Autonomous networks without the need for infrastructure
A study of zero configuration mesh networks in Linux environments

Author: Anton Roskvist, Filip Roskvist, Jimmy Månsson
Supervisor: Patrik Brandt
Date: 2014-06-25
Course Code: 2DV00E, 15 credits
Level: Bachelor
Department of Computer Science
Abstract
Autonomous Mesh Networks potentially allows for cheaper networks, of use for impoverished areas with poor infrastructure and little interest from service providers for expansion. The subject of wireless mesh networks is interesting for several reasons. Non-reliance, or at the very least reduced reliance on existing infrastructure and service providers gives more control of a network to the users and their communities. These kinds of networks are however conceived to be quite complex to set up, manage and maintain. The goal of this paper was to create an autonomous network without any need for infrastructure, that was relatively easy to configure, use, and performs well. The implementation technique used succeeds at reaching these goals. The script and environment that was constructed makes it easy to set up and join nodes into the network, and the network can increase and decrease in size without affecting the core functionality of the network. The implementation for automatic host discovery makes it simple for anyone with a small amount of knowledge to find and communicate with other hosts, and the network has proven to be resilient to some common ways of tampering.

Keywords: Wireless mesh networks, Autonomous networks, B.A.T.M.A.N-Adv, zero configuration
# Contents

1 Introduction .................................................................................. 1
  1.1 Background ........................................................................... 1
  1.2 Previous Research .................................................................. 1
  1.3 Problem .................................................................................. 2
  1.4 Purpose .................................................................................. 2
  1.5 Ethical implications ................................................................. 2
  1.6 Demarcation .......................................................................... 3
  1.7 Target Audience ..................................................................... 3
  1.8 Report outline ......................................................................... 4

2 Theory .......................................................................................... 5
  2.1 Wireless Mesh Networks .......................................................... 5
     2.1.1 Controversies ................................................................... 6
     2.1.2 Mesh technologies .......................................................... 6
  2.2 Zero Configuration Networking ................................................ 7
  2.3 Hypervisor .............................................................................. 8
  2.4 B.A.T.M.A.N-advanced ........................................................... 8
  2.5 Ubuntu .................................................................................... 9

3 Method .......................................................................................... 10
  3.1 Scientific approach .................................................................. 10
  3.2 Environment ........................................................................... 10
  3.3 Tests ...................................................................................... 11
     3.3.1 Hardware Usage ............................................................. 11
     3.3.2 Response Time .............................................................. 12
     3.3.3 Redundancy .................................................................... 12
     3.3.4 Threats .......................................................................... 13
  3.4 Software .................................................................................. 14
  3.5 Implementation ........................................................................ 15
     3.5.1 Network performance ...................................................... 15
     3.5.2 Network security ............................................................. 18
  3.6 Reliability ............................................................................... 20

4 Results .......................................................................................... 21
  4.1 Script and network ................................................................... 21
  4.2 Test results ............................................................................. 22
     4.2.1 Network performance results ........................................... 22
     4.2.2 Security results ............................................................... 23

5 Discussion .................................................................................... 25
  5.1 Usability ................................................................................ 25
  5.2 Possible applications .............................................................. 25
  5.3 Conclusion ............................................................................. 26
  5.4 Future research ...................................................................... 26

References ....................................................................................... 28

A Scripts ........................................................................................... 1
  A.A Bat ......................................................................................... 1
  A.B Init ........................................................................................ 5

B Test results .................................................................................... 6
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.A. Hardware usage</td>
<td>6</td>
</tr>
<tr>
<td>B.B. Response time</td>
<td>7</td>
</tr>
<tr>
<td>B.C. Full mesh</td>
<td>9</td>
</tr>
<tr>
<td>B.D. Redundancy</td>
<td>9</td>
</tr>
</tbody>
</table>
1 Introduction

The subject of wireless mesh networks is interesting for several reasons. Non-reliance, or at the very least reduced reliance on existing infrastructure and service providers gives more control of a network to the users and their communities. It potentially allows for cheaper networks, of use for impoverished areas with poor infrastructure and little interest from service providers for expansion. For those who wish it, it can provide a greater level of anonymity.

