ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Marc Liechti

Real-World Evaluation of Ad Hoc Routing Algorithms

Semester Thesis SA-2012-35
July 2012 to April 2013

Tutor: Mahdi Asadpour
Co-Tutor: Dr. Domenico Giustiniano
Supervisor: Dr. Prof. Bernhard Plattner

Abstract

In the last decade, a lot of new routing algorithms have been proposed and some of them have been implemented for current computer architectures. In this semester thesis, we evaluated relevant routing algorithms in a real-world environment consisting of multiple virtual machine nodes for some predefined scenarios and measured their performance. The focus was on scenarios which are very likely to happen in the environment of flying drones as in the SWARMIX project (see [19]). We also measured the impact of using virtual machines instead of real hardware. Our results show that B.A.T.M.A.N.-Advanced outperforms B.A.T.M.A.N. and OLSR.

Contents

Chapter 1

Introduction

In existing research, we find many topics in simulating routing algorithms for MANET (Mobile Ad-Hoc Networks) and evaluating their performance by means of using network simulators like ns-2 ([9], [18], [17]). In this semester thesis, we don't use a network simulator, but we create a testing environment (called testbed) consisting of multiple virtual machines running on two laptops with real wireless dongles. We want to use a real-world testbed, since there has a lot of work been done with simulators and it's known that an implementation of a routing algorithm, which seems to perform well in a simulation environment, has nearly never the same performance when used in a real network ([14]). This semester thesis belongs to the SWARMIX [19] project, so we will focus on scenarios which are likely to happen in such a heterogeneous multi-agent system.

SWARMIX is a project by the Swiss National Science Foundation for laying the foundations for the design, implementation, and adaptive control of heterogeneous multi-agent systems working in cooperation to solve distributed tasks. For this type of system to work, a stable network has to be built up betweem all the mobile parts consisting of humans, animals and robots. Because all the nodes are only able to send in a short range, routing algorithms are needed to forward the packets to the correct destination. This task is not so easy since all involved parts are mobile and therefore the routing algorithm needs to adapt to the topology changes. An overview of such a network is depicted in Figure 1.1.

1.1 Motivation

The motivation for this semester thesis is to test a new approach for creating a testbed with multiple virtual machines. Recent works which built a real-world testbed used separate laptops or routers on which the routing protocols were running. We present a new approach by using

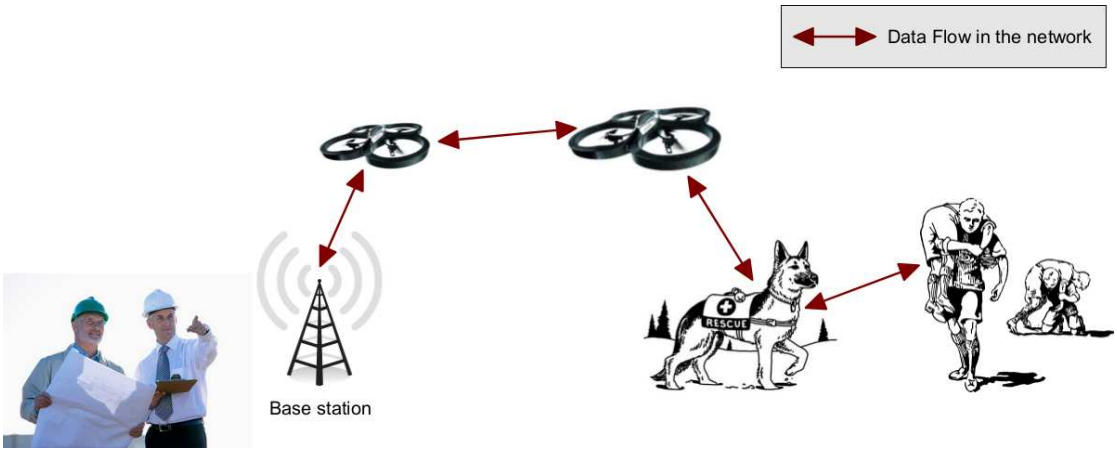


Figure 1.1: SWARMIX Overview

virtual machines as nodes and also determine the influence of using them instead of real computers. Using this testbed, we then evaluate the best routing protocol for the SWARMIX project.

1.2 The Task

The task consists of:

- Find dynamic routing algorithms and evaluation metrics in the literature
- Getting familiar with network tools and virtual environments like VirtualBox, iptables, arptables, ebtables, iperf, wireless-tools
- Installing and setting up the wireless dongles on all nodes
- Installing the routing algorithms on all nodes
- Drawing multiple scenarios with various topologies
- Reflect how to measure the metrics found
- Performing the scenarios and collecting results

1.3 Related Work

There has been little investigation in the literature about similar works which consider a real-world model and the mobility of the nodes. In [7] we find a real-world evaluation of OLSR, BABEL and B.A.T.M.A.N., but the stations are stationary. The work is still related to ours because they also evaluated some of the metrics we do and used mostly the same routing algorithms. Another work is [13]. They present a real-world evaluation of Babel, OLSR and B.A.T.M.A.N. and have done a stationary evaluation of the metrics. Then we find a lot of routing protocol comparisons in a simulation environment like ns-2, see [17] and [8] for instance.

1.4 Overview

Chapter 2 introduces the routing protocols and gives a background of how they are working. This will also be of importance for interpreting the results. Chapter 3 presents our testbed and will also shortly mention the hardware used. In chapter 4 we will show the tests we made, which consists of the scenarios and the metrics. The results from the comparison of the routing protocols is presented in chapter 5. The interpretation can be found in chapter 6 and the final conclusion in chapter 7.

Chapter 2

Routing Protocols

The routing algorithms which we consider in this semester thesis have to fulfill the following three criterias:

- Capable of handling mobility
- Implemented for linux (native support)
- Running on Gumstix/OpenEmbedded

First they must be able to handle the mobiltiy since we want to use it in a mobile network. They also have to be implemented for linux and should run natively on the computer architecture, because we don't want to have to install some additional runtime environments. Thirdly the routing algorithm needs to run on Gumstix/OpenEmbedded. This is the platform which runs on the drones. We finally come up with the following routing algorithms.

2.1 AODV

AODV (Ad-hoc on-demand distance vector) [20] is a reactive routing protocol. This means that a route to another node is only requested if needed and is not already known before. When a node wants to send data to a destination node, it sends out a route request (*RREQ*). Each node receiving this message looks if it already has a valid route to the requested destination. If this is true, the node responds with a route reply (*RR*) to the initiator, else it sends out the *RREQ* again until it reaches a node with a valid route and stores the source of the request in a table. AODV avoids the counting-to-infinity problem by using sequence numbers on route updates. The main disadvantages of this kind of approach is the high initial latency when trying to connect the first time to another node and the time to adapt when a better path appears, because this requires the origin node to repeat the route request. We've chosen this routing protocol more for comparison reasons. As we will see later, most of the times AODV was not even able to create a route between a simple network as also determined by other works (see [7] and [13]).

2.2 OLSR

OLSR (Optimized Link State Routing) [6] is a proactive routing protocol. This type of algorithms want to achieve that each node in the network has always a complete valid list of all destinations. The information about changes and existence of nodes is spread by periodically sending out routing informations through the network. In the case of OLSR, the information is spread by *HELLO* and *TC* (topology control) messages. A node in an OLSR network discovers his 1-hop and 2-hop neighbours by the continuous sent *HELLO* messages. These packets contain the already known 1-hop neighbours as well as the state of the connection to them. From its 1-hop neighbours, each node elects one or more multipoint relays (*MPR*), over which it can reach all 2-hop nodes. The elected MPRs distribute the *TC* messages, which contain a list of nodes which elected this node as *MPR*. This is done to make the flooding more efficient. OLSR is one of the

most often used ad-hoc routing protocols, for instance in the freifunk networks in the German region [11] and is also designed to work with mobile ad-hoc networks. That's also the reason why we've chosen to take measurements with this protocol. The main disadvantages although are the amount of routing data overhead sent out and the slow reaction to topology changes.

2.3 B.A.T.M.A.N.

B.A.T.M.A.N. (Better Approach To Mobile Adhoc Networking) [16] is a routing protocol which is developed by the freifunk community [11]. The goal is to replace OLSR with B.A.T.M.A.N. in the future. The main idea behind B.A.T.M.A.N. is to only distribute the knowledge of the best connections to all nodes to reduce the package overhead. For that, a node with the protocol running sends out periodical broadcast messages which indicate that the node is alive. This message is repeated by each node. A node receiving such a message stores in its table in which direction it has to send the packet. This approach is in literature also called "biologically inspired", since it's not belonging to one of the routing protocol classifications (see section 2.5). Because B.A.T.M.A.N. is a relatively new approach and developed by the freifunk community, we also want to test it in this semester thesis.

2.4 B.A.T.M.A.N.-Advanced

B.A.T.M.A.N. has a little brother called B.A.T.M.A.N.-Advanced. The difference is that the latter does all the routing on ISO/OSI layer 2 (data link layer). This means that every layer 3 protocol (network layer) runs on it. The routing scheme behaves more or less the same as the former did, but the packets are tunneled through a virtual interface *bat0* to the destination node. Since this is a new approach and looks very interesting, we also selected this routing protocol.

2.5 Others

In literature, we also find other approaches for routing algorithms, like flow-oriented, hybrid, location aided and hierarchical routing. A hybrid implementation is the Zone Routing Protocol (ZRP), but the code was quite outdated, the latest version we found was from 2005. For the other classes, we didn't find any implementations. Most of them have been implemented for network simulators, but haven't found a way into current computer architectures.

Chapter 3

Testbed

To perform the measurements, we create a testbed. Instead of using multiple laptops, we use Virtual Machines to build multiple nodes. On two laptops, a separate USB dongle is forwarded to a running instance of VirtualBox, so the communication is indeed happening over the wireless dongles. To see the impact of virtualization on the overall performance, we've done some throughput and delay tests (see chapter 5.2). Because we've set the MCS (Modulation and Coding Scheme) to 1, a throughput of maximum 30 Mb/s can be reached (see [1]) in our operating mode (channel 60, 5.3GHz, 40MHz bandwidth). We limit the operating mode of the dongles due to the mode switches which can occur during testing. We want to have equal conditions for all protocols. To create a topology in our testbed, we use iptables, arptables and ebtables to disable a link between two nodes. We have to use ebtables because batman-adv uses layer 2 routing, and iptables can only block layer 3 packages. Arptables is used for blocking the arp requests and replies from the blocked nodes. The overall setup is depicted in 3.1. When for instance node 4 wants to send data to node 3, the data originating on Virtual Machine 4 on the second laptop sends the data with its attached USB dongle over the wireless channel to node 1, which is running on the first laptop. According to the routing algorithm used, the data is then forwarded until it reaches the node 3, always over the wireless channel.

The main disadvantage of this testbed is that it only allows to either completely block or unblock other nodes, and not to reduce the link quality between them. All mentioned protocols calculate a link quality to determine the best route towards a destination. But because all wireless dongles are in a short range, the received quality is very good and therefore the channel is more stable. Another deviation from the real world model is the data link layer. A node which is for instance completely blocked by all other nodes will still use the same channel and therefore interfere the other nodes, which would not happen in the real case (e.g. the node is far away). The main advantage is that we can economize on hardware since there are multiple nodes running on one laptop and it's therefore also highly scalable.
The advantages and disadvantages of using this testbed are outlined in the following comparison.

Advantages

- Economize on hardware
- High scalable
- Channel is more stable
- Easy testable in one room
- Higher control

Disadvantages

- Influence of the Virtual Machines
- Testbed cannot completely emulate the real world
- Limited possibilities to reduce the signal strength

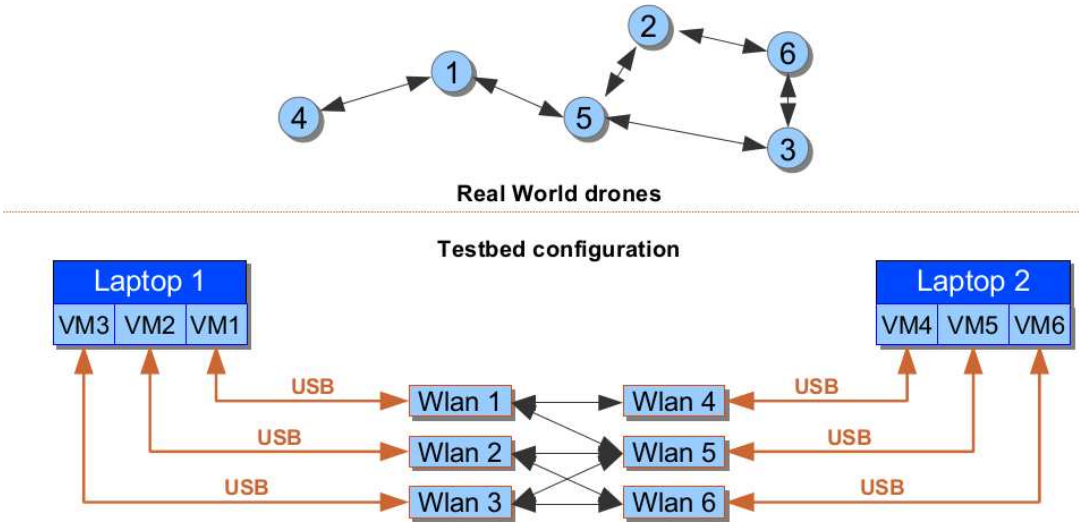


Figure 3.1: Testbed

Figure 3.2: Linksys AE1000 USB Dongle

3.1 Hardware

For this semester thesis, we've considered two USB dongles. The first is a newer one from TP-Link, the other is from Linksys. To have enough bandwidth, it is important for the hardware to support 802.11N mode. The dongles also need linux support. For reasons we'll mention below, we finally use the Linksys dongle.

