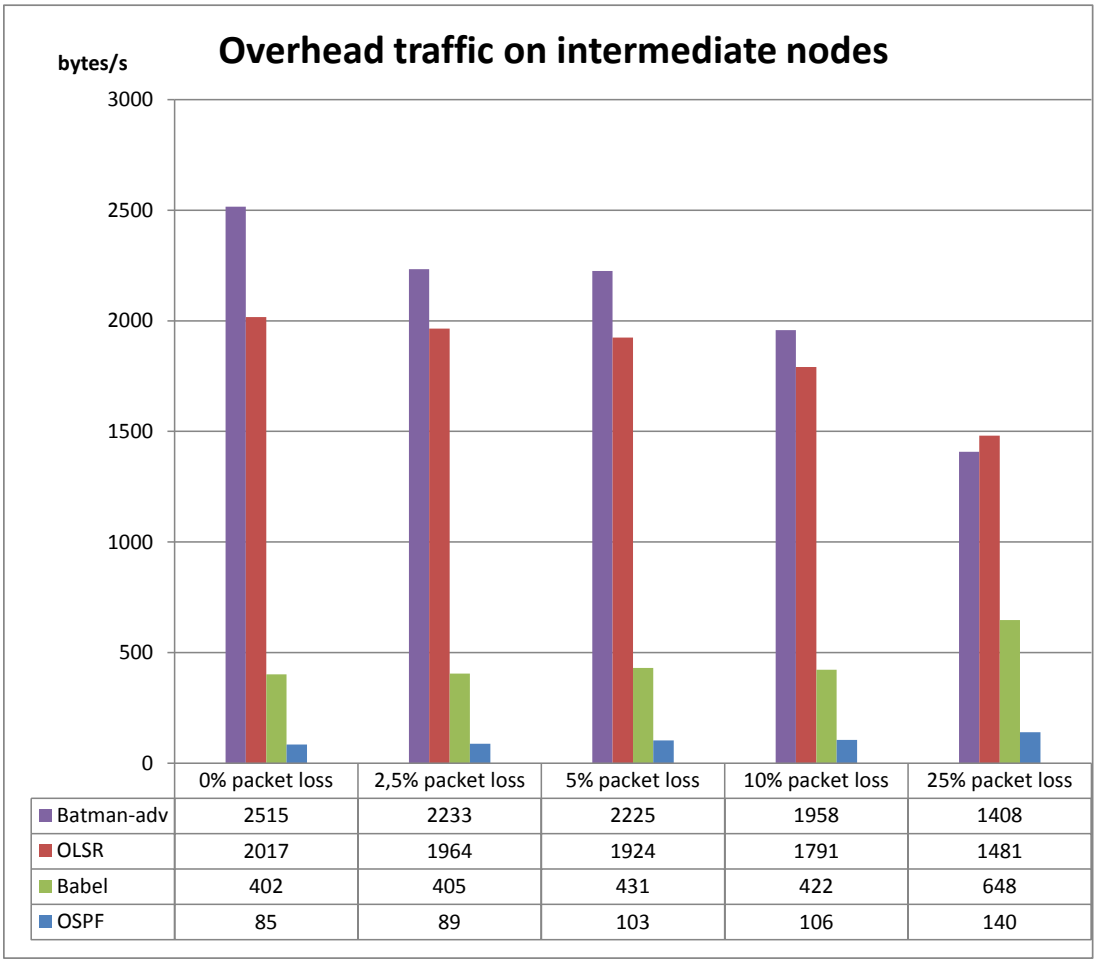
Table 4.1: *Convergence times measured at the node N2RLE3 with default configurations for all protocols.*

Note that OLSR has a higher convergence time than Babel but still gets a better ping ratio. This is because OLSR is slower to change back to the shortest path when this path is available.

For all nodes in the figure, except for N2RLE6 and N2RLE9, there is always a path to the host that they are sending ping messages to. At first, all protocols will find the shortest path. However, after some time the shortest path are affected by disruption forcing the protocol to find a new route. The test waits for 60 seconds in order for the protocol to converge which according to Table 4.1 is enough. After the 60 seconds the shortest path are available again, and there is no other disruption in the network. The test waits for 30 seconds for the protocols to change back to the shortest path before it is taken down again. Babel finds the shortest path again and changes back to it. In contrast, OLSR is not changing to the shortest path, but is still using the same route. So when the 30 seconds has passed Babel are experiencing the same procedure again while OLSR is still on the same route and therefore does not notice any difference. Batman-adv has the same behaviour as OLSR and is not changing back to the shortest path during the 30 seconds that is available. OSPF does change back to the shortest path and for node N2RLE3 it takes about 20 seconds to find the old path again.

The other test cases, where the links are constantly suffering from packet loss, are more difficult to evaluate with a ping ratio diagram. This is because packets are continuously dropped randomly which make the statistical variance large from one test to the other.

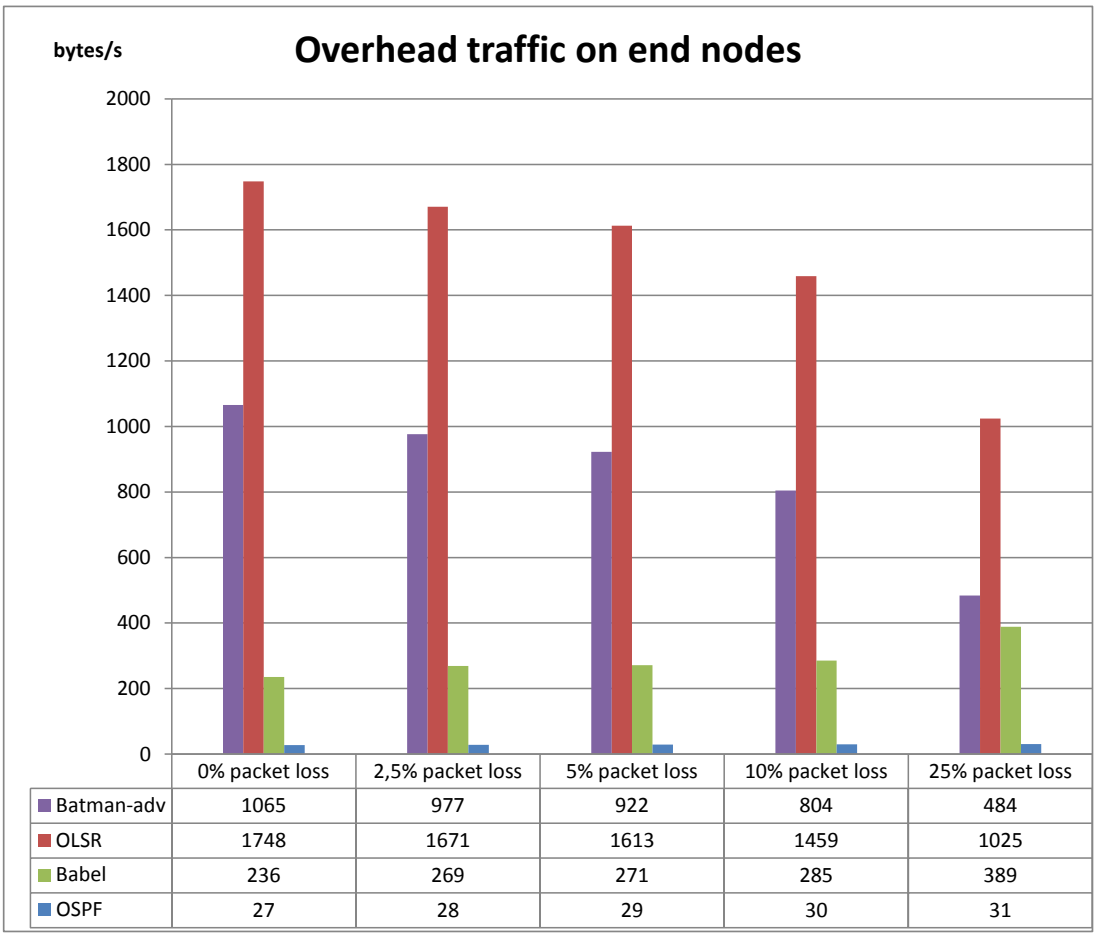
Figure 4.2 describes the outgoing overhead traffic for six different test cases with packet loss from 0 to 25 percent on every link. The diagram shows an average from the intermediate nodes in the network, that is all nodes connected to more than one node.