The aim of this paper is to examine the possibility of creating a mesh network with the help of scripting that is to a high degree self-configuring and flexible. Achieving this, the aim is to test the usability of the network in the broadest sense, in terms of ease of use and performance. This is done in order to evaluate the potential of the network for future development and more widespread usage.

1.1 Background

The subject that this paper is based on originally came from an idea to create a network that could be used for social networking based on location rather than people you know. By having a network where anyone could join and automatically find anyone within the area of the network, as long as they are within sufficient range of others utilizing it, it could be used for a multitude of social purposes, without being limited to people you know.

A full scale implementation of this idea would go far beyond what would be possible with the time and resources available, and would also veer more into the territory of programming. The goal was therefore simplified to be more about proving the concept of such a network and potentially create further interest for the subject down the line.

1.2 Previous Research

In 2011 A. Quartulli and R. Lo Cigno conducted a research project focused on using the B.A.T.M.A.N-Adv routing protocol to study the client announcement- and the roaming mechanisms in regards to previous versions of the routing algorithm. They found a significant improvement of performance over earlier versions of B.A.T.M.A.N. Their focus was on finding an efficient strategy to announce clients that removes the current limitations as well as giving the bases on top of which it is possible to implement an efficient roaming procedure. [1]

On the general subject of wireless mesh networks, the Nepal Wireless Networking Project (NWNP) is something that is of great importance. In the 1990s Mahabir Pun returned to Nepal from his studies in the U.S. Tired of having to walk for hours every day to check his email account, he started a project to bring the internet to his home village. The project expanded and in
2011 the NWNP supplied 150 Nepali villages with a basic internet connection. This is a prime example of what can be achieved through the usage of mesh networks in areas that are lacking in existing networking infrastructure. [2]

1.3 Problem
The problem defined in this paper is to find how it can be made possible to create an autonomous network, without the need for any underlying infrastructure, which has the flexibility to seamlessly increase or decrease in size depending on the number currently active nodes through the use of an ad hoc routing protocol and furthermore to determine how useful or practical such a network can be. With useful and practical being defined by the ease of distribution, usage and the scalability of performance in the network in terms of transfer speed, response time, convergence time and hardware usage.

This paper defines an autonomous network as a network that, although it can connect to other networks can exists and fully function on its own accord. Underlying infrastructure is defined as any access point, switch, router or other type of networking node. While the network can use such nodes it should be capable of functioning without them.

1.4 Purpose
The intention of this thesis is to find a way to construct an autonomous network without a need for underlying infrastructure and to minimize the amount of work necessary to set up such a network. Achieving this, the purpose becomes to test the performance of the network.

The idea is that, because there are no dedicated networking nodes available it becomes the job of all the nodes involved in the network to make sure that any node can reach any other node. This potentially means that there can be issues of performance where units with limited hardware become overworked, or the network may be get congested when multiple nodes are transferring data between each other. With a network that has the possibility of constantly changing there is also a question of how well the routing protocol can adapt to frequently changing routes through the network. All these questions are intended to be answered in order to thereby determine how well the proposed network can function.

1.5 Ethical implications
The type of network proposed has some potential ethical ramifications worth mentioning. One of the functions of the network is the ability to extend the reach of a single Internet connection over distances far greater than it would normally do, enabling a single connection to be shared by several network participants. This is something that could be of great use for impoverished areas without the resources necessary to establish a traditional infrastructure.
However, it could also be something that may not be permitted by the Internet service provider, depending on what the Internet service provider’s end user license agreement states. Obviously, such conditions must be reviewed before sharing an Internet connection.

The ability to use the network anonymously may also be something worth noting. Being anonymous in itself is not illegal, but anonymity may aid illegal activity by leaving the perpetrator hidden from scrutiny. While this may be a bad thing anonymity is also important as a tool to help with communication, especially for areas of the world where concerns for censorship and freedom of speech are present.

1.6 Demarcation

There is a multitude of routing protocols for mesh networks, following several different design principles. [8] Due to time constraints a single protocol was used to construct and test the network used in this paper. The B.A.T.M.A.N-Adv protocol was chosen as it is filled all the needs of the network and is a protocol that is still actively being worked on. [24] The environment was likewise narrowed down to only include Unix-like systems as it would have required an amount of work far beyond the scope for the paper to enable cross platform usage of the routing protocol for a system like Windows.