3.1.1 Linksys AE1000

The Linksys dongle AE1000 has linux support from the manufacturer [12]. The drivers work very good. Also some tests on the throughput show very good results. Between two laptops, we are able to transmit up to 100 Mb/s. Because 802.11N can switch between different modes and we want to have equal conditions during the whole time the measurements are running, we set the MCS to 1, so that the results are more comparable (for MCS settings, see [1] page 347). This limits the max throughput to 30 Mb/s.

3.1.2 TP-Link TL-WN822N

We've also considered another wireless dongle. The TP-Link TL-WN822N also supports 802.11N mode and has support for linux because it can be used with the compat-drivers (see [3]). With this dongle however, we are not able to get throughputs over 1 Mb/s. Also when plugging the dongle into Virtualbox, the message "The device could not be initialized" appears on

Figure 3.3: TP-Link TL-WN822N USB Dongle

the screen. Also with VMware the problem doesn't disappear. We were not able to solve the problem in the given time and decide to use the Linksys AE1000 USB dongle.

3.1.3 Laptops

For creating the testbed, we use two laptops where we plug in the USB dongles. One of the two is a Sony Vaio VGN-Z11WN and the other is a Lenovo Thinkpad T400. Both laptops have a Intel Centrino 2 Duo processor inside (Intel Core2Duo P8600 2.4GHz) which supports IVT (Intel Virtualization Technology) which is essential for our testbed to run. Both laptops have 4GB of RAM.

3.2 Virtual Machines

As we've mentioned above, we use virtual machines running on two laptops as nodes. We use VirtualBox in the version 4.2.10 with the host operating system Ubuntu 12.10 (quantal quetzal). For doing some measurements on AODV, we have to switch to Ubuntu 10.04, because AODV is only running on linux kernels 2.4 and 2.6.

3.3 Control Unit

Over a separate, cable-connected network we send the commands to the virtual machines. We've created a control unit on which we can give commands to all nodes, for instance to disable a link (block with iptables) or start transmitting data to another node. For that, we use netcat [4] to listen on a port of the cabled network and redirect the data arriving to the bash shell. We don't use ssh command forwarding since the delay added by authentication is too big. We've created commands for the complete initialization of the wireless interface and getting it up, (de-)blocking of other nodes, starting ping and throughput tests and logging everything. All the scripts can be found in the Appendix B.

Chapter 4

Test

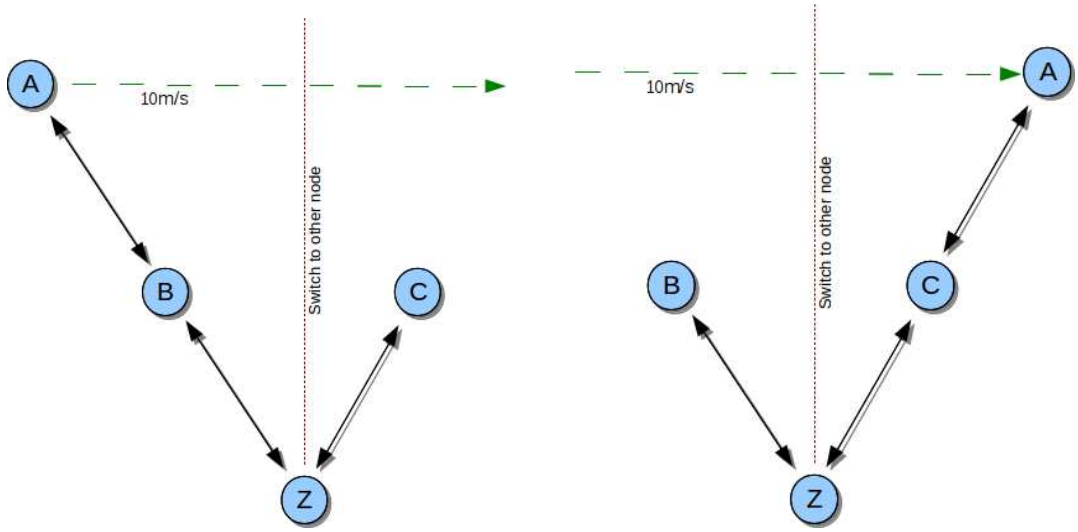
To test the routing protocols, we use these scenarios and metrics to evaluate their performance. Note that each routing algorithm offers many parameters to adapt. We use each of them with its default settings. We know that you could tune it to be more efficient for any of the scenarios considered. Each scenario takes 5 minutes to complete. It takes so long to be sure that each routing algorithm is able to settle in this time.

4.1 Scenarios

In this semester thesis, we want to measure some predefined scenarios which represent a given topology change of the flying drones in the SWARMIX project. Contrary to some approaches of other works which used a random walk of the stations, we predetermine by hand how this scenarios should look like, because the movements of the nodes shall be as realistic as possible. The drones get controlled by a base station and their pattern of flight is given and not unpredictable. Finally we come to the following 5 scenarios.

4.1.1 Scenario 1 - Switch to another node

In the first scenario, the ability of the routing algorithm of switching the traffic over another node is tested. This scenario can be understood as a drone (*Node A*) flying away from a close node (*Node B*) and gets disconntected. A new node (*Node C*) has to be found and the traffic needs to be forwarded to it to reach another node in the network (*Node Z*).



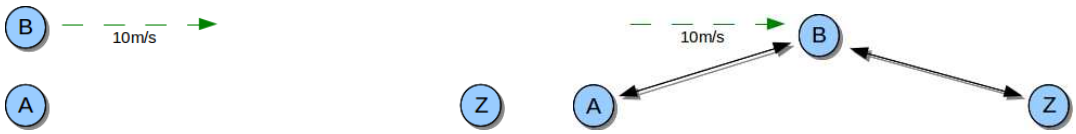
(a) Situation the node moves

(b) Situation after the node has moved

Figure 4.1: Scenario 1 - Switch to another node

4.1.2 Scenario 2 - Ferry

In the second scenario the performance of the routing algorithm concerning the ferry model is tested. It is important that a node can be used as a ferry for transporting data from one node to another. Because the routing algorithms we test are not delay tolerant, the performance of setting up a connection between the two nodes is evaluated. *Node A* constantly tries to send data to *Node Z*. When *Node B* reaches the middle of the two nodes, the connection is set up and the routing algorithm needs to find the path to the destination.

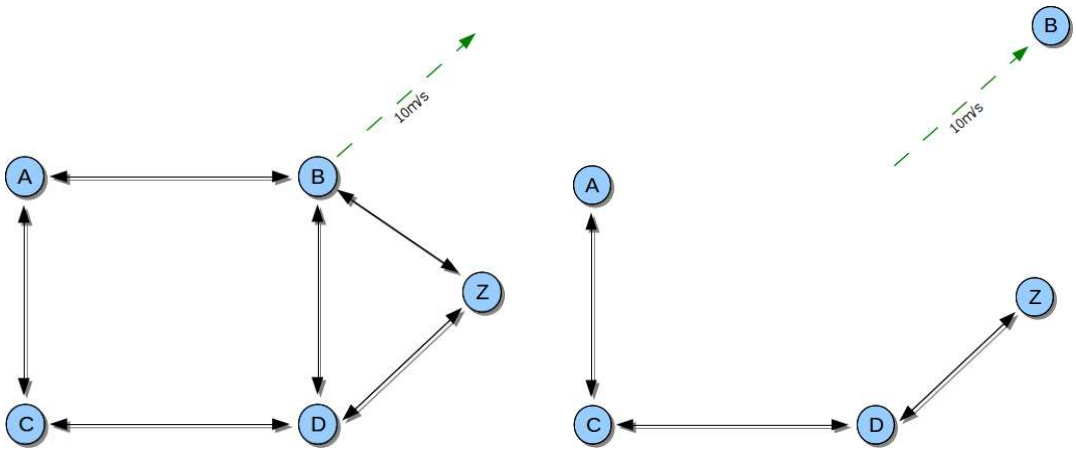


(a) Situation the node moves (b) Situation after the node has moved

Figure 4.2: Scenario 2 - Ferry

4.1.3 Scenario 3 - Self healing ability

The third scenario differs from scenario 1 that *Node A* has two valid paths to the destination *Node Z*. At the beginning, the routing algorithm will use the shorter path over *Node B*. When it flies away, the routing algorithm needs to heal itself and use the longer path over *Node C* and *Node D*.



(a) Situation the node moves (b) Situation after the node has moved

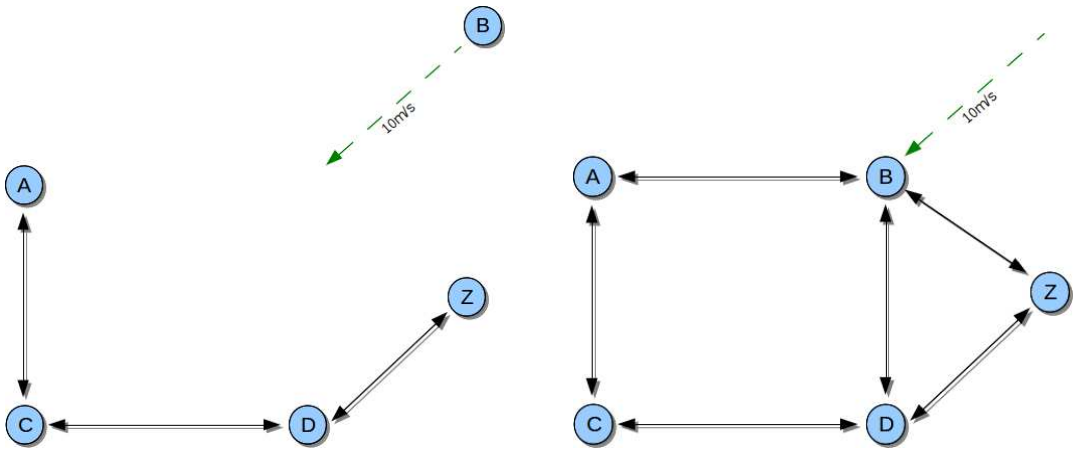
Figure 4.3: Scenario 3 - self healing ability

4.1.4 Scenario 4 - Path optimization

For considering how long it takes for the routing algorithms to use a shorter route, we add the scenario 4. In the beginning, *Node A* has only one path to the destination *Node Z*. But after a while, another *Node B* arrives and adds a shorter path.

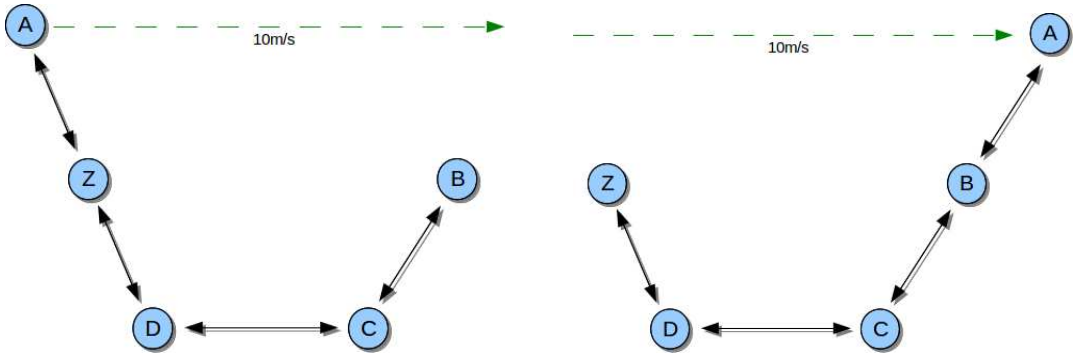
4.1.5 Scenario 5 - Long path

Because we dont't have a scenario with a long path yet, we add the fifth. It is nearly the same as scenario 1, but with more nodes involved. This is also interesting to see how the throughputs, round-trip times and packet losses behave under such a long path.



(a) Situation the node moves (b) Situation after the node has moved

Figure 4.4: Scenario 4 - Path optimization



(a) Situation the node moves (b) Situation after the node has moved

Figure 4.5: Scenario 5 - Long path

4.2 Metrics

Many metrics to evaluate routing algorithms are proposed in literature. There's an RFC (2501) [10] which suggests metrics to consider which have also been used in a related work [9]. We find nearly the same metrics in [8]. In [17] even more detailed metrics have been used. Getting the same metrics for a real-world evaluation is very hard. For example energy consumption comparison is nearly impossible to measure, since most of the routing protocols run as a kernel module. We focus in our work on metrics which are feasible and important to the network stability. We also want to take metrics specific for the scenarios we consider (see chapter 4.1), because the flying drones need a fast reaction of the network to topology changes. Finally, we use the metrics which can be found below.

4.2.1 Throughput

One of the most important metrics seems to be throughput. It's defined as the average rate at which packets are successfully delivered to a destination. Although in our case, where the number of nodes is very small compared to simulation environments, throughput will probable be more a bottleneck of the wireless dongles than the routing protocols. So only in scenario 4, where multiple links to the destination are possible, and in scenario 5, where the traffic goes over multiple hops, this metric should differ most. We make this test after each scenario has settled, to have some results for comparison, because the throughput goes to 0 when no valid route exists. We don't measure during the topology change since this is evaluated by the metric Route Convergency Test discussed later.