	0% packet loss	2,5% packet loss	5% packet loss	10% packet loss	25% packet loss
Batman-adv	2515	2233	2225	1958	1408
OLSR	2017	1964	1924	1791	1481
Babel	402	405	431	422	648
OSPF	85	89	103	106	140

Figure 4.2: *Overhead traffic on intermediate nodes when all links have packet loss from 0 to 25 percent. All protocols have their default configuration.*

Because all protocols have their default properties in the tests shown in Figure 4.2, the overhead traffic is significantly different. In comparison to Figure 4.1, one can see that the overhead traffic seems to correlate with the packet ratio for the different protocols. The protocols with a higher ping ratio have higher overhead traffic. One interesting point in Figure 4.2 is how the overhead traffic changes with higher packet loss on the links. With higher packet loss on the links running Batman-adv and OLSR the overhead traffic decreases with higher packet loss. This seems correct because none of the protocol use any type of acknowledge message to verify that a message has been received. Furthermore, when less information is received by a node, less information is also sent out from a node. However, running the same scenarios for OSPF and Babel the opposite happens. The nodes generate more overhead traffic when the packet loss on the links is higher. This can be explained with acknowledge messages that both OSPF and Babel use. If a packet that should be acknowledged is dropped by a RLE, this triggers a retransmit of the packet after a specific timeout value. Therefore, with increasing packet loss on the links, OSPF and Babel generates an increasing output of overhead traffic from the nodes. This behaviour with increased overhead traffic when the links experience higher packet loss might be positive because the probability of a connection increases. However, if too much overhead traffic is generated, it consumes both more resources and might lead to congestion.



	0% packet loss	2,5% packet loss	5% packet loss	10% packet loss	25% packet loss
Batman-adv	1065	977	922	804	484
OLSR	1748	1671	1613	1459	1025
Babel	236	269	271	285	389
OSPF	27	28	29	30	31

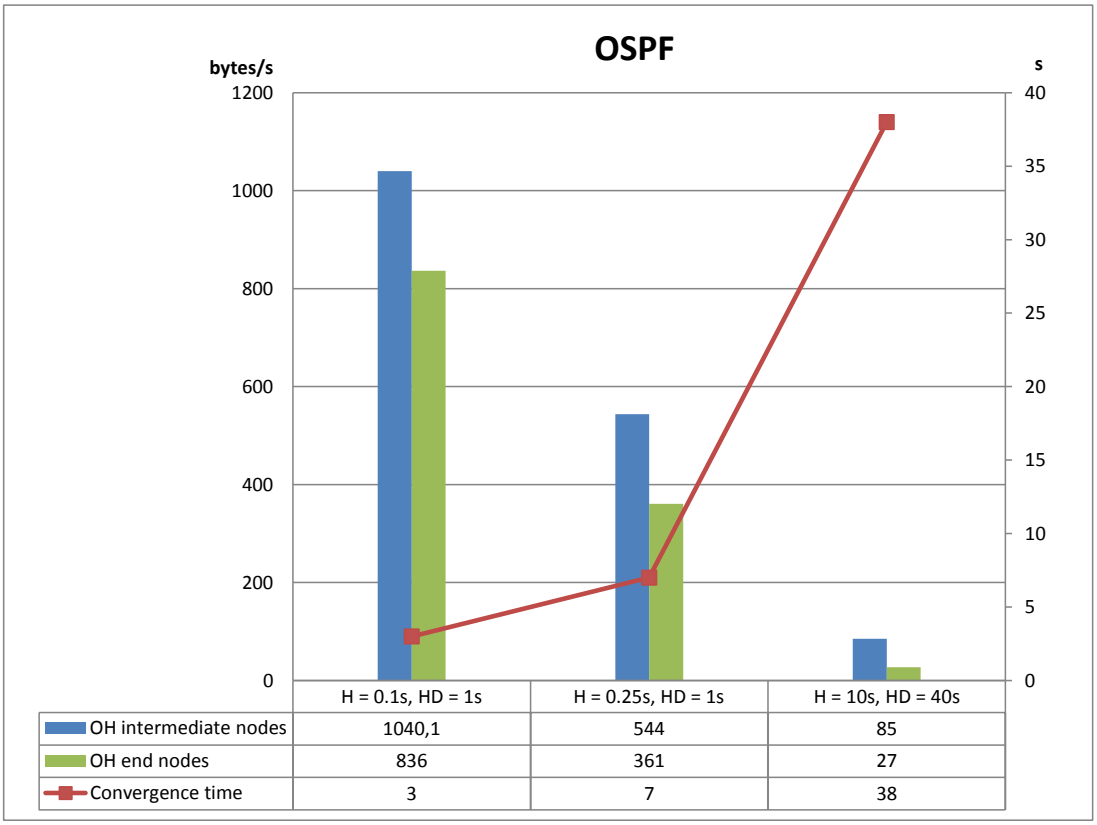
Figure 4.3: *Overhead traffic on end nodes when all links have packet loss from 0 to 25 percent. All protocols have their default configuration.*

The outgoing overhead traffic from the end nodes are shown in Figure 4.3. The end nodes are nodes that only have a connection to one other node. The overall observation is that all end nodes generate less overhead traffic than the intermediate nodes; this is true for all four protocols. Moreover, this is because the intermediate nodes communicate with several neighbours while the end nodes only have one neighbour to send data to. Both Figure 4.2 and 4.3 express the overhead traffic in bytes per second. However, Appendix B also includes the same test results expressed in packets per second.