While real hardware is used to the extent that it is possible, and to ensure that it is possible to implement the routing protocol on the required hardware, the majority of the nodes in the network used for this paper are virtual machines. Due to a lack of resources this was the only way to achieve a network of a reasonable size for testing purposes.

A question that was considered but left out is the scalability of size in the network, both with mobile devices on their own as well as with the help of higher range static access points. This is an important question when considering the real world implementation of the type of network used in this paper, but one that exceeds its reach. While limited tests were conducted, essentially ensuring that two nodes unable to individually reach each other could indeed communicate through a third node placed between them, a significant amount of additional resources and manpower would have been necessary to properly carry out a test on a larger scale.

1.7 Target Audience

Although the idea behind this paper may at some point become interesting to the general population, at this stage it is considered to be primarily aimed at those with a prior knowledge and interest in networking. Given that a certain level of knowledge is assumed it will therefore not go into detail about the more trivial aspects of how a network is constructed and functions. Likewise,
ease of use is considered in the context of an audience with a reasonable amount of knowledge and not for the average end user.

1.8 Report outline
In section 2 the different technologies and concepts relevant to this paper are presented. Section 3 presents the scientific approach taken for the thesis and the testing environment that was designed for the tests, including the hardware and software that was used. Section 4 reviews the scripts that have been constructed to set up the network in terms of design and function as well as presenting the results gathered from all the performed tests on the network. Section 5 provides the discussion and analysis on the research and results of the paper.
2 Theory

This section presents the theoretical groundwork of the thesis as it pertains to technologies, concepts, applications and theories that were used during the research and implementation of the papers stated goals. It intends to give the reader a general level of knowledge that will help to understand the work presented in subsequent chapters.

2.1 Wireless Mesh Networks

A Wireless Mesh Network (WMN) is a communications network made up of multiple nodes connected in a mesh topology using wireless communications (WiFi) [1]. A mesh topology is here defined as a topology in which each node can act both as host and router, and all can contribute to the propagation of the network [9]. These WMNs consists of decentralized wireless networks, so called ad hoc networks, which are defined as networks that does not require central administration or preexisting infrastructure such as routers or WiFi access points (AP) [2]. Ad hoc networks can in practice be set up by anyone with sufficient knowledge and two or more available wireless devices, and being separated from any infrastructure means the network traffic will not pass through any centralized authority or organization.

WMNs automatically reconfigure themselves according to the availability and proximity of bandwidth which makes them resistant to disaster and other interference. Dynamic connections between nodes enable packets to use multiple routes to travel through the network, which makes these networks very robust, and makes them more resilient to failure the more devices that are added. [3]

There are two classes of WMNs nodes: mesh routers and mesh clients. The routers are in charge of the network backbone and are usually fixed equipment. These routers can also be connected to a larger network such as the Internet, and can provide access to these outside networks for the clients in the network. If this is the case, the router is called a mesh gateway. Mesh clients are usually end-user devices such as laptops, workstations, tablets or mobile phones. [1]
2.1.1 Controversies
Although there are many inherent advantages to WMNs and ad hoc networking they are technologies that still sees limited usage in most of the world. Primavera De Filippi, a researcher at the Berkman Center for Internet & Society at Harvard Law, sees a number of reasons for why mesh networking have not reached a greater level of popularity as of yet: [4]

Technical issues
There is a certain level of complexity in setting up, managing and maintaining a mesh network, which can be an obstacle in reaching widespread deployment. The issue of limited wireless range from APs and end user devices combined with issues of scaling in the available routing protocols results in a limited network coverage.

Perception
There is a general perception that mesh networks are used as emergency tools in wartimes or after natural disasters. If WMNs were to be used as a precautionary measure this could provide an alternate and more resilient network infrastructure.

Political issues
Although WMNs could support the government in providing internet connectivity to poor neighborhoods, mesh networks cannot by their very nature be easily monitored or regulated by third parties. Because of this, mesh networks are usually regarded by the state as a potential danger that could disrupt public order by providing a platform for criminal activities. Mesh networks are also seen as a ”threat” to ISPs and mobile operators. This is due to mesh networking clashing with these companies traditional business models based on pay-per-use and monthly bills.

2.1.2 Mesh technologies
There are several routing protocols that can be used in Wireless Mesh Networks. This section will provide a short overview of three of the most used protocols.