Measurement The throughput is measured with iperf, always sending from *Node A* to *Node Z* after the topology has changed. We start the utility with the following options:

```
iperf -c <NODE_Z_IP> -t 60 -l 1400 -i 10 -u -b 100M
```

We measure 10 seconds intervalls over total 1 minute and set the bandwith to 100 Mb/s, so iperf transmits as fast as possible. We reduce the packet length to 1400 bytes because B.A.T.M.A.N.-Advanced cannot use the full 1500 bytes packet load, because of the additional header (see [15] in the wiki site Fragmentation).

4.2.2 End-to-end delay/round-trip time

End-to-end delay is the delay added by the the network until a packet reaches its destination. As the metric before, this one will equally probable mostly depend on the hardware than on the routing protocol itself, because there are not a lot of scenarios which have multiple paths between the interesting nodes. We measure the round-trip time, so roughly two times the end-to-end delay, since its easier to measure with available network tools.

Measurement Round-trip time measurement start on node A with the simple ping tool, again after the scenario has converged to the new state:

```
ping -i 0.5 -c 60 <NODE_Z_IP>
```

We send 60 packets to the Node Z in the intervall of 0.5 seconds.

4.2.3 Route convergence latency test

Because our work consists more on evaluating the convergence ability and how fast they adapt to topology changes, we use this metric found in [7]. It's mentioned as a self-healing ability of the routing protocols. It can be seen as the time the protocol needs to adapt to a change in the topology, until the communication works again.

Measurement Route convergence is measured with the output of the routing algorithms on all nodes. We then calculate the difference between the initial timestamp when the node moves and when the new topology is known such that the communication is built up again. For time synchronization, we use the network time protocol (ntp).

4.2.4 Routing overhead

The routing overhead is defined as the number of packets or bytes added by the routing algorithm to the network. We also measure this metric by using tcpstat and corresponding filters to count the packets and bytes captured. For B.A.T.M.A.N.-Advanced however, we use the internal statistic tool of batctl, because the routing is happening on layer 2 and for tcpstat we can only define layer 3 filters (see [5]).

Measurement We measure the routing packages overhead with *tcpstat* for B.A.T.M.A.N. and OLSR with corresponding filter definitions, and for B.A.T.M.A.N.-Advanced with the internal statistics tool *batctl*. *tcpstat* reports statistics about an interface and can be configured to filter only specific packets. We measure over the whole scenario' time and use only the statistics reported on node A.

4.2.5 Packet loss

Packet loss is another important metric for our application. It's defined as the percentage of packets lost during a given time interval compared to the total sent packets. Since we want to stream e.g. audio or video over over the network and use probably UDP, a huge packet loss would result in useless data arriving at the destination.

Measurement Packet loss is reported by iperf as a server report after each measurement. We use this data for our results.

4.2.6 Path optimality

The path optimality is the difference between the number of hops a packet takes to reach its destination and the number of hops of the shortest path. This metric is not very helpful in our work, because we only use at maximum 5 nodes. In scenario 4, where a better path is available later, this could be considered. But the result on how long it took to find the new path is a better benchmark for that case. Also measuring this metric can be very hard in a real-world testbed. We could use the standard ping tool, which reports the remaining TTL (time to life) of the response packet. The problem with this approach is that B.A.T.M.A.N.-Advanced uses a tunneling based on layer 2, which doesn't decrease the TTL on each hop.

Chapter 5

Results

5.1 Distance measurements

To see the performance of the Linksys dongle, we make a distance measurement. In figure 5.1 we see that the link has a good throughput up to 120m. Note that tests have been done outside on 1.5m over the ground. We realized that the distance increases when the dongles are in the air.

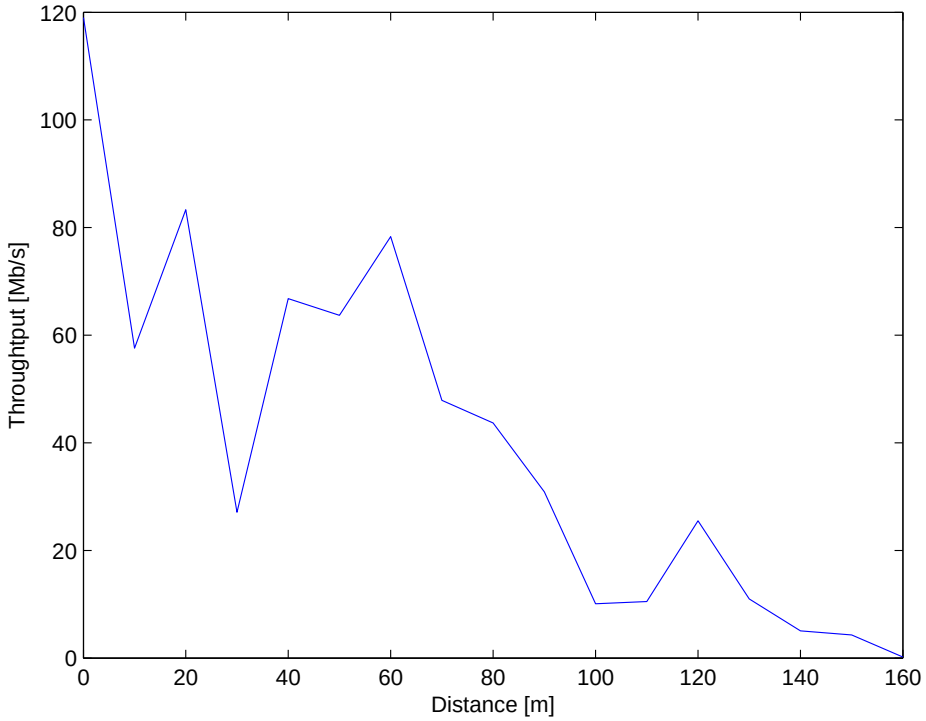
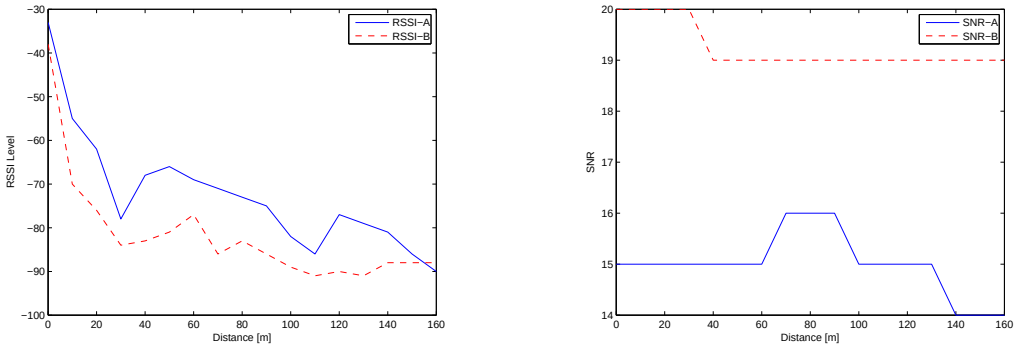


Figure 5.1: Distance to throughput plot

During the throughput test, we've also logged the RSSI (Received Signal Strength Indication) and SNR (Signal to Noise Ratio). The results are depicted in figure 5.2. Because the Linksys AE1000 have two antennas, there are always two lines in these figures.



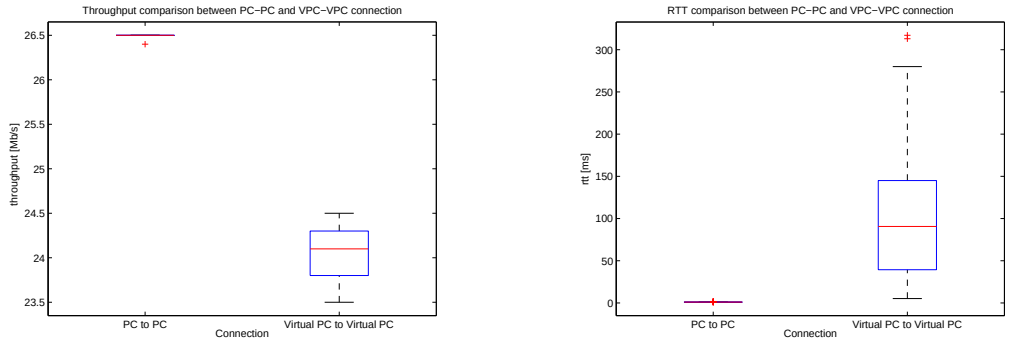
(a) Distance to RSSI result

(b) Distance to SNR result

Figure 5.2: Distance to RSSI and SNR results

5.2 Virtual machine to pc-pc comparison

In the figure 5.3, we see the influence of the virtual machine compared to a laptop-laptop connection. We see that the throughput of a UDP connection over 10 seconds decreses by approximately 2 Mb/s. The round-trip time increases much more by roughly 100ms in the mean, and the variance is also much bigger. Altough there is quite a big influence, we test the routing algorithms with this testbed, since the conditions are the same for all of them. At the end, they are still comparable.



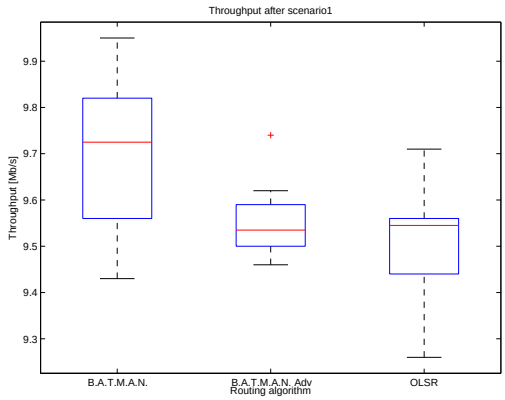
(a) Throughput comparison

(b) RTT comparison

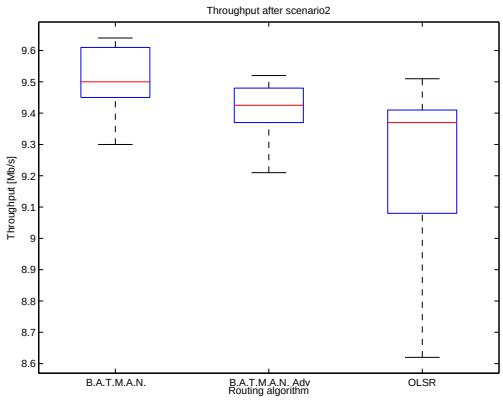
Figure 5.3: Throughput and RTT comparison

5.3 Results from the scenarios

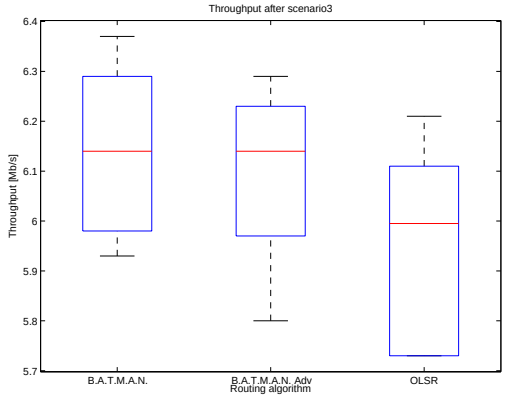
In this section, we present the results from the framework. We extracted all the data from the log files which have been created by all the tools used, e.g. iperf, ping, tcpstat and of course the log output of the routing algorithms. Each scenario has been passed 10 times as in [7]. For each run the mean and statistics of the data have been evaluated and is depicted in the following figures. Please note that we didn't add the results for AODV in the plots. As already noticed, AODV failed to maintain or sometimes even establish a connection. So the results for AODV aren't very precise. The results in the text are calculated with values when AODV worked. Please keep in mind that these results are therefore inaccurate and only reported for the sake of completeness.