The difference between Figure 4.2 and 4.3 when comparing the protocols individually is that OLSR has not decreased as much in comparison with the other protocols. However, it is important to remember that there are a lot of configuration possibilities for all of the protocols. All protocols are tested with different configurations in order to see how overhead traffic, ping ratio and convergence time are affected. Some of the protocols have a lot of configuration opportunities, for example OLSR. However, only a few of the available parameters are tested, the rest of them are left as default values. Subsection 4.3 describes results from OLSR with different configurations. It shows that it is possible to configure OLSR in a way that reduce the amount of overhead traffic but almost keep the convergence time. Therefore, it is clearly possible to optimize the configuration for OLSR.

4.2 OSPF

For OSPF, the hello-dead and hello interval are varied. The hello interval specifies how often hello messages are sent out from a routing node. The hello-dead interval specifies how long time the routing node should wait to receive a hello message before declaring the link as lost. The convergence time and overhead traffic for the intermediate and end nodes are illustrated in Figure 4.4.



	H = 0.1s, HD = 1s	H = 0.25s, HD = 1s	H = 10s, HD = 40s
OH intermediate nodes	1040,1	544	85
OH end nodes	836	361	27
Convergence time	3	7	38

Figure 4.4: *The relation between overhead traffic and convergence time for different settings for OSPF.*

The default settings are hello (H) to 10 seconds and hello-dead (HD) to 40 seconds. From Figure 4.4 one can see that these settings had a poor convergence time of 38 seconds and generated overhead traffic of 27-85 bytes/s on respective nodes. The lowest hello and hello-dead interval that is possible to set in the Quagga version of OSPF is hello interval to 0.1 second and hello-dead interval to 1 second (note that according to the specification it should be possible to select a lower hello interval but this generated an error message from OSPFd when it was done in this test).

A convergence time of 3 seconds in combination with overhead traffic between 836-1040 is a good result compared to how the other protocols performed in this test. In Figure 4.5 the ping ratio for OSPF is displayed for the three different settings. As expected, the settings which result in the lowest convergence time have the best ping ratio for all nodes.

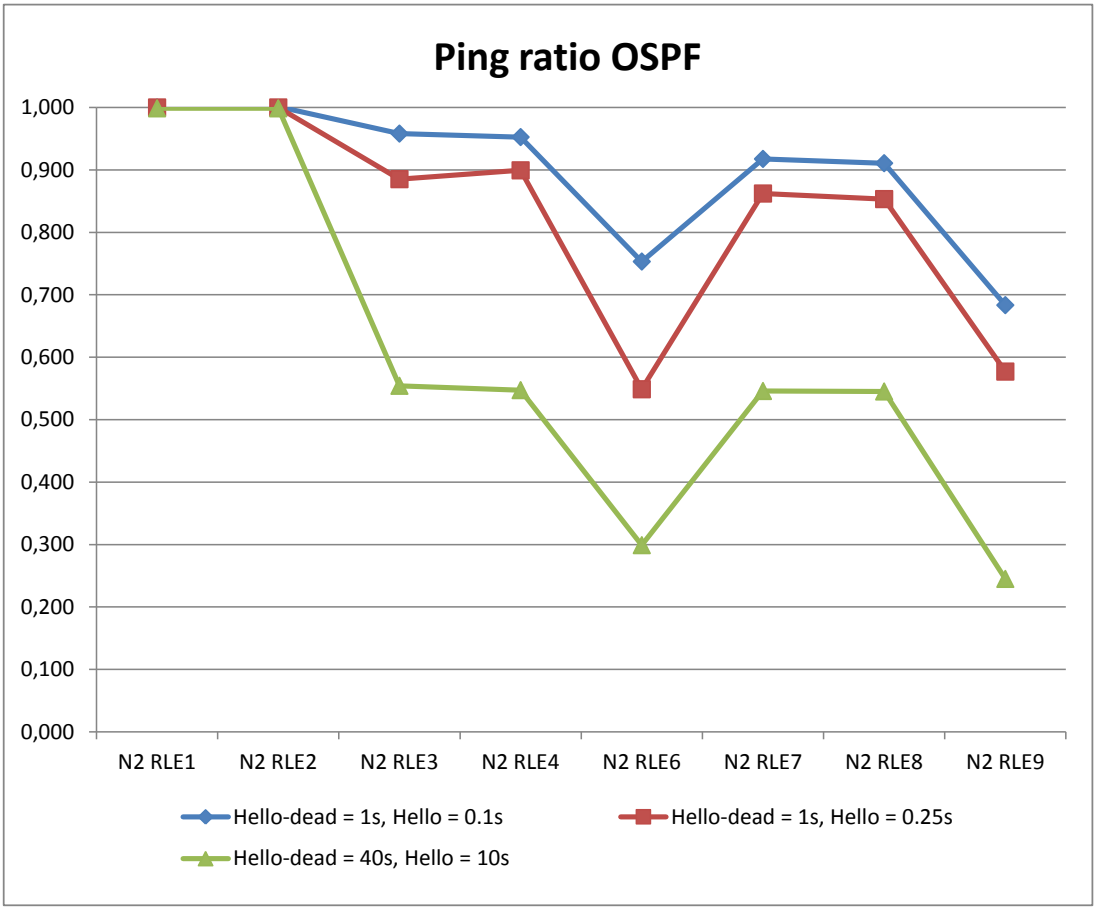


Figure 4.5: *The ping ratio for different settings for OSPF.*

4.3 OLSR

Because OLSR has a large number of possible configuration options only a few of these where varied. In Figure 4.6 one can see how the overhead traffic and the convergence time relate to each other when the topology control (TC) and hello (H) intervals are varied. The default settings are TC to 0.5 seconds and H to 6 seconds. These settings generated an overhead traffic of 1748-2017 bytes/s and a convergence time of 20.5 seconds which is poor compared to the other protocols. However, changing the TC to 1 second significantly decreases the overhead traffic but the convergence traffic remains almost the same. Furthermore, the reason why the TC interval is much lower than the hello interval in all tests is to avoid the possibility to create loops. This is especially important when the fish eye setting is used because that minimizes the amount of TC messages that are sent.