Ad hoc On-Demand Distance Vector Routing (AODV) is developed by Nokia Research Center, University of California, Santa Barbara and University of Cincinnati and is a routing protocol for mobile ad hoc networks (MANETs). The AODV Routing Protocol uses an on-demand approach for finding routes, meaning that a route is established only when it is required by a source node for transmitting data packets. AODV allows mobile nodes to obtain routes quickly for new destinations, and does not require nodes to maintain routes to destinations that are not in active communication. AODV
allows mobile nodes to respond to link breakages and changes in network topology in a timely manner. [5]

**Dynamic Source Routing protocol (DSR)** is a simple and efficient routing protocol designed specifically for use in multi-hop wireless ad hoc networks consisting of mobile nodes. DSR allows the network to be completely self-organizing and self-configuring without the need for any existing network infrastructure or administration. The protocol is composed of the two main mechanisms of "Route Discovery" and "Route Maintenance", which work together to allow nodes to discover and maintain routes to arbitrary destinations in the ad hoc network. All aspects of the protocol operate entirely on demand, allowing the routing packet overhead of DSR to scale automatically to only what is needed to react to changes in the routes currently in use. [6]

**Better Approach to Mobile Ad-hoc Networking - Advanced (B.A.T.M.A.N-Adv)** is a proactive mesh routing protocol which mostly can be classified as a Distance Vector protocol. The protocol is based on flooding the network with a particular message called Originator Message (OGM) and uses the information contained in these messages to create and maintain the routing table. [1]

The approach of the B.A.T.M.A.N-Adv algorithm is to divide the knowledge about the best end-to-end paths between nodes in the mesh to all participating nodes. Each node perceives and maintains only the information about the best next hop towards all other nodes. Thereby the need for a global knowledge about local topology changes becomes unnecessary. Additionally, an event-based but timeless (timeless in the sense that B.A.T.M.A.N-Adv never schedules or timeouts topology information for optimizing its routing decisions) flooding mechanism prevents the accrual of contradicting topology information (the usual reason for the existence of routing loops) and limits the amount of topology messages flooding the mesh (thus avoiding overly overhead of control-traffic). The algorithm is designed to deal with networks that are based on unreliable links. [7]

### 2.2 Zero Configuration Networking

Zero configuration networking, or zeroconf for short, is a design concept and a set of technologies aimed towards providing communication across IP based networks without any need for manual configuration. A zeroconf implementation typically involves services such as DHCP and DNS to provide hosts with configuration and information. This means that hosts participating in such a network will get their network configuration and hostname resolution handled by the zeroconf implementation instead of manually configuring it [10]. The implementation of zeroconf chosen for this project is called Avahi, which is an open source daemon licensed under
LGPL that can be used to discover hosts on a network without any conventional DNS server through the use of Multicast DNS [11][17].

2.3 Hypervisor

A hypervisor is a piece of software that is able to split up a single hardware configuration into multiple execution environments. That is to say, hypervisors enables the installation and simultaneous usage of multiple operating systems, so called “Virtual Machines”, on a single piece of hardware, a “Host Machine”. These virtual machines will generally act the exact same way that a physical one would, and do indeed act without any knowledge of being a virtual machine. There are two types of hypervisors.

The most efficient one, called a type 1 hypervisor enables the installation of multiple virtual machines directly on top of the hardware, which is achieved through a very specialized separate operating system. A type 2 hypervisor, or a hosted hypervisor, is a hypervisor that is installed on top of a preexisting operating system which is not exclusively designed to handle virtualization. This is less efficient because of the host operating system adding a layer of abstraction between the virtual machines and hardware resources [12].

The hypervisor used in this paper is VirtualBox, a free, open source type 2 hypervisor which is released and maintained by Oracle under the GPL version 2 license [13]. VirtualBox is optimized to run the most common guest operating systems, including Ubuntu which was used in this paper, at a close to native level of performance [18].