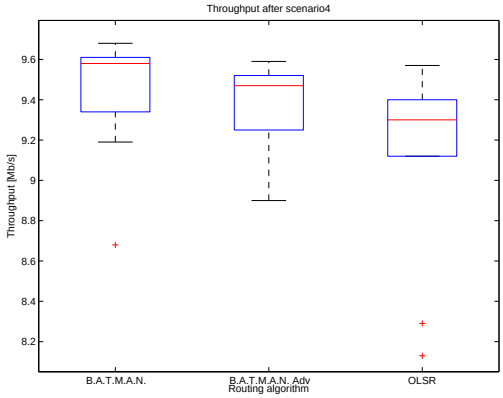
(a) Throughput results from Scenario 1



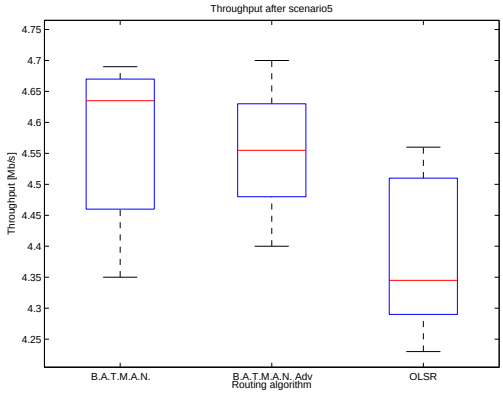
(b) Throughput results from Scenario 2



(c) Throughput results from Scenario 3



(d) Throughput results from Scenario 4



(e) Throughput results from Scenario 5

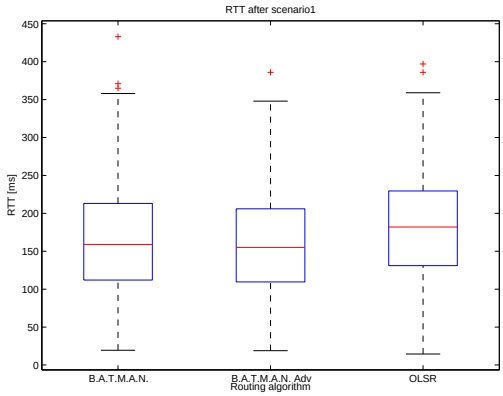
Figure 5.4: Throughput results for all scenarios

5.3.1 Throughput

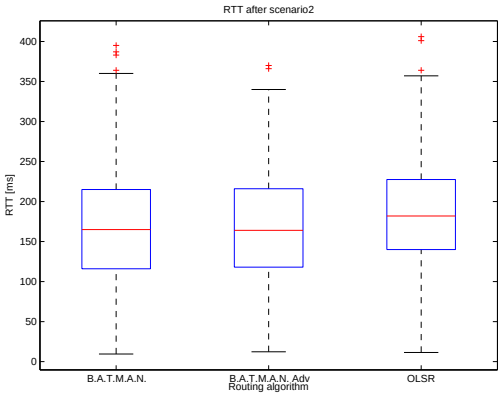
We can see from the plots 5.4, that all routing algorithms have more or less the same throughput in the range of 9.4 to 9.8 Mb/s for 2 hops (scenario 1) and down to 4.3 to 4.7 Mb/s for 4 hops (scenario 5). This should be sufficient to stream audio and low-resolution video over the MANET while the topology is stable. And we have to keep in mind that we set the MCS parameter to 1 and limited the max throughput between two nodes to 30 Mb/s. It's also noticeable from the graphs, that B.A.T.M.A.N. is always a little bit faster than the others, but the difference is nearly negligible, since it's in maximum 0.3 Mb/s faster than OLSR and B.A.T.M.A.N.-Advanced in the mean. The variance seems to be equal. When AODV worked, it reached a throughput of 7.49 Mb/s in the mean for scenario 1, 7.83Mb/s for scenario 2 and 4.89Mb/s for scenario 3. For scenario 4 and 5, throughput measurements with AODV were not possible. We cancelled the

tests after several attempts.

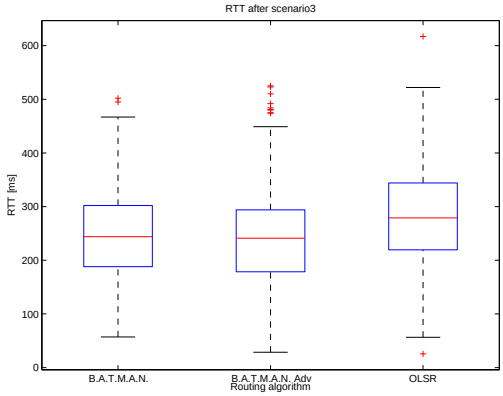
5.3.2 End-to-end delay/round-trip time



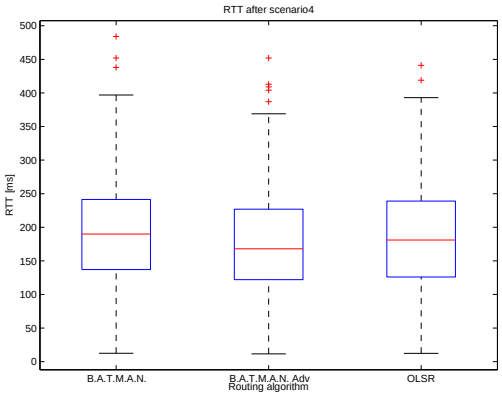
(a) RTT results from Scenario 1



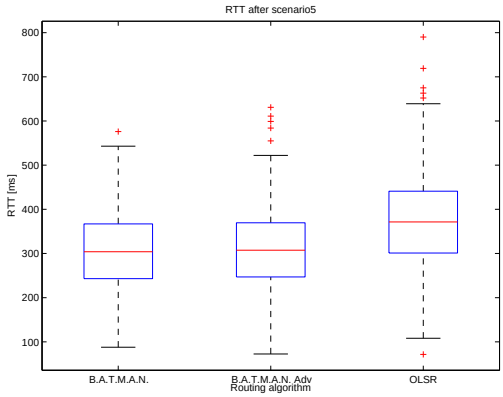
(b) RTT results from Scenario 2



(c) RTT results from Scenario 3



(d) RTT results from Scenario 4

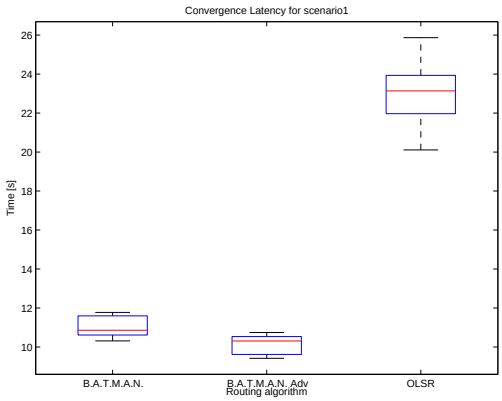


(e) RTT results from Scenario 5

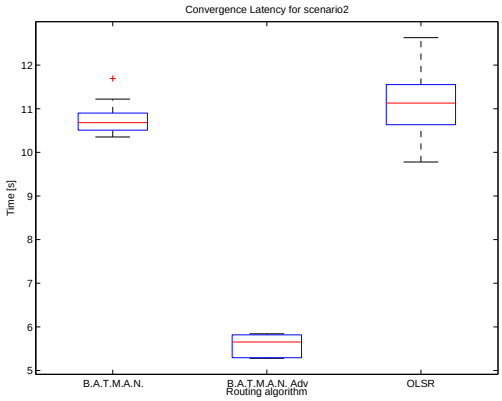
Figure 5.5: RTT results for all scenarios

We see from figure 5.5a an average RTT of 150 to 200ms in the mean for 2 hops and up to 250 to 400ms delay for 4 hops. When we look at the observation made by the virtual machine to pc-pc comparison result, we realize that most of the delay comes from the fact that we used virtual machines. But the results show that B.A.T.M.A.N. is the protocol with the best performance, but again nearly negligible. For scenario 1-4, the difference among the three is in the mean not even 50ms. And also the deviation is almost the same. AODV had a mean RTT for scenario 1 of 183.4ms, 182.8ms for scenario 2 and 265.97ms for scenario 3. For the other two scenarios AODV was not able to establish a connection.

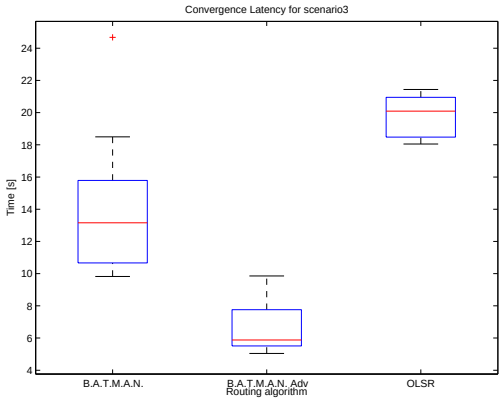
5.3.3 Route convergence latency test



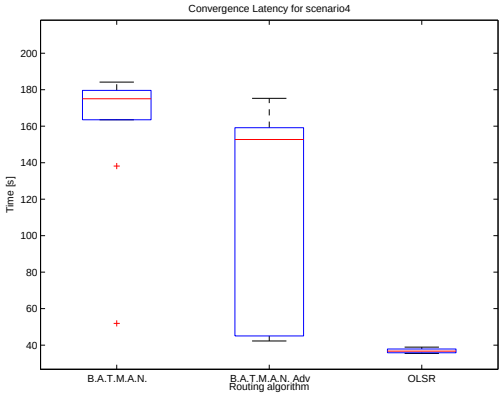
(a) Convergence time results from Scenario 1



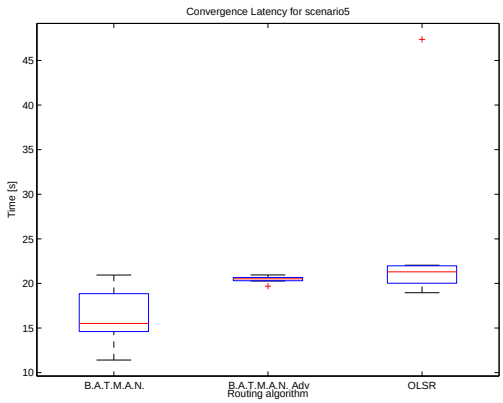
(b) Convergence time results from Scenario 2



(c) Convergence time results from Scenario 3



(d) Convergence time results from Scenario 4



(e) Convergence time results from Scenario 5

Figure 5.6: Convergence time results for all scenarios

Route convergence latency tests are very important for the SWARMIX project, because these are the tests which concern the mobility of the nodes. Roughly, we can say that B.A.T.M.A.N. and B.A.T.M.A.N.-Advanced perform best compared to OLSR. For scenario 1, where the communication is switched over to another node, OLSR needs in the mean time twice as long as B.A.T.M.A.N.-Advanced to find the new route. Only in scenario 4, where the connection stays up, but a shorter path is added, OLSR beats the others. AODV managed to adapt to the topology changes in 2.28s, 1.91s and 2.29s for the first 3 scenarios in the mean. For the other two scenarios we were not able to extract some results. The reason for these huge differences are the class of routing algorithm the protcols belong to, the messages interval for the pro-active protocols and the internal parameters set. As mentioned in chapter 2, OLSR, B.A.T.M.A.N. and

B.A.T.M.A.N.-Advanced use periodically sent *HELLO* messages to distribute their existence in the network. If a route fails, the routing algorithms notice that the packets aren't any longer received directly from the node before. Therefore, they internally compare all links known to the destination node and chose the link which has the best quality indicator. To avoid fast swapping of the routes, these indicators only adapt slowly and can be adjusted by some parameters. So the time when the new route is used depends on these parameters and also the interval of the messages, since they also influence how fast this quality indicator changes. Very outstanding is the result from AODV (when it workend). It managed to discover the new route in about 2s. This is due the fact that AODV is a reactive protocol. Once it recognises that the link failed, it will send out a new *RREQ*, which will be answered very fast and the new route can then be used. The idea behind AODV is very good, but the implementation seems to fail for bigger networks.

5.3.4 Routing packets overhead

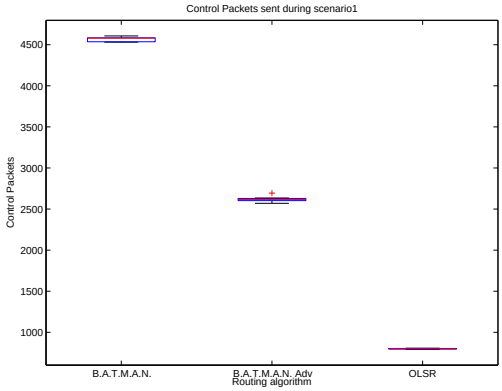
Routing overhead is also important, since it's an indicator for how much traffic a node adds to the network. We can see that OLSR adds only few packets and B.A.T.M.A.N. most. This is true for all scenarios, which is also expected. The number of packets sent out is proportional to the packet interval of the routing protocols. AODV added 1415.25, 1397.75, 1464.33, 1373.0 and 1109.33 packets to the scenarios 1 to 5. This amount is very low, since AODV only looks for a route when requested in contrary to the pro-active protocols.

5.3.5 Routing data overhead

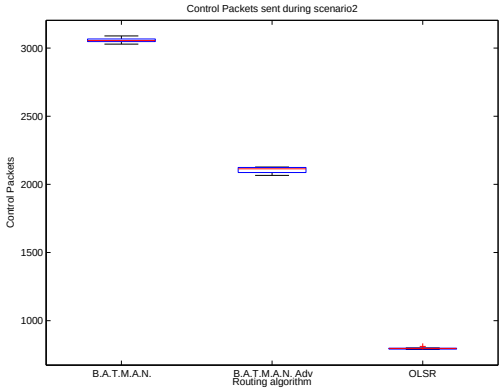
We've seen that B.A.T.M.A.N. adds most packages to the network, and also most data summed up over the scenario time. Compared to the result before, we notice that B.A.T.M.A.N.-Advanced actually adds a lot of packages, but they're quite small, so the amount of added data is not half the data B.A.T.M.A.N. adds as could be expected. For AODV, the results are 67620, 67002, 69802.7, 65928 and 53454 bytes in the mean.

5.3.6 Packet Loss

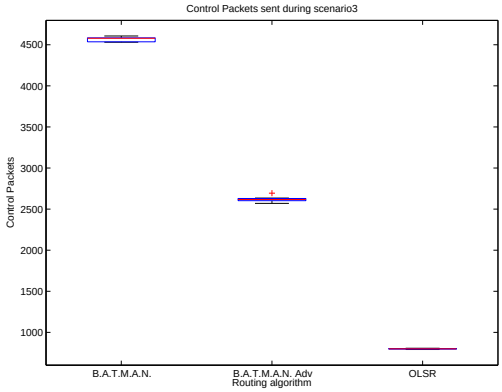
After each scenario, we did a packet loss test with all the routing algorithms. It's astonishing that the packet losses greatly differ, since we could expect that this is a matter of the channel. We can see that B.A.T.M.A.N. and OLSR have more or less the same packet loss, while B.A.T.M.A.N.-Advanced performs far better. While B.A.T.M.A.N.-Advanced looses only up to 5% of the packages, OLSR and B.A.T.M.A.N. loose in scenario 5 over 5 hops about 25%. AODV lost, even if it managed to establish a connection, 74% for scenario 1 up to 97% in scenario 3.