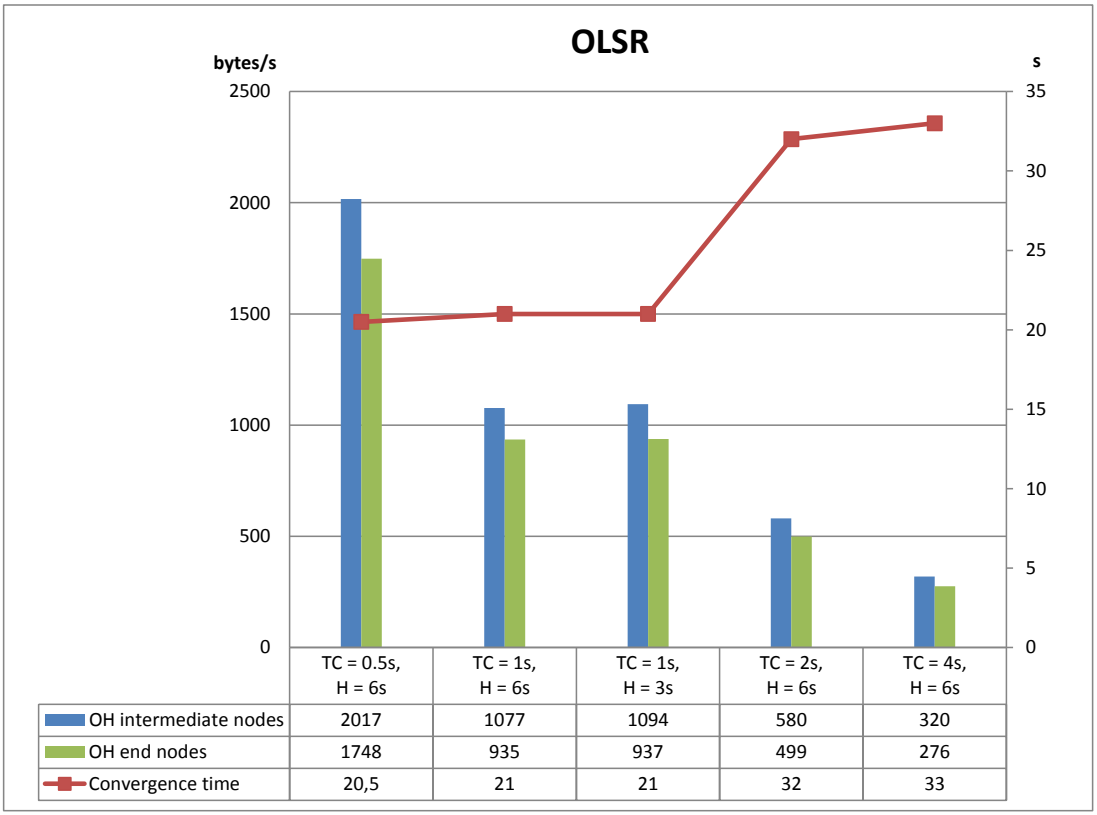


Figure 4.6: *The relation between overhead traffic and convergence time for different settings for OLSR.*

As one can see in Figure 4.6, the convergence time for OLSR is not as low as the convergence time for the other protocols. It might however be possible to get a lower convergence time if one optimize more of the available settings. However, it does not seem likely that the overhead traffic should get any significant improvement.

The ping ratio for OLSR is displayed in Figure 4.7. The results does not differ much between the five configurations. There are two reasons for this, one is that the convergence times have a relatively small variability between the different configurations. The second, and most important reason, is that the OLSR protocol did not change back to the shortest path in several cases as described earlier. Furthermore, this causes the small difference in ping ratio because it is only in the beginning of the test that the protocol changes to a new route.

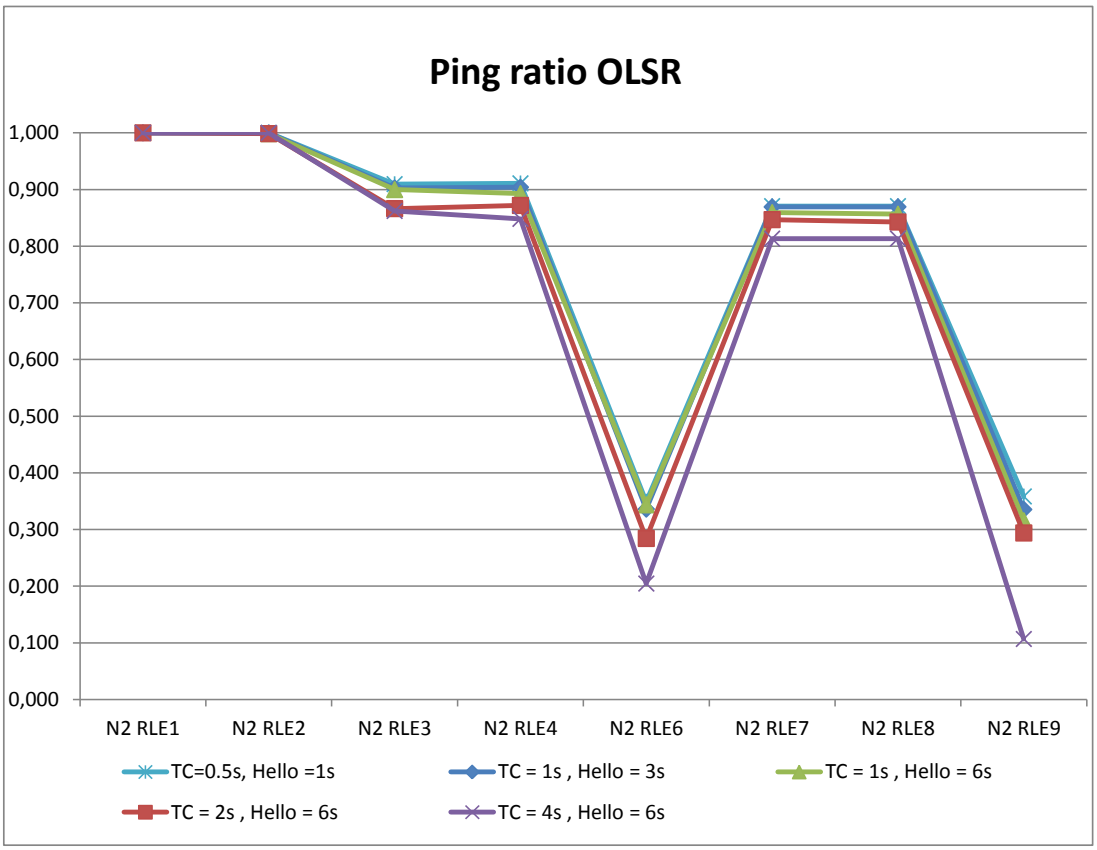
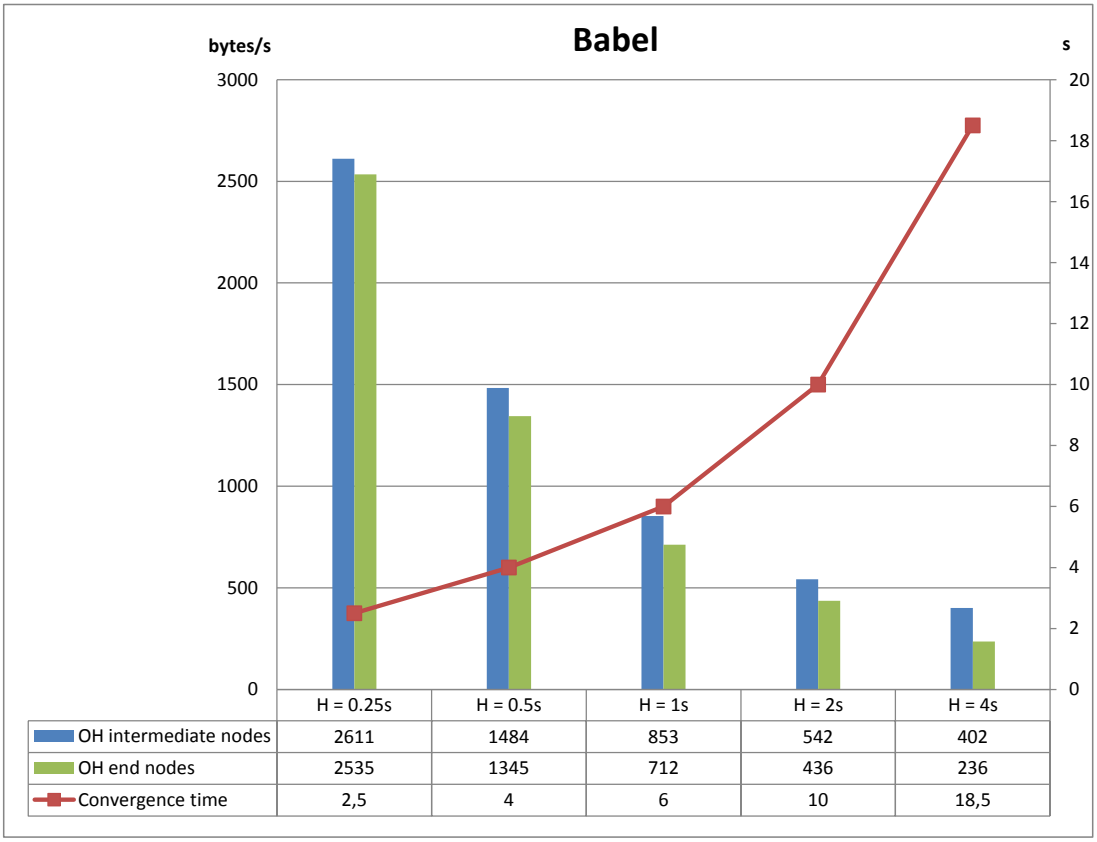