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

B.A.T.M.A.N-advanced, short for “Better Approach To Mobile Ad-Hoc Networks” is a routing protocol in the form of a module written for the Linux kernel, which implements routing based on hardware addresses. Earlier versions of the routing protocol was a user space daemon. The reason for moving the application down to kernel level was to reduce the performance cost in regards to the processing power needed per packet. Every node which participates in a B.A.T.M.A.N-Adv mesh network is actively routing traffic for other nodes, thus creating a mesh network where all participants are able to forward data. Once the network has converged each nodes will view every other node within range as being connected to the same switch. The B.A.T.M.A.N-Adv wiki page lists the following as important characteristics of the network [14]:

- Network-layer agnostic - you can run whatever you wish on top of batman-adv: IPv4, IPv6, DHCP, IPX.
- nodes can participate in a mesh without having an IP
- Easy integration of non-mesh (mobile) clients (no manual HNA
- Fiddling required
- Roaming of non-mesh clients
- Optimizing the data flow through the mesh (e.g. interface alternating, multicast, forward error correction, etc)
- Running protocols relying on broadcast/multicast over the mesh and non-mesh clients (Windows neighborhood, mDNS, streaming, etc)

In this context HNA refers to host and network association, which is a list of network addresses and netmasks typically used by gateways to associate themselves with the network. Forward error correction is a method for detecting and managing transmission errors of data in noisy or unreliable networks.

An important asset to the B.A.T.M.A.N-Adv routing protocol is batctl. This is an application made to communicate with the B.A.T.M.A.N-Adv routing protocol. It implements ease of use in terms of setting up, adding or removing network interfaces to the mesh, configuring the kernel module by reading and writing to and from the modules file system. Furthermore it can be used to view nodes in the network which originate traffic for the mesh as well as clients connected to the network [15].

2.5 Ubuntu

Ubuntu is a distribution of GNU/Linux which is based on Debian. It is geared towards being easy to setup and use. It is a highly popular distribution of Linux, and works on a multitude of different platforms such as desktops, servers, virtual machines, embedded, phones and tablets. The development of the distribution is partly community based but is also supported by the company behind Ubuntu, Canonical [16].
3 Method
This section details the scientific approach taken for the thesis and presents the testing environment that was designed for the tests, including what hardware and software that was used. The section also includes detailed information on how the tests were performed, the reliability of the gathered information as well as a description of how the tests were conducted.

3.1 Scientific approach
A mixed method approach was chosen for this thesis, implementing elements of both quantitative and qualitative research. In order to generate relevant and reliable test data, a set of straightforward tests were designed to test the capabilities of the network. In the cases where it was possible, this data was created in a quantitative manner where each test was performed multiple times, or over longer periods of time, with as little potential for deviation in the testing environment as possible. In other cases, such as with tests regarding network threats, qualitative testing was used where the results were interpreted and analyzed. In general, the data and overall test results were analyzed using an inductive approach, creating theories from observations of results and potential non-results. Lastly, a deductive approach was used to try and resolve possible non-results, find ways to resolve these issues and find ways to make the network more efficient.

3.2 Environment
The basic test environment consisted of two laptops connected at either end of a network consisting of one to three host machines running ten virtual machines each, and a third laptop for monitoring purposes, all connected through a Cisco 3560 switch. The switch was configured with VLANs so that each inward connection only had one outward interface to travel through, essentially forcing the network to form a linear chain, see figure 3.1. This structure was used in order to ensure that the network traffic traversed through every unit in the network which was important to test the effect that increasing the network size would have.
The laptops and the virtual machines running on the hosts were configured with a set of software necessary for the conduction of the tests, listed under section 3.4. The installation and configuration of the software on these machines was managed through the scripts listed in Appendix A.A and A.B. With the use of these scripts anyone with sufficient knowledge and the right hardware as well as software could recreate a test environment with the same properties.

3.3 Tests

A handful of tests were designed to test the network, these tests were used to test hardware usage, response time, redundancy and resistance to some basic threats. For response time and redundancy each test was performed multiple times to make sure that the results were somewhat consistent. The hardware test was performed by observing the hardware usage over an extended period of time to see that the hardware usage was at a stable level before recording it. The tests against threats were instead analyzed to understand the results, as they did not provide any meaningful recordable data. The performance tests were repeated with an increasing number of virtual machines, going from 10 to 20 to 30 by adding an additional host with 10 VMs between each test. In the full mesh topology, a total of 40 virtual machines were used.