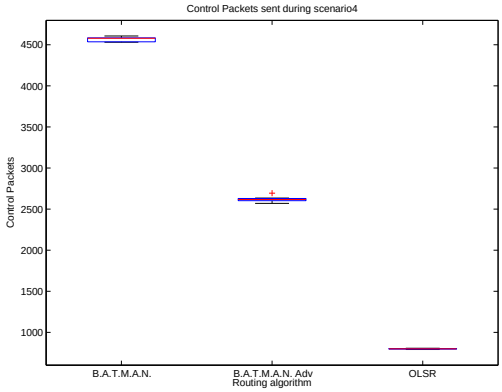
(a) Routing packages overhead from Scenario 1



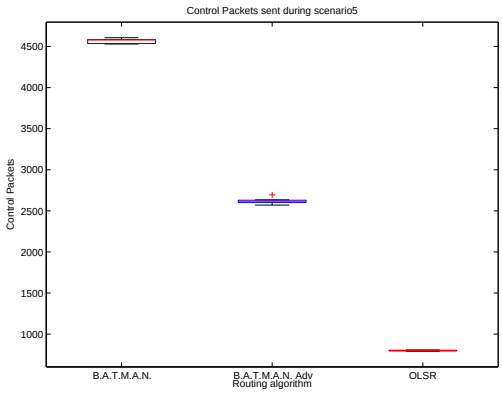
(b) Routing packages overhead from Scenario 2



(c) Routing packages overhead from Scenario 3

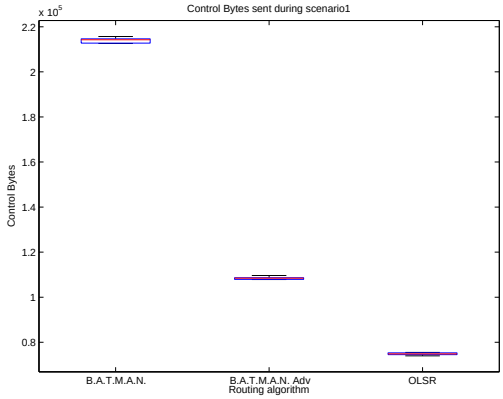


(d) Routing packages overhead from Scenario 4

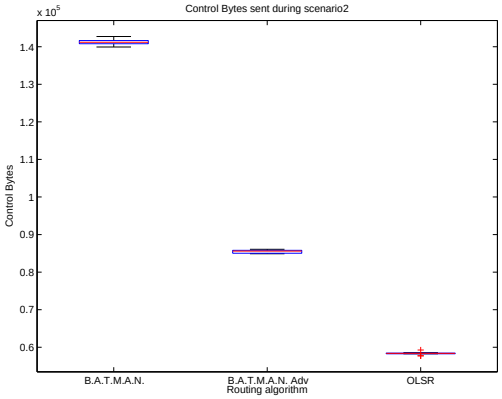


(e) Routing packages overhead from Scenario 5

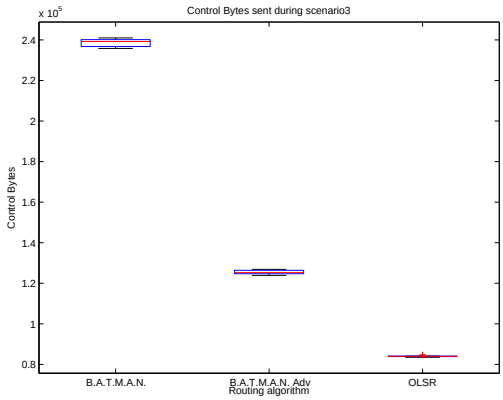
Figure 5.7: Routing packages overhead for all scenarios



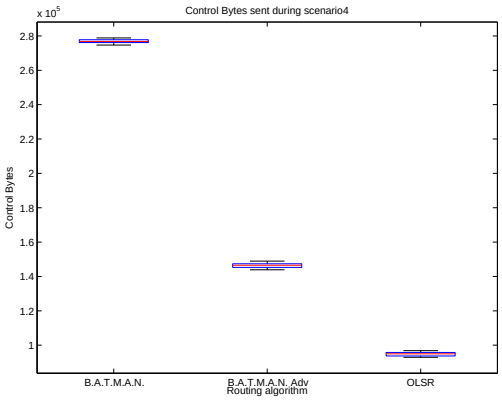
(a) Routing data overhead from Scenario 1



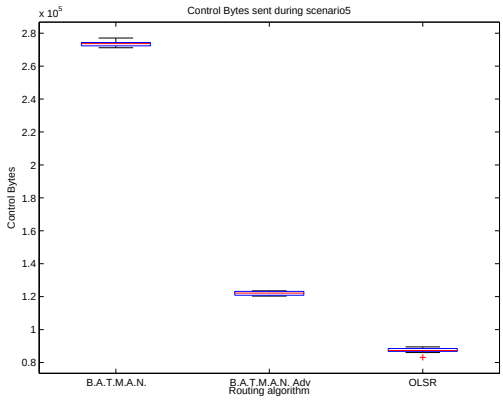
(b) Routing data overhead from Scenario 2



(c) Routing data overhead from Scenario 3

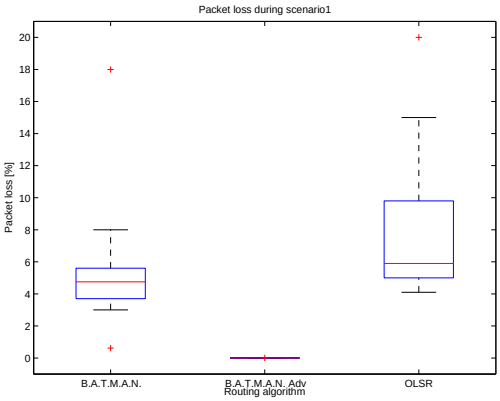


(d) Routing data overhead from Scenario 4

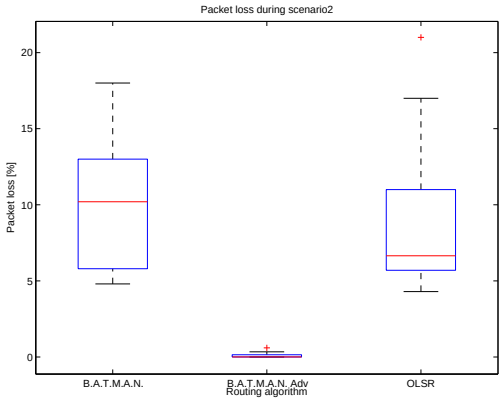


(e) Routing data overhead from Scenario 5

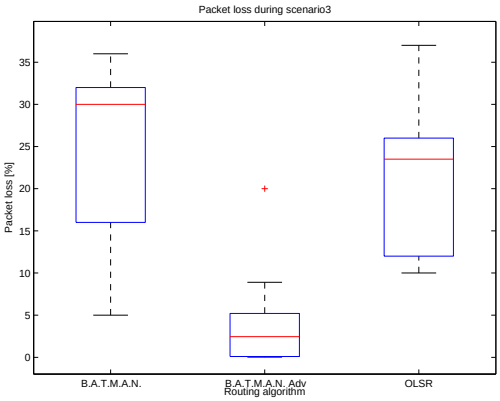
Figure 5.8: Routing data overhead for all scenarios



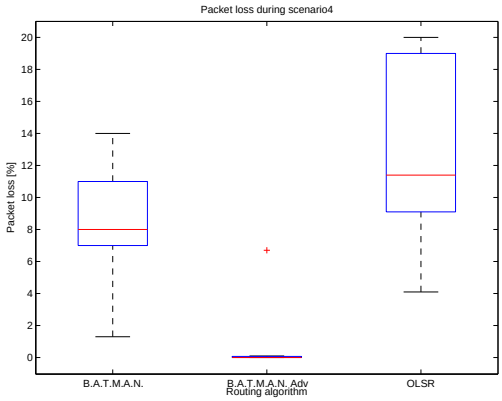
(a) Packet loss from Scenario 1



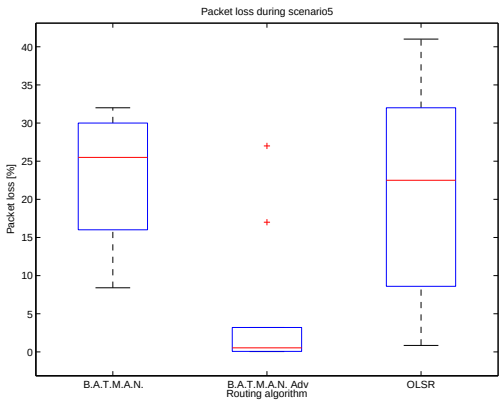
(b) Packet loss from Scenario 2



(c) Packet loss from Scenario 3



(d) Packet loss from Scenario 4



(e) Packet loss from Scenario 5

Figure 5.9: Packet loss for all scenarios

Chapter 6

Interpretation

Throughput The results from chapter 5 show that the throughput and round-trip time are for all routing algorithms in the same range. But it is noticeable that OLSR always performs a little bit worse than the other two protocols. We could guess that this comes from the added packet overhead the protocols generate or the higher packet loss. But if we look at these results, we see that a very low loss doesn't automatically mean that the throughput is higher or that a higher packet overhead automatically means that the the packet delivery ratio gets reduced due to this additional load. But another work [13] showed that throughput primarily decreases with the amount of packets and data the respective routing algorithm adds to the network. This direct influence cannot be seen in our results. We can argue that only considering 5 nodes isn't that much that there could be a dramatical influence. But for bigger ad hoc networks, it would be interesting to see if we would also see this impact. Since the round-trip time of OLSR is also minimal higher in all scenarios compared to the others, we can conclude that OLSR has more computational cost until it can forward the packages. B.A.T.M.A.N. and B.A.T.M.A.N.-Advanced seem to be more efficient there. And we should also not neglect that the virtual machines have a influence on the overall performance.

Round-trip time As already mentioned in the paragraph above, the round-trip time behaves like the throughput. The routing algorithms which reach a high throughput also perform well in the sense of small round-trip time. But the numbers are factors higher than expected. This is due to the fact that the virtual machines add a lot of delay to the network as shown in 5.2.

Route convergence latency test Most interesting for the SWARMIX project are the convergence latency tests, since this are the results which consider the mobility of the nodes. We see that for scenario 1, B.A.T.M.A.N. and B.A.T.M.A.N.-Advanced find the new route within 12 seconds, while OLSR has about 24 seconds. This most probably comes from the fact that OLSR uses a *HELLO* message interval of 2 seconds, while B.A.T.M.A.N. uses a originator interval of 0.5 seconds and B.A.T.M.A.N.-Advanced of 1 second. On the website of B.A.T.M.A.N.-Advanced, they also propose to decrease this value in a highly mobile network: "[..] it might prove helpful to decrease the value in a highly mobile environment ([...]) but keep in mind that this will drastically increase the traffic. Unless you experience problems with your setup, it is suggested you keep the default value." As already mentioned, we used each routing protocol with the standard configuration. We can summarize that both B.A.T.M.A.N. implementations react very fast to topology changes when the route is disconnected. OLSR is only faster than the others when a better route is added to the network (see 4.1.4). When AODV worked, it performed very well and beat the pro-active routing protocols, but unluckily it didn't work all the time, especially for bigger network with more than 4 nodes.

Routing packets and data overhead The *HELLO* messages mentioned before also affect the routing packets and data overhead. We see that B.A.T.M.A.N. adds most and OLSR least packages. This is proportional to the *HELLO* packets inveral sent by the protocols. Here we can also see one benefit of B.A.T.M.A.N.-Advanced. Because the routing is happening on ISO/OSI layer 2, the packets added are very small, since they're missing the 40bytes layer 3 header.

Packet loss Astonishing is the huge difference of packet loss of B.A.T.M.A.N.-Advanced compared to the others. Our results are similar to those in [7], but contradict a newer paper [13]. In the latter they reach nearly no losses with all routing algorithms. Also here we should not forget that the virtual machines affect the measurements.

Chapter 7

Conclusion and Outlook

7.1 Conclusion

From our results, we see that the routing protocols distinctly vary in performance. The results contradict some recent works in the sense of that they determine that throughput is primarily decreased by the packets they add to the network. But also support some other work which also had almost as large packet losses as we have. The impact of using virtual machines for doing the measurements is also reflected in the results, for instance the huge round-trip time. But we've seen that using virtual machines is indeed a possiblity to compare routing algorithms, but there should be more investigation in how to make the impact smaller.
The conclusion from our measurements is that B.A.T.M.A.N.-Advanced is the most suitable for our highly mobile network. The convergence times are very low and the packet loss is nearly zero percent for all scenarios. This suites best for our mobile network we need in SWARMIX. Also in metrics throughput and round-trip time, B.A.T.M.A.N.-Advanced performs very good.

7.2 Outlook

With our thesis we showed that using virtual machines as nodes for a testbed is indeed a possibility, but the impact of the these virtual computers is noticeable. Measurements on how many virtual machnines can be running on one computer until bigger limitation appear should be investigated. Also the performance of the protocols with different network loads should be considered. Other routing algorithms like babel, which can be found many times in literature, should also be measured.