Figure 4.7: *The ping ratio for different settings for OLSR.*

In order to get a better variability of the ping ratio, the test should be sure to always affect the selected route. This is not the case for the OLSR protocol which makes Figure 4.7 limited useful.

4.4 Babel

Babel makes use of triggered update messages. Therefore, the only parameter that is varied is the hello interval. The convergence time and overhead traffic is shown in Figure 4.8 where the hello interval varies between 0.25 seconds up to the default value of 4 seconds. Overall, the Babel protocol shows a relatively good performance with respect to convergence time and generated overhead traffic. For example, a hello interval of 1 second generate the overhead traffic of 712-853 bytes/s which is a lot better than OLSR comparing just these two parameters. However, when the convergence time goes under 6 seconds the overhead traffic increases rapidly. When changing the default hello interval value of 4 seconds to 2 seconds one gains a lot in convergence time but does not lose so much in overhead traffic. The opposite applies when the hello interval goes under 1.



	H = 0.25s	H = 0.5s	H = 1s	H = 2s	H = 4s
OH intermediate nodes	2611	1484	853	542	402
OH end nodes	2535	1345	712	436	236
Convergence time	2,5	4	6	10	18,5

Figure 4.8: *The relation between overhead traffic and convergence time for different settings for Babel.*

Unlike OLSR, the Babel protocol did change back to the shortest path when it became available. Therefore, the ping ratio diagram shown in Figure 4.9 is more useful than corresponding diagram for OLSR. Note that the default configuration, with hello interval of 4 seconds, is the worst case with respect to ping ratio. All configurations with lower hello interval clearly improve the ping ratio.