Chapter 8

Summary

In this semester thesis, we created a testbed consisting of multiple virtual machines running on 2 laptops to simulate the behaviour of flying drones in an aerial environment. We haven't found any relational work similar to ours in terms of using virtual machines as nodes. We used this testbed to run several scenarios, which can happen while these drones fly around, and used multiple tools to obtain metrics about the efficiency of the three routing algorithm B.A.T.M.A.N., B.A.T.M.A.N.-Advanced and OLSR. We also made some measurements with AODV, but it showed that it's not working well. From the results, we conclude that using B.A.T.M.A.N.-Advanced should be the best solution for our highly mobile network.

Appendix A

Installation and Configuration

A.1 Installation

We installed all routing protocol with the default configuration expect B.A.T.M.A.N.-Advanced. We set in the Makefile

```
CONFIG_BATMAN_ADV_DEBUG=y
```

to enable the debug option.
The versions used were:

Routing protocol	Version
B.A.T.M.A.N.	0.3.2
B.A.T.M.A.N.-Advanced	2012.4.0
OLSR	0.6.4
AODV	aodv-uu 0.9.6

A.2 Configuration

The configuration file for the Linksys AE1000 USB dongle looked like the following. We set the MCS to 1 for the reasons mentioned before and used channel 60 (5.3Ghz) for less interferences compared to the highly overused 2.4 Ghz band.

```
#The word of "Default" must not be removed
Default
CountryRegion=5
CountryRegionABand=7
CountryCode=CH
ChannelGeography=1
SSID=swarmix
NetworkType=Adhoc
WirelessMode=5
Channel=60
BeaconPeriod=100
TxPower=100
BGProtection=0
TxPreamble=0
RTSThreshold=2347
FragThreshold=2346
TxBurst=1
PktAggregate=0
WmmCapable=1
AckPolicy=0;0;0;0
AuthMode=OPEN
EncrypType=NONE
```

```
WPAPSK=
DefaultKeyID=1
Key1Type=0
Key1Str=
Key2Type=0
Key2Str=
Key3Type=0
Key3Str=
Key4Type=0
Key4Str=
PSMode=CAM
AutoRoaming=0
RoamThreshold=70
APSDCapable=0
APSDAC=0;0;0;0
HT_RDG=1
HT_EXTCHA=0
HT_OpMode=0
HT_MpduDensity=4
HT_BW=1
HT_BADecline=0
HT_AutoBA=1
HT_AMSDU=0
HT_BAWinSize=64
HT_GI=1
HT_MCS=1
HT_MIMOPSMode=3
HT_DisallowTKIP=1
HT_STBC=0
IEEE80211H=0
TGnWifiTest=0
WirelessEvent=0
CarrierDetect=0
AntDiversity=0
BeaconLostTime=4
PSP_XLINK_MODE=0
```

Appendix B

Scripts

Here are the scripts which have been used for doing the tests. Each node had a copy of the code folder from the repository. The nodes were set up to be reachable with ip adresses 192.168.2.x over the cabled network and have a customized config.sh. Then from the control unit (a computer where we control the nodes), the following commands have been executed.

1. ./setup_pca.sh # This sets up the command forwarding with netcat on all nodes
2. ./pca.sh 'source config.sh' # Load config on all nodes
3. ./pca.sh 'source functions.sh' # Load functions on all nodes
4. ./pca.sh './initial.sh' # Initializes all dongles on all nodes
5. ./pca.sh './create_nodes_list.sh' # Creates the nodes_list file with ip and mac
6. ./loop_scenarios.sh # Start doing all tests

Listing B.1: Sample configuration (config.sh)

```
1  #/bin/bash
2  # Configuration file for each node
3
4  export SWARMX_NR=1
5  #SWARMX_ROUTING='static'
6  export SWARMX_ROUTING=batman
7  #SWARMX_ROUTING='batman-adv'
8  #SWARMX_ROUTING=olsr
9
10 export SWARMX_SCENARIO=1
11
12 # This part automatically sets the DEVICE and ADAPTER
13
14 SWARMX_ADAPTER="NO"
15 SWARMX_DEVICE=""
16 if lsusb | grep -q TL-WN821N
17 then
18     SWARMX_ADAPTER="TPLINK"
19     SWARMX_DEVICE="wlan3"
20 fi
21
22 if lsusb | grep -q AE1000
23 then
24     SWARMX_ADAPTER="LINKSYS"
25     SWARMX_DEVICE="ra0"
26 fi
27
28 if lsusb | grep -q WUSB600N
29 then
30     SWARMX_ADAPTER="LINKSYS"
31     SWARMX_DEVICE="ra0"
32 fi
33
34 # Used for Linksys under 10.04nano c
35 if lsusb | grep -q Linksys
36 then
37     SWARMX_ADAPTER="LINKSYS"
38     SWARMX_DEVICE="ra0"
39 fi
40
41 export SWARMX_DEVICE
42 export SWARMX_ADAPTER
```

Listing B.2: Block a node (block.sh)

```
#!/bin/bash
# Block a node
source functions.sh

EXPECTED_ARGS=1
E_BADARGS=65

if [ $# -ne $EXPECTED_ARGS ]
then
  echo "Usage: ‘basename $0‘ #number_of_swarm_to_block"
  exit $E_BADARGS
fi

NR_OF_SWARM=$1

get_node_ip $1
get_node_mac $1

if [ "$NODE_MAC" = "" ]
then
        echo "MAC of swarm $1 not found, doing nothing"
        exit 0
fi

if [ "$SWARMX_ROUTING" = "batman-adv" ]
then
        echo "Blocking Swarm $1 with IP: $NODE_IP and MAC: $NODE_MAC"
        sudo ebtables -A INPUT -s $NODE_MAC -j DROP
elif [ "$SWARMX_ROUTING" = "babeld" ]
then
        sudo iptables -A INPUT -m mac --mac-source $NODE_MAC -i
            $SWARMX_DEVICE -j DROP
        sudo ip6tables -A INPUT -m mac --mac-source $NODE_MAC -i
            $SWARMX_DEVICE -j DROP
else
        echo "Blocking Swarm $1 with IP: $NODE_IP and MAC: $NODE_MAC"
        sudo iptables -A INPUT -m mac --mac-source $NODE_MAC -i
            $SWARMX_DEVICE -j DROP
        sudo arptables -A INPUT -s $NODE_IP -j DROP
        sudo arptables -A INPUT --source-mac $NODE_MAC -j DROP
        sudo arp -d $NODE_IP
fi
```

Listing B.3: Block all nodes (block_all.sh)

```
#!/bin/bash
# Blocks all other nodes
source functions.sh



for i in {1..5};
do
    if [ $i != $SWARMX_NR ];
    then
        ./block.sh $i
    fi
done
```

Listing B.4: Unblock a node (unblock.sh)

```
1  #!/bin/bash
2  source functions.sh
3
4  # Unblocks a specifig node
5
6  EXPECTED_ARGS=1
7  E_BADARGS=65
8
9  if [ $# -ne $EXPECTED_ARGS ]
10 then
11   echo "Usage: ‘basename $0‘ #number_of_swarm_to_unblock"
12   exit $E_BADARGS
13 fi
14
15 NR_OF_SWARM=$1
16
17 get_node_ip $1
18 get_node_mac $1
19
20 if [ "$NODE_MAC" = "" ]
21 then
22     echo "MAC of swarm $1 not found"
23     exit 0
24 fi
25
26 if [ "$SWARMX_ROUTING" = "batman-adv" ]
27 then
28     IP=192.168.101.$(($1*10))
29     echo "Unblocking Swarm $1 with IP: $NODE_IP and MAC: $NODE_MAC"
30     sudo ebtables -D INPUT -s $NODE_MAC -j DROP
31 elif [ "$SWARMX_ROUTING" = "babeld" ]
32 then
33     sudo iptables -D INPUT -m mac --mac-source $NODE_MAC -i
           $SWARMX_DEVICE -j DROP
34     sudo ip6tables -D INPUT -m mac --mac-source $NODE_MAC -i
           $SWARMX_DEVICE -j DROP
35 else
36     echo "Unblocking Swarm $1 with IP: $NODE_IP and MAC: $NODE_MAC"
37     sudo iptables -D INPUT -m mac --mac-source $NODE_MAC -i
           $SWARMX_DEVICE -j DROP
38     sudo arptables -D INPUT -s $NODE_IP -j DROP
39     sudo arptables -D INPUT --source-mac $NODE_MAC -j DROP
40 fi
```

Listing B.5: Unblock all nodes (unblock_all.sh)

```
1  #!/bin/bash
2
3  # Unblocks all nodes
4
5  sudo iptables -F
6  sudo ip6tables -F
7  sudo arptables -F
8  sudo ebtables -F
```

Listing B.6: Generic functions (functions.sh)

```
1  #!/usr/bin/bash
2
3  # Generic functions used by the scripts
4
5  # Sets NODE_IP to the IP for the node given in $1
6  function get_node_ip() {
7      # Sets NODE_IP to the ip of the remote node
8      if [ $SWARMX_ROUTING = 'batman-adv' ];
9      then
10         NODE_IP=192.168.101.$(($1*10))
11     else
12         NODE_IP=192.168.100.$(($1*10))
13     fi
14 }
15
16 # Sets NODE_MAC to the mac of the node $1
17 # Needs the nodes_list file created with
18 # create_nodes_list.sh
19 function get_node_mac() {
20     # Returns the MAC of a given node
21     temp=192.168.100.$(($1*10))
22     NODE_MAC=$(cat nodes_list | grep $temp | cut -c 16-)
23 }
24
25 # Starts the iperf client to node 5
26 function start_iperf_client() {
27     # Starts udp stream to server
28     get_node_ip 5
29     echo "Starting speed test" | ./predate.sh >> $SWARMX_LOGFILE &
30     iperf -c $NODE_IP -t 60 -l 1400 -i 10 -u -b 100M >>
           $SWARMX_LOGFILE &
31 }
32
33 # Stops the iperf client
34 function stop_iperf_client() {
35     # Kills all iperf processes
36     pkill iperf
37 }
38
39 # Starts the iperf server in udp mode
40 function start_iperf_server() {
41     # Starts iperf in listening mode on server side
42     # Logging is done on client side
43     iperf -s -u >> $SWARMX_LOGFILE &
44 }
45
46 # quits the iperf server
47 function stop_iperf_server() {
48     # Kills all iperf processes
49     pkill iperf
50 }
51
52 # Starts tcpstat with the correct filter
53 function start_tcpstat() {
54     if [ $SWARMX_ROUTING = 'batman' ];
55     then
```

```
56          sudo tcpstat -i ra0 -F -o 'TCPSTAT: Packets: %n, Bytes: %N\n'
                -f 'net 192.168.100.0/24 and udp dst port 4305' 10 >>
               $SWARMX_LOGFILE &
57      elif [ $SWARMX_ROUTING = "batman-adv" ]
58      then
59          # For batman-adv, we use the integrated statistics tool
60          # sudo batctl td -x 128 ra0 >> $SWARMX_LOGFILE &
61          echo 'Not starting tcpstat for batman-adv'
62      elif [ $SWARMX_ROUTING = "olsr" ]
63      then
64          sudo tcpstat -i ra0 -F -o 'TCPSTAT: Packets: %n, Bytes: %N\n'
                -f 'net 192.168.100.0/24 and udp dst port 698' 10 >>
               $SWARMX_LOGFILE &
65      elif [ $SWARMX_ROUTING = "aodv" ]
66      then
67          sudo tcpstat -i ra0 -F -o 'TCPSTAT: Packets: %n, Bytes: %N\n'
                -f 'udp dst port 654' 10 >> $SWARMX_LOGFILE &
68      elif [ $SWARMX_ROUTING = "babeld" ]
69      then
70          sudo tcpstat -i ra0 -F -o 'TCPSTAT: Packets: %n, Bytes: %N\n'
                -f 'host ff02::1:6' 10 >> $SWARMX_LOGFILE &
71      else
72          echo "TCPstat not starting"
73      fi
74  }
75
76  # Stops tcpstat
77  function stop_tcpstat() {
78      if [ $SWARMX_ROUTING = 'batman-adv' ];
79      then
80          # Write statistics to logfile
81          sudo batctl s | ./predate.sh >> $SWARMX_LOGFILE &
82      else
83          sudo pkill tcpstat
84      fi
85  }
86
87  # Starts a ping test to node 5
88  function start_ping_test() {
89      # Starts pinging node 5 for 60sec
90      get_node_ip 5
91      echo "Starting ping test" | ./predate.sh >> $SWARMX_LOGFILE &
92      sudo ping -i 0.5 -c 60 $NODE_IP | ./predate.sh >> $SWARMX_LOGFILE
            &
93  }
```

Listing B.7: Set date before the output (predate.sh)

```
1  #!/bin/bash
2
3  # Adds the current timestamp before an output
4  # Can be used with e.g.:
5  # echo "OK" | ./predate.sh > file.txt
6
7  while read -r line ; do
8      echo "$(date +%s.%N): ${line}"
9  done
```

Listing B.8: Initialize node (initial.sh)

```
1  #!/bin/bash
2
3  # Initializes the adapter, sets the correct IP etc
4  # as specified in config.sh
5
6  source config.sh
7
8  IP=192.168.100.$(($SWARMX_NR*10))
9
10 echo $SWARMX_ADAPTER "detected. Using device" $SWARMX_DEVICE
11
12 if [ $SWARMX_ADAPTER = "LINKSYS" ]
13 then
14         if lsmod | grep rt3572sta
15         then
16                 echo "Unloading module rt3572sta"
17                 sudo rmmod rt3572sta
18         fi
19         echo "Loading module rt3572sta"
20         sudo modprobe rt3572sta
21         echo "Waiting 3sec"
22         sleep 3
23         echo "Setting ra0 down"
24         sudo ifconfig ra0 down
25         echo "Waiting 3sec"
26         sleep 3
27         echo "Setting IP to: $IP"
28         sudo ifconfig ra0 $IP up
29 elif [ $SWARMX_ADAPTER = "TPLINK" ]
30 then
31         echo "Unloading module ath9k_htc"
32         sudo modprobe -r ath9k_htc
33         echo "Waiting 3sec"
34         sleep 3
35         echo "Loading module ath9k_htc"
36         sudo modprobe ath9k_htc
37         echo "Waiting 3sec"
38         sleep 3
39         sudo ifconfig $SWARMX_DEVICE down
40         sudo iwconfig $SWARMX_DEVICE mode ad-hoc
41         sudo ifconfig $SWARMX_DEVICE $IP up
42         sudo iwconfig $SWARMX_DEVICE essid swarmx
43         sudo iwconfig $SWARMX_DEVICE channel $SWARMX_CHANNEL
44 fi
45
46 echo "$SWARMX_DEVICE initialized"
47
48 echo "Settings system time"
49 sudo ntpdate 0.ch.pool.ntp.org
50 sudo ntpdate 0.ch.pool.ntp.org
51
52 exit 0
```

Listing B.9: Uninitialize node (uninitial.sh)

```
1  #!/bin/bash
2
3  #This script shutdowns the ra0 interface and removes the modules from
       the kernel
4
5  if [ $SWARMX_DEVICE = "ra0" ]
6  then
7          # shutdown interface linksys
8          sudo ifconfig ra0 down
9          sudo rmmod rt3572sta
10
11 elif [ $SWARMX_DEVICE = 'wlan3' ]
12 then
13     sudo ifconfig wlan3 down
14     sudo modprobe -r ath9k_htc
15 fi
```

Listing B.10: Create the nodes_list file (create_nodes_list.sh)

```
1  #/bin/bash
2  # Pings each node ip and creates the nodes_list file
3
4  for i in {1..9}; do
5          ping -c 1 -W 2 192.168.100.$((i*10)) > /dev/null &
6  done
7  arp -n | grep 192.168.100. | grep -v incomplete | cut -c 1-14,33-51 >
       nodes_list
8  sleep 2
9  date
10 cat nodes_list
```

Listing B.11: Setup the nodes to accept commands (setup_pca.sh)

```
#!/bin/bash

# Scripts initializes the netcat bash forwarding on all nodes for
# faster command forwarding than ssh
# Assumes that nodes are reachable over ip's 192.168.2.x

for i in 10 20 30 40 50;
do
    ssh ubuntu@192.168.2.$i 'cd ~/Semesterarbeit/code; mkfifo pipe;
        nc -l -k 1337 <pipe | /bin/bash &>pipe &'
done
```

Listing B.12: Push a command to a node (pc.sh)

```
#!/bin/bash

# Push a command to the node specified in $1
# command is in $2

# e.g. to block node 3 from 2:
# ./pc.sj 2 './block.sh 3'

# ssh ubuntu@192.168.2.$(($1*10)) 'cd ~/Semesterarbeit/code;' $2
echo $2 | nc 192.168.2.$(($1*10)) 1337
```

Listing B.13: Push a command to all nodes (pca.sh)

```
#!/bin/bash

# Push a command to all nodes

for i in 10 20 30 40 50;
do
    echo $1 | nc 192.168.2.$i 1337
done
```

Listing B.14: The scenarios and timings (scenarios.sh)

```
1  #!/bin/bash
2
3  # This script represents the scenarios mentioned in the thesis
4  # Scripts must be run on control unit
5
6  source functions.sh
7
8  SCENARIO=$1
9
10 ./pca.sh 'source functions.sh'
11 ./pca.sh "export SWARMX_SCENARIO=$SCENARIO"
12 ./pca.sh 'mkdir -p ../results/scenarios/$SWARMX_SCENARIO'
13 ./pca.sh 'export SWARMX_LOGFILE=../results/scenarios/$SWARMX_SCENARIO
       /$SWARMX_NR-$SWARMX_ROUTING.log'
14
15 echo "Scenario $1"
16 echo "Starting Routing on all nodes"
17 ./pca.sh './unblock_all.sh'
18 ./pca.sh 'echo "----NEW----" | ./predate.sh >> $SWARMX_LOGFILE'
19 ./pca.sh './start.sh'
20 # Starting iperf Server on node 5
21 ./pc.sh 5 'start_iperf_server'
22
23 # Starting byte monitoring on all nodes
24 ./pca.sh 'start_tcpstat'
25
26 echo "Waiting 30 Seconds"
27 sleep 30
28
29 if [ $SCENARIO == 1 ];
30 then
31     ./pc.sh 1 './block.sh 3'
32     ./pc.sh 1 './block.sh 4'
33     ./pc.sh 1 './block.sh 5'
34
35     ./pc.sh 2 './block.sh 3'
36     ./pc.sh 2 './block.sh 4'
37
38     ./pc.sh 3 './block_all.sh'
39
40     ./pc.sh 4 './block.sh 1'
41     ./pc.sh 4 './block.sh 2'
42     ./pc.sh 4 './block.sh 3'
43
44     ./pc.sh 5 './block.sh 1'
45     ./pc.sh 5 './block.sh 3'
46
47     echo "Waiting 60sec"
48     sleep 60
49
50     echo "Begin pinging node 5 from node 1"
51     ./pc.sh 1 'get_node_ip 5'
52     ./pc.sh 1 'echo "Begin pinging" | ./predate.sh >> $SWARMX_LOGFILE
           '
53     ./pc.sh 1 'sudo ping -i 0.1 -c 400 $NODE_IP | ./predate.sh >>
         $SWARMX_LOGFILE &'
54
```

```
55      echo "Waiting 20sec"
56      sleep 20
57
58
59      echo "Node moving"
60      ./pca.sh 'echo "Node Moving" | ./predate.sh >> $SWARMX_LOGFILE'
61      ./pc.sh 1 './unblock.sh 4'
62      ./pc.sh 1 './block.sh 2'
63
64      ./pc.sh 2 './block.sh 1'
65
66      ./pc.sh 4 './unblock.sh 1'
67
68      echo "Waiting another 60sec to settle"
69      sleep 60
70
71      echo "Doing speed test"
72      # Starting performance measurement
73      ./pc.sh 1 'start_iperf_client'
74
75      sleep 65
76
77      echo "Doing ping test"
78      ./pc.sh 1 'start_ping_test'
79
80      sleep 65
81
82
83  elif [ $SCENARIO == 2 ]
84  then
85      ./pc.sh 3 './block_all.sh'
86
87      ./pc.sh 4 './block_all.sh'
88
89      ./pc.sh 1 './block.sh 3'
90      ./pc.sh 1 './block.sh 4'
91      ./pc.sh 1 './block.sh 5'
92
93      ./pc.sh 2 './block.sh 3'
94      ./pc.sh 2 './block.sh 4'
95      ./pc.sh 2 './block.sh 5'
96
97      ./pc.sh 5 './block.sh 1'
98      ./pc.sh 5 './block.sh 2'
99      ./pc.sh 5 './block.sh 3'
100     ./pc.sh 5 './block.sh 4'
101
102     echo "Waiting 60sec"
103     sleep 60
104
105     echo "Begin pinging node 5 from node 1"
106     ./pc.sh 1 'get_node_ip 5'
107     ./pc.sh 1 'echo "Begin pinging" | ./predate.sh >> $SWARMX_LOGFILE
            '
108     ./pc.sh 1 'sudo ping -i 0.1 -c 400 $NODE_IP | ./predate.sh >>
            $SWARMX_LOGFILE &'
109
110     echo "Waiting 20sec"
```

```
111     sleep 20
112
113     echo "Node moving"
114     ./pca.sh 'echo "Node Moving" | ./predate.sh >> $SWARMX_LOGFILE'
115     ./pc.sh 2 './unblock.sh 5'
116
117     ./pc.sh 5 './unblock.sh 2'
118
119     echo "Waiting another 60sec to settle"
120     sleep 60
121
122     echo "Doing speed test"
123     # Starting performance measurement
124     ./pc.sh 1 'start_iperf_client'
125
126     sleep 65
127
128     echo "Doing ping test"
129     ./pc.sh 1 'start_ping_test'
130
131     sleep 65
132
133 elif [ $SCENARIO == 3 ]
134 then
135     ./pc.sh 1 './block.sh 4'
136     ./pc.sh 1 './block.sh 5'
137
138     ./pc.sh 2 './block.sh 3'
139
140     ./pc.sh 3 './block.sh 2'
141     ./pc.sh 3 './block.sh 5'
142
143     ./pc.sh 4 './block.sh 1'
144
145     ./pc.sh 5 './block.sh 1'
146     ./pc.sh 5 './block.sh 3'
147
148     echo "Waiting 60sec"
149     sleep 60
150
151     echo "Begin pinging node 5 from node 1"
152     ./pc.sh 1 'get_node_ip 5'
153     ./pc.sh 1 'echo "Begin pinging" | ./predate.sh >> $SWARMX_LOGFILE
            '
154     ./pc.sh 1 'sudo ping -i 0.1 -c 400 $NODE_IP | ./predate.sh >>
          $SWARMX_LOGFILE &'
155
156     echo "Waiting 20sec"
157     sleep 20
158
159     echo "Node moving"
160     ./pca.sh 'echo "Node Moving" | ./predate.sh >> $SWARMX_LOGFILE'
161
162     ./pc.sh 1 './block.sh 2'
163
164     ./pc.sh 2 './block.sh 1'
165     ./pc.sh 2 './block.sh 4'
166     ./pc.sh 2 './block.sh 5'
```

```
167
168     ./pc.sh 4 './block.sh 2'
169
170     ./pc.sh 5 './block.sh 2'
171
172     echo "Waiting another 60sec to settle"
173     sleep 60
174
175     echo "Doing speed test"
176     # Starting performance measurement
177     ./pc.sh 1 'start_iperf_client'
178
179     sleep 65
180
181     echo "Doing ping test"
182     ./pc.sh 1 'start_ping_test'
183
184     sleep 65
185
186 elif [ $SCENARIO == 4 ]
187 then
188     ./pc.sh 1 './block.sh 2'
189     ./pc.sh 1 './block.sh 4'
190     ./pc.sh 1 './block.sh 5'
191
192     ./pc.sh 2 './block.sh 1'
193     ./pc.sh 2 './block.sh 3'
194     ./pc.sh 2 './block.sh 4'
195     ./pc.sh 2 './block.sh 5'
196
197     ./pc.sh 3 './block.sh 2'
198     ./pc.sh 3 './block.sh 5'
199
200     ./pc.sh 4 './block.sh 1'
201     ./pc.sh 4 './block.sh 2'
202
203     ./pc.sh 5 './block.sh 1'
204     ./pc.sh 5 './block.sh 2'
205     ./pc.sh 5 './block.sh 3'
206
207     echo "Waiting 60sec"
208     sleep 60
209
210     echo "Begin pinging node 5 from node 1"
211     ./pc.sh 1 'get_node_ip 5'
212     ./pc.sh 1 'echo "Begin pinging" | ./predate.sh >> $SWARMX_LOGFILE
        '
213     ./pc.sh 1 'sudo ping -i 0.1 -c 400 $NODE_IP | ./predate.sh >>
          $SWARMX_LOGFILE &'
214
215     echo "Waiting 20sec"
216     sleep 20
217
218     echo "Node moving"
219     ./pca.sh 'echo "Node Moving" | ./predate.sh >> $SWARMX_LOGFILE'
220
221     ./pc.sh 1 './unblock.sh 2'
222
```

```
223     ./pc.sh 2 './unblock.sh 1'
224     ./pc.sh 2 './unblock.sh 4'
225     ./pc.sh 2 './unblock.sh 5'
226
227     ./pc.sh 4 './unblock.sh 2'
228
229     ./pc.sh 5 './unblock.sh 2'
230
231     echo "Waiting another 60sec to settle"
232     sleep 60
233
234     echo "Doing speed test"
235     # Starting performance measurement
236     ./pc.sh 1 'start_iperf_client'
237
238     sleep 65
239
240     echo "Doing ping test"
241     ./pc.sh 1 'start_ping_test'
242
243     sleep 65
244
245 elif [ $SCENARIO == 5 ]
246 then
247     ./pc.sh 1 './block.sh 2'
248     ./pc.sh 1 './block.sh 3'
249     ./pc.sh 1 './block.sh 4'
250
251     ./pc.sh 2 './block.sh 1'
252     ./pc.sh 2 './block.sh 4'
253     ./pc.sh 2 './block.sh 5'
254
255     ./pc.sh 3 './block.sh 1'
256     ./pc.sh 3 './block.sh 5'
257
258     ./pc.sh 4 './block.sh 1'
259     ./pc.sh 4 './block.sh 2'
260
261     ./pc.sh 5 './block.sh 2'
262     ./pc.sh 5 './block.sh 3'
263
264     echo "Waiting 60sec"
265     sleep 60
266
267     echo "Begin pinging node 5 from node 1"
268     ./pc.sh 1 'get_node_ip 5'
269     ./pc.sh 1 'echo "Begin pinging" | ./predate.sh >> $SWARMX_LOGFILE
            '
270     ./pc.sh 1 'sudo ping -i 0.1 -c 400 $NODE_IP | ./predate.sh >>
          $SWARMX_LOGFILE &'
271
272     echo "Waiting 20sec"
273     sleep 20
274
275     echo "Node moving"
276     ./pca.sh 'echo "Node Moving" | ./predate.sh >> $SWARMX_LOGFILE'
277
278     ./pc.sh 1 './unblock.sh 2'
```

```
279     ./pc.sh 1 './block.sh 5'
280
281     ./pc.sh 2 './unblock.sh 1'
282
283     ./pc.sh 5 './block.sh 1'
284
285     echo "Waiting another 60sec to settle"
286     sleep 60
287
288     echo "Doing speed test"
289     # Starting performance measurement (60s)
290     ./pc.sh 1 'start_iperf_client'
291
292     sleep 65
293
294     echo "Doing ping test"
295     ./pc.sh 1 'start_ping_test'
296
297     sleep 65
298
299 else
300     echo "Not implemented"
301 fi
302
303 ./pca.sh 'stop_tcpstat'
304 ./pc.sh 1 'stop_iperf_client'
305 ./pc.sh 5 'stop_iperf_server'
306
307
308 ./pca.sh './unblock_all.sh'
309 echo "Stopping Routing"
310 ./pca.sh './stop.sh'
311
312 echo "Scenario $SCENARIO done"
```