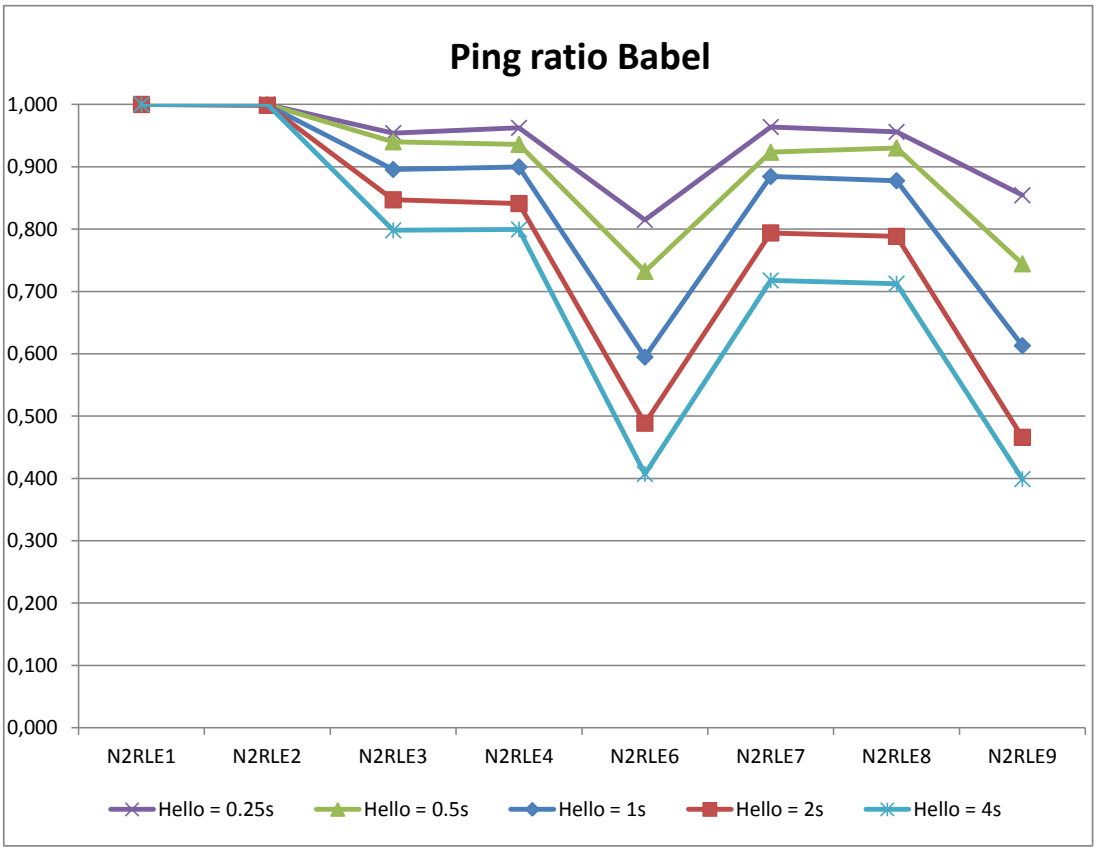


Figure 4.9: *The ping ratio for different settings for Babel.*

4.5 Batman-adv

Like Babel, Batman-adv is also tested with just adjusting one parameter which is the originator interval. The originator interval decides the time interval for how often OGM messages are sent out. The default value for this parameter is 1 second. Figure 4.10 displays the results when the originator interval is adjusted from 0.5 to 4 seconds. Compared to the other protocols, especially compared to Babel and OSPF, Batman-adv generated a lot of overhead traffic to deliver the equivalent convergence time.

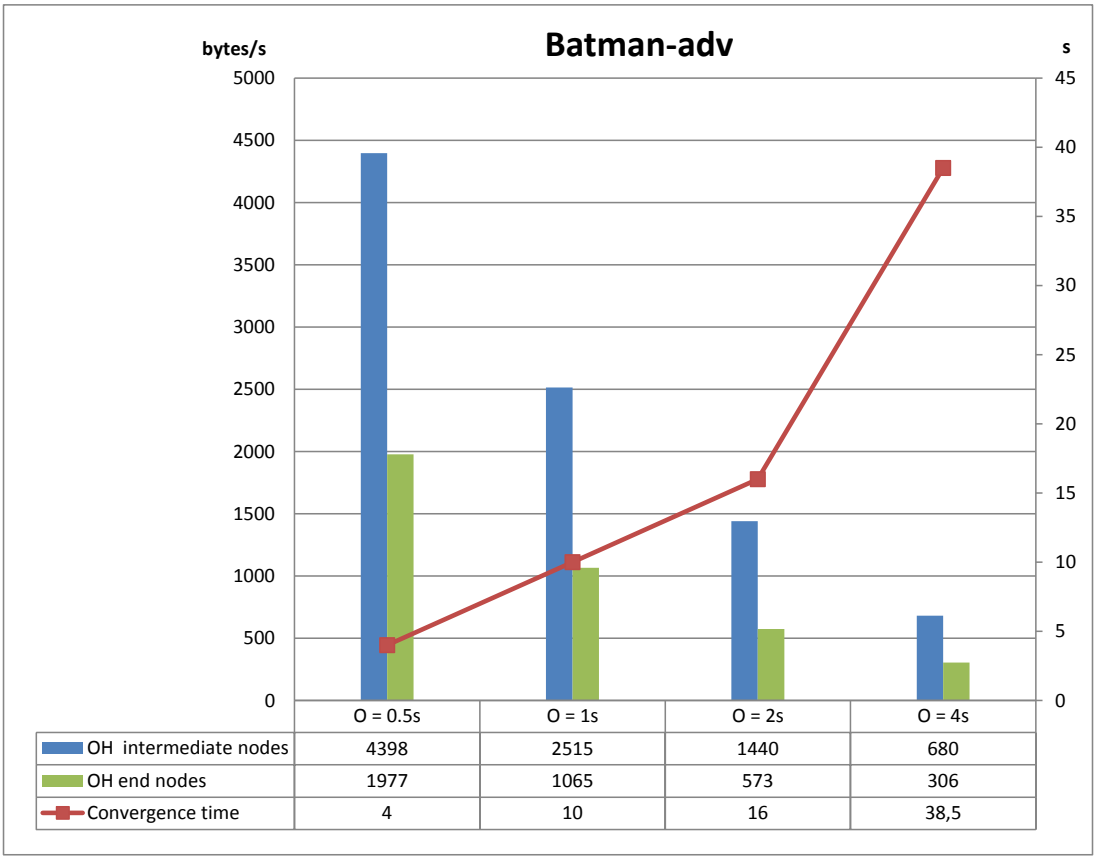


Figure 4.10: *The relation between overhead traffic and convergence time for different settings for Batman-adv*

Batman-adv has the same behaviour as OLSR when the shortest path gets available after an interrupt. It does not change back to the shortest path for a relatively long time. When the originator value is increased, this time seems to increase as well. Therefore, the diagram in Figure 4.11 is not strictly related to the convergence time as one might expect.

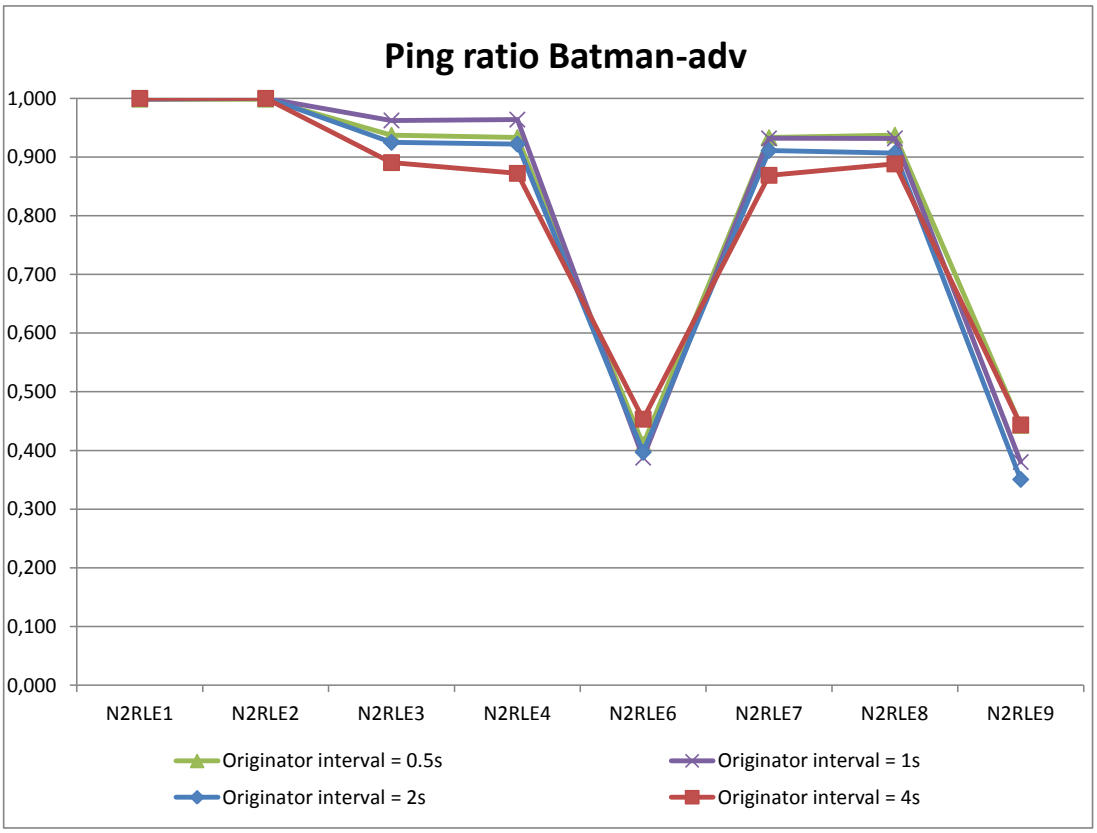


Figure 4.11: *The ping ratio for different settings for Batman-adv.*

One interesting observation is to compare the ping ratio of node N2RLE6 and N2RLE9 for the different protocols. These two nodes are completely disconnected from the network during some times in the test. All other nodes always have some neighbour that it is connected to. In comparison with the other protocols, batman-adv and OLSR have the worst performance on these two nodes. The reason why Batman-adv get so low ping ratio on these two nodes seems to be because it is slow to detect interruption. The nodes are still sending out ping packets for some time even though the connection is lost. However, for the nodes to be able to route ping data, they are dependent on the kernel routing table. When batman-adv detects that a route is no longer available, it should immediately modify the kernel routing table. If this is not the case, the ping program continues to send out ICMP messages.

Chapter 5

Conclusion

It is important to emphasize that by changing settings for all the tested routing protocols, it is possible to achieve better results than the default configuration could offer. However, there is always a trade off between obtaining a low convergence time or limit the amount of generated overhead traffic. Furthermore, it should be possible to optimize these values for the intended use.

With a tactical network set up as the one tested, OSPF is an alternative to Babel, Batman-adv and OLSR. When configuring OSPF to achieve the lowest convergence time, the overhead traffic that is generated is still better than the other protocols. However, with the Quagga implementation of OSPF, it is not possible to configure OSPF to achieve a lower convergence time.

Abolhasen et al. [6] states that both Batman and Babel performs better than OLSR. However, the test shows that it is possible to reduce the overhead traffic and almost maintain the convergence time for OLSR compared to the default configuration. Because there are much more configuration options available for OLSR, it might be possible to optimize the configuration to better adapt to a tactical network set up. Moreover, the result does not strictly support the results from [29] and [5] that claim that OLSR generates more overhead traffic than Batman-adv. However, none of [29] and [5] used a tactical network set up or varied the settings for the protocols. With respect to the trade off between convergence time and overhead traffic Babel is the second best protocol after OSPF. Batman-adv generates more overhead traffic to achieve the same convergence time. Moreover, even though Batman-adv takes advantage of routing on layer 2, both Babel and OSPF shows better performance with lower routing overhead for corresponding convergence time.

Further work could include optimizing the protocol settings to adapt to the limitations of what the selected implementation requires.

Bibliography

[1] *Ad-hoc Networks: Fundamental Properties and Network Topologies*. Springer Netherlands, 2006.

[2] J. Loo, S. Khan, and A. N. Al-Khwildi", "Mobile ad hoc routing protocols," in *Mobile Ad Hoc Networks: Current Status and Future Trends*, J. L. M. Jonathan Loo and J. H. Ortiz, Eds. CRC Press, 2011, pp. 3–18. [Online]. Available: http://www.crcnetbase.com/isbn/9781439856512

[3] tcpdump.org, "Tcpdump," http://www.tcpdump.org/, 2013, available 2013-09-22.

[4] jnetpcap.com, "jnetpcap," http://jnetpcap.com/, 2013, available 2013-09-22.

[5] D. Johnson, N. Ntlatlapa, and C. Aichele, "Simple pragmatic approach to mesh routing using batman," 2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries, October 2008, pretoria, South Africa.

[6] M. Abolhasan, B. Hagelstein, and J. Wang, "Real-world performance of current proactive multi-hop mesh protocols," in *Communications, 2009. APCC 2009. 15th Asia-Pacific Conference on*, 2009, pp. 44–47.

[7] T. Plesse, C. Adjih, and P. Minet", "Olsr performance measurement in a military mobile ad hoc network," in *Ad Hoc Networks 3*. Elsevier, September 2004, pp. 575–588.

[8] M. Marina and S. Das, "Routing in mobile ad hoc networks," in *Ad Hoc Networks*, P. Mohapatra and S. Krishnamurthy, Eds. Springer US, 2005, pp. 63–90. [Online]. Available: http://dx.doi.org/10.1007/0-387-22690-7_3

[9] R. Bellman, "On a routing problem," 1958.

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," 1959.

[11] J. Moy, "Ospf version 2," http://www.ietf.org/rfc/rfc2328.txt, April 1998, available: 2013-06-26.

[12] E. Baccelli, P. Jacquet, D. Nguyen, and T. Clausen, "Ospf multipoint relay (mpr) extension for ad hoc networks," http://tools.ietf.org/html/rfc5449, February 2009, available: 2013-06-26.

[13] A. Roy and M. Chandra, "Extensions to ospf to support mobile ad hoc networking," http://tools.ietf.org/html/rfc5820, March 2010, available: 2013-06-26.

[14] R. Ogier and P. Spagnolo, "Mobile ad hoc network (manet) extension of ospf," http://tools.ietf.org/html/rfc5614, August 2009, available: 2013-06-26.

[15] P. B. Charles E. Perkins, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers." [Online]. Available: http://dx.doi.org/10.1109/90.554729

[16] G. Malkin and B. Networks, "Rip version 2," http://tools.ietf.org/html/rfc2453, November 1998, available: 2013-06-26.

[17] G. He, "Destination-sequenced distance vector (dsdv) protocol," Networking Laboratory, Helsinki University of Technology, Tech. Rep.

[18] T. Clausen, P. Jacquet, and S. Das, "Optimized link state routing protocol (olsr)," http://www.ietf.org/rfc/rfc3626.txt, October 2003, available: 2013-06-26.

[19] olsr.org, "olsrd an adhoc wireless mesh routing deamon," http://www.olsr.org, 2013, available 2013-06-10.

[20] A. Neumann, C. Aichele, and M. Lindner, "Better approach to mobile ad-hoc networking (b.a.t.m.a.n.)," http://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00, April 2008, available: 2013-09-16.

[21] J. Chroboczek, "The babel routing protocol," http://tools.ietf.org/html/rfc6126, April 2007, available: 2013-09-16.