Listing B.15: Loop over the scenarios (loop_scenarios.sh)

```
 1 #!/bin/bash
 2
 3 # Loops through all scenarios and does all tests
 4 # This script has to be run on the control unit
 5
 6 # Do 10 iterations of 5 scenarios
 7 for q in {1..10}
 8 do
 9     for s in 1 2 3 4 5
10     do
11         for r in batman batman-adv olsr
12         do
13             echo "Using $r and doing Scenario $s"
14             ./pca.sh "SWARMX_ROUTING=$r"
15             sleep 1
16             ./scenarios.sh $s
17             sleep 1
18         done
19     done
20     echo "Full Loop $q"
21 done
```

Listing B.16: Start routing algorithm (start.sh)

```
1  #!/bin/bash
2
3  # Starts the routing algorithms on each the current node
4  # config.sh has to be loaded before with 'source config.sh'
5
6  source functions.sh
7
8  if [ $SWARMX_ROUTING = "olsr" ]
9  then
10         echo "Starting␣OLSRD" | ./predate.sh >> $SWARMX_LOGFILE
11         sudo olsrd -i $SWARMX_DEVICE -d 2 | grep --line-buffered KERN
               | ./predate.sh >> $SWARMX_LOGFILE &
12 elif [ $SWARMX_ROUTING = "batman-adv" ]
13 then
14         # We need to setup a bridge else we cannot filter the packets
               on layer2 with ebtables
15         echo "Starting␣Batman-Adv" | ./predate.sh >> $SWARMX_LOGFILE
16         sudo modprobe batman-adv
17
18         sudo brctl addbr meshx
19         sudo brctl addif meshx $SWARMX_DEVICE
20
21         sudo ip link set meshx up
22
23         sudo batctl if add meshx
24
25         NEW_IP=192.168.101.$(($SWARMX_NR*10))
26
27         sudo ifconfig bat0 $NEW_IP up
28         sudo ifconfig bat0 mtu 1472
29
30         sudo batctl ll routes
31
32         sudo cat /sys/kernel/debug/batman_adv/bat0/log | ./predate.sh
               >> $SWARMX_LOGFILE &
33
34 elif [ $SWARMX_ROUTING = "batman" ]
35 then
36         echo "Starting␣batman" | ./predate.sh >> $SWARMX_LOGFILE
37         sudo batmand -d 3 $SWARMX_DEVICE 2>&1 | ./predate.sh >>
              $SWARMX_LOGFILE &
38
39 elif [ $SWARMX_ROUTING = "bmx6" ]
40 then
41         echo "Starting␣bmx6" | ./predate.sh >> $SWARMX_LOGFILE
42         sudo bmx6 dev=ra0
43
44 elif [ $SWARMX_ROUTING = "babeld" ]
45 then
46         echo "Starting␣babeld" | ./predate.sh >> $SWARMX_LOGFILE
47         sudo babeld -d 1 $SWARMX_DEVICE | ./predate.sh >>
              $SWARMX_LOGFILE &
48
49 elif [ $SWARMX_ROUTING = "aodv" ]
50 then
51         echo "Starting␣aodv" | ./predate.sh >> $SWARMX_LOGFILE
```

```
52          sudo aodvd -i $SWARMX_DEVICE | ./predate.sh >>
                $SWARMX_LOGFILE &
53  else
54          echo "Routing not found"
55
56  fi
```

Listing B.17: Stop routing algorithm (stop.sh)

```
 1  #!/bin/bash
 2  source functions.sh
 3
 4  # Stops the current routing protocol
 5
 6  echo "Stopping $SWARMX_ROUTING" | ./predate.sh >> $SWARMX_LOGFILE
 7
 8  if [ $SWARMX_ROUTING = "olsr" ]
 9  then
10      echo "Stopping OLSRD"
11      sudo pkill olsrd
12  elif [ $SWARMX_ROUTING = "batman-adv" ]
13  then
14      # We need to setup a bridge else we cannot filter the packets on
            layer2 with ebtables
15      echo "Stopping Batman-Adv"
16      sudo pkill cat
17      sudo batctl if del meshx
18      sudo ip link set meshx down
19      sudo brctl delif meshx $SWARMX_DEVICE
20      sudo brctl delbr meshx
21      # sudo ifconfig bat0 mtu 1500
22      sudo modprobe -r batman_adv
23
24  elif [ $SWARMX_ROUTING = "batman" ]
25  then
26      echo "Stopping batman"
27      sudo pkill batmand
28
29  elif [ $SWARMX_ROUTING = "bmx6" ]
30  then
31      echo "Stopping bmx6"
32      sudo pkill bmx6
33
34  elif [ $SWARMX_ROUTING = "babeld" ]
35  then
36      echo "Stopping babeld"
37      sudo pkill babeld
38
39  elif [ $SWARMX_ROUTING = "aodv" ]
40  then
41      echo "Stopping aodv"
42      sudo pkill aodvd
43  fi
```

Appendix C

Timetable

Real-World Evaluation of Ad Hoc Routing Algorithms

Task \Week	01	02	03	04	05	06	07	08	09	10	11	12	13	14
Literature review on different dynamic routing algorithms for wireless ad-hoc and evaluation metrics														
Preparing the workspace by installing necessary tools such as vmware, iptables, arptables, iperf, wireless-tools														
Getting familiar with network analysis tools, such as iperf, iw/iwconfig, iwpriv, traceroute, ping etc														
Installing the Linksys and tplink wireless cards and setting up ad-hoc connection between two laptops														
Setting up ad-hoc connection between two virtual machines														
Selecting and installing dynamic routing algorithms such as AODV, OLSR, BATMAN etc														
Drawing multiple scenarios with various topologies having well-written scripts														
Performing the scenarios and collecting results														
Writing the report														
Preparing a 15 min presentation														

Appendix D

Originalproblem

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

FNSNF
Schweizerischer Nationalfonds zur Förderung der wissenschaftlichen Forschung

Real-World Evaluation of Ad Hoc Routing Algorithms

Master or semester thesis for a student in department D-ITET/D-INFK

Rescue missions require timely and flexible communications operating even in lack of infrastructure networks. In the SWARMIX project, we investigate the interactions of heterogeneous agents on a search and rescue mission. A swarm comprises rescue professionals, dogs, and UAVs (Unmanned Aerial Vehicles) cooperating to find a victim as fast as possible. Communication comprises images, voice recordings, positions, and other sensor data sent from each agent back to the ground station via an ad hoc network (see Figure 1).

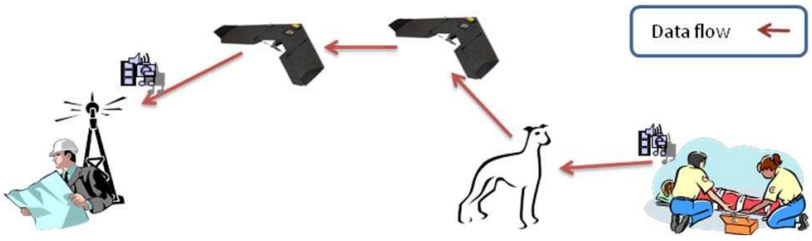


Figure 1. Receiving data wirelessly at a ground station.

As all agents move in unpredictable patterns, nodes that are within transmission range may soon move out of range. Dynamic routing algorithms such as AODV (Ad-hoc On Demand Vector routing) and OLSR (Optimized Link-State Routing) are designed to tackle such challenges. (See Figure 2 for an example of a route changing due to disconnections.)

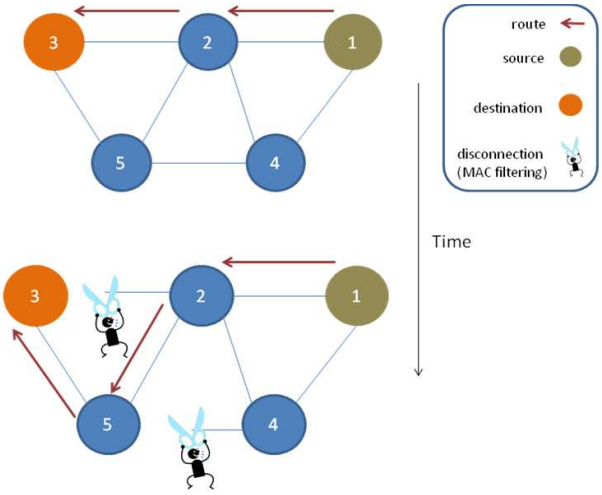


Figure 2. Discovering new route upon disconnections.

The goal of this thesis is to study and measure performance of selected standard ad hoc routing algorithms in a testbed comprising multiple (stationary) laptops. To emulate disconnections, we use MAC filtering to temporarily block wireless frames exchanged between certain pairs of nodes. This enables studying how well a dynamic routing algorithm responds under a variety of circumstances, and also estimating the overhead imposed by re-routing which can be used to improve routing algorithm approaches. We propose different test scenarios and measure delay, number of disconnections, achieved throughput, signal quality etc.

Kind of Work: 70% practical, 30% theory

Requirements: Linux experience (network configuration, scripting), routing basics

Contact Persons:

Mahdi Asadpour, mahdi.asadpour@tik.ee.ethz.ch, ETZ G96, +41 44 63 27539

Dr. Domenico Giustiniano, domenico.giustiniano@tik.ee.ethz.ch, ETZ G60.1, +41446327461

Professor: Prof. Dr. Bernhard Plattner

Bibliography

[1] IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput. *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pages 1–565, 2009.

[2] iperf homepage. `http://sourceforge.net/projects/iperf/`, March 2013.

[3] Linux wireless drivers. `http://linuxwireless.org/`, March 2013.

[4] Netcat homepage. `http://netcat.sourceforge.net/`, March 2013.

[5] pcap-filter linux man page. `http://www.manpagez.com/man/7/pcap-filter/`, March 2013.

[6] OLSR homepage. `http://www.olsr.org/`, March 2013.

[7] M. Abolhasan, B. Hagelstein, and J.C.-P. Wang. Real-world performance of current proactive multi-hop mesh protocols. In *Communications, 2009. APCC 2009. 15th Asia-Pacific Conference on*, pages 44 –47, oct. 2009.

[8] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '98, pages 85–97, New York, NY, USA, 1998. ACM.

[9] P. Chenna Reddy and P. ChandraSekhar Reddy. Performance analysis of adhoc network routing protocols. In *Ad Hoc and Ubiquitous Computing, 2006. ISAUHC '06. International Symposium on*, pages 186 –187, dec. 2006.

[10] S. Corson and J. Macker. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501 (Informational), January 1999.

[11] Förderverein Freie Netzwerke e.V. Freifunk homepage. `http://start.freifunk.net/`, mar 2013.

[12] Linksys. Linksys. `http://www.linksys.com/`, March 2013.

[13] D. Murray, M. Dixon, and T. Koziniec. An experimental comparison of routing protocols in multi hop ad hoc networks. In *Telecommunication Networks and Applications Conference (ATNAC), 2010 Australasian*, pages 159–164, 31 2010-Nov. 3.

[14] E Nordstrom, Per Gunningberg, Christian Rohner, and Oskar Wibling. A comprehensive comparison of manet routing protocols in simulation, emulation and the real world. *Uppsala University*, pages 1–12, 2006.

[15] open mesh. B.A.T.M.A.N. advanced homepage. `http://www.open-mesh.org/projects/batman-adv`, March 2013.

[16] open mesh. B.A.T.M.A.N. homepage. `http://www.open-mesh.org/projects/batmand`, March 2013.

[17] C.E. Perkins, E.M. Royer, S.R. Das, and M.K. Marina. Performance comparison of two on-demand routing protocols for ad hoc networks. *Personal Communications, IEEE*, 8(1):16 –28, feb 2001.

[18] C. Samara, E. Karapistoli, and A.A. Economides. Performance comparison of manet routing protocols based on real-life scenarios. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2012 4th International Congress on*, pages 870–877, 2012.

[19] Swiss National Science Foundation (SNSF). Synergistic interactions in swarms of heterogeneous agents. `http://www.swarmix.org/`, March 2013.

[20] Uppsala University. AODV homepage. `http://aodvuu.sourceforge.net/`, March 2013.