[22] J. J. Garcia-Luna-Aceves and S. Murthy, "A path-finding algorithm for loop-free routing," *IEEE/ACM Trans. Netw.*, vol. 5, no. 1, pp. 148–160, Feb. 1997. [Online]. Available: http://dx.doi.org/10.1109/90.554729

[23] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," http://www.ietf.org/rfc/rfc3561.txt, July 2003, available: 2013-06-26.

[24] D. Johnson, Y. Hu, and D. Maltz, "The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4," http://www.ietf.org/rfc/rfc4728.txt, February 2007, available: 2013-06-26.

[25] virtualbox.org, "Virtualbox manual," https://www.virtualbox.org/manual/ch01.html, 2013, available 2013-08-10.

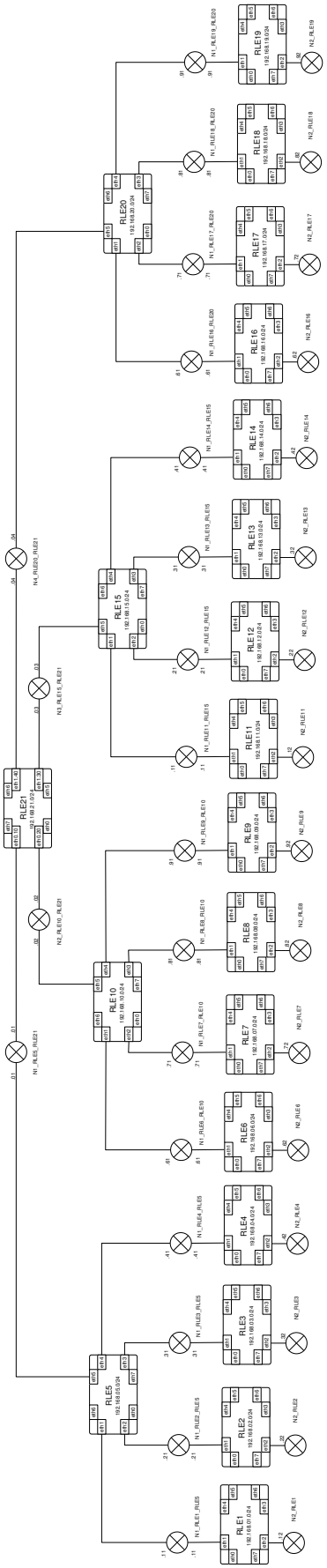
[26] M. A. Brown, *Traffic Control HOWTO*, 1st ed., October 2006, available 2013-06-10. [Online]. Available: http://ftp.eenet.ee/LDP/HOWTO/pdf/Traffic-Control-HOWTO.pdf

[27] open mesh.org, "B.a.t.m.a.n. protocol concept," http://www.open-mesh.org, 2013, available 2013-08-10.

[28] J. Whitbeck, Y. Lopez, J. Leguay, V. Conan, O. Rosenberg, and O. Tessier, "Using uhf connectivity to off-load vhf messaging in tactical manets," in *MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011*, 2011, pp. 961–966.

[29] D. Murray, M. Dixon, and T. Koziniec, "An experimental comparison of routing protocolsin multi hop ad hoc networks," 2010 Australasian Telecommunication Networks and Applications Conference.

Appendix A - Network topology of test environment



Appendix B - Overhead traffic for different configurations, relation between bytes and number of packets

OSPF

OSPF default settings, Hello-dead = 40s, Hello = 10s						
	Test case	0%	2.5%	5%	10%	25%
Intermediate nodes	packets/s	0.49	0.55	0.66	0.65	0.79
	bytes/s	85	89	103	106	140
End nodes	packets/s	0.27	0.29	0.30	0.30	0.32
	bytes/s	27	28	29	30	31

OSPF test case 1 (0%), Hello-dead = 1s, Hello = 0.25s		
	Intermediate nodes	End nodes
packets/s	5.1	4.2
bytes/s	544	361

OSPF test case 1 (0%), Hello-dead = 1s, Hello = 0.1s		
	Intermediate nodes	End nodes
packets/s	10.8	10.0
bytes/s	1040	836

OLSR

OLSR default settings, TC = 0.5s, Hello = 6s						
	Test case	0%	2.5%	5%	10%	25%
Intermediate nodes	packets/s	2.1	2.1	2.1	2.1	2.1
	bytes/s	2017	1964	1924	1791	1481
End nodes	packets/s	2.1	2.1	2.1	2.1	2.1
	bytes/s	1748	1671	1613	1459	1025

OLSR test case 1 (0%), TC = 1s, Hello = 6s		
	Intermediate nodes	End nodes
packets/s	1.1	1.1
bytes/s	1077	935

OLSR test case 1 (0%), TC = 1s, Hello = 3s		
	Intermediate nodes	End nodes
packets/s	1.1	1.1
bytes/s	1094	937

OLSR test case 1 (0%), TC = 2s, Hello = 6s		
	Intermediate nodes	End nodes
packets/s	0.6	0.6
bytes/s	580	499

OLSR test case 1 (0%), TC = 4s, Hello = 6s		
	Intermediate nodes	End nodes
packets/s	0.3	0.3
bytes/s	320	276

Babel

Babel default settings, Hello = 4s						
	Test case	0%	2.5%	5%	10%	25%
Intermediate nodes	packets/s	1.4	1.5	1.6	1.6	2.4
	bytes/s	402	405	431	422	648
End nodes	packets/s	0.6	0.7	0.7	0.7	1.2
	bytes/s	236	269	271	285	389

Babel test case 1 (0%), Hello = 2s		
	Intermediate nodes	End nodes
packets/s	1.7	1.1
bytes/s	542	436

Babel test case 1 (0%), Hello = 1s		
	Intermediate nodes	End nodes
packets/s	2.4	1.7
bytes/s	853	712

Babel test case 1 (0%), Hello = 0.5s		
	Intermediate nodes	End nodes
packets/s	3.7	3.0
bytes/s	1484	1345

Babel test case 1 (0%), Hello = 0.25s		
	Intermediate nodes	End nodes
packets/s	6.1	5.5
bytes/s	2611	2535

Batman-adv

Batman-adv default settings, O = 1s						
	Test case	0%	2.5%	5%	10%	25%
Intermediate nodes	packets/s	28.1	25.2	25.8	23.5	19.5
	bytes/s	2515	2233	2225	1958	1408
End nodes	packets/s	11.7	11.2	10.8	10.0	7.0
	bytes/s	1065	977	922	804	484

Batman-adv test case 1 (0%), O = 0.5s		
	Intermediate nodes	End nodes
packets/s	33.7	14.6
bytes/s	4398	1977

Batman-adv test case 1 (0%), O = 2s		
	Intermediate nodes	End nodes
packets/s	21.4	8.6
bytes/s	1440	573

Batman-adv test case 1 (0%), O = 4s		
	Intermediate nodes	End nodes
packets/s	12.1	5.5
bytes/s	680	306