3.3.1 Hardware Usage

To measure hardware usage each of the two laptops at either end of the network performed a ping flood to each other to ensure a steady flow of traffic through the entirety of the network. During the traffic flow the CPU...
load, RAM allocation and network usage (packages sent/received) were measured using the built in system monitor and iostat on the test nodes. A more detailed description of the commands used and the raw data from the tests can be found in Appendix B.A. CPU, RAM and network load were also measured while no traffic beyond idle communication were running on the network to provide a baseline. This was performed once each time the number of VMs were increased.

3.3.2 Response Time

To measure response time, a preset number of ICMP packages were sent from either end of the network, after which the average response time of all the sent packets was recorded. This was repeated multiple times, each time that the number of VMs were increased to ensure that the results were consistent. The response time was tested both with the original network design of a linear chain that went through every unit in the network, and in a meshed environment where multiple paths were available and each unit could directly communicate with several other units. A more detailed description of the commands used and the raw data from the results can be found in Appendix B.B and B.C.

3.3.3 Redundancy

While the ideal way to measure redundancy would have been to use a full mesh topology and then use a path tracking tool such as traceroute [19] to establish the exact route of the traffic, this was not possible because such tools were incompatible with the network. This could have been used to find the path an established connection took through the network, deliberately break a link in the route and measure how long it would take to calculate a new route and resume traffic. The reason that this does not work is that B.A.T.M.A.N-Adv abstracts the network layer that traceroute operates at, leading traceroute to report a total of one hop, regardless of actual distance to the node.

What was done instead was to keep making use of the chain network model from the previous tests and simply introduce two bridges in the middle, through which traffic would always travel if possible as it was a shorter path than going through all the VMs in the first host. See figure 3.2. An ICMP stream was established between the two ends of the network after which the bridge that went through the first host was removed, forcing the protocol to recalculate its route. The time taken for communication to reestablish was measured, and the test was run three times for each phase, with one two and three hosts. A more detailed description of the commands used and the raw data from the results can be found in Appendix B.D.
Figure 3.2 – The network with two bridges introduced in the path through the first host

3.3.4 Threats

As the network works on the principle of not needing any underlying infrastructure, there are generally not going to be any security oriented hardware nodes in the network. Security precautions such as ASAs (firewalls), packet filtering services in routers and similar units simply have no place in the network by default. With this in mind three types of threats were put forth as being especially relevant to the function and integrity of the network:

**IP duplication**
This is defined as an instance where a malicious user spoofs an IP address that already exists in the network to disrupt or monitor its traffic. As the network self-configures the addresses of all hosts, this problem should not occur due to faulty usage. However, it could occur if a malicious user does it in order to disrupt the network or to attain information designated for another node in the network. Since the network by default lacks central management, this could potentially be very difficult to handle.

**Hostname duplication**
This is similar to IP duplication but instead of an IP address, the hostname of a preexisting node is spoofed to potentially confuse nodes trying to reach that hostname. This problem is more likely to occur on the network since the host name is chosen by the user of each node. This also means that it is trivial to enter the network with an already known host name. It could potentially mean that users of the network cannot be sure that the host name they communicate with leads to the correct node, as the host file housing all host name to IP address resolutions is updated continuously and could accidentally clear out the correct hostname from the table.
**Hardware address spoofing**

This is an attack where a malicious user spoofs an existing host’s hardware address in order to place itself on the route of its target. This is a common method used for so called “Man-In-The-Middle” attacks. If successful, the network routes all traffic destined to and from the target through the attacker, which enables the attacker to sift through potentially sensitive data [22].

Based on these threats three tests were constructed to analyze how the network would handle these types of attacks. More information on how these tests were performed and the results thereof can be found in sections 3.5.2 Network security and 4.2.2 Security results.

### 3.4 Software

In order to achieve reliable and consistent results from testing, all nodes participating in the network were using the same software. Following is a list of all software, including version numbers that were directly utilized:

- Ubuntu Server 13.10 x86_64
- batman-adv 2014.0.0
- batctl 2013.3.0-2
- avahi-daemon 0.6.31-2ubuntu4.1
- avahi-utils 0.6.31-2ubuntu4.1
- htop 1.0.2-2
- bmon 1:2.1.1~pre1-1
- sysstat 10.1.6-2
- build-essential 11.6ubuntu5

With the exception for the B.A.T.M.A.N-Adv routing protocol all software was retrieved from Ubuntu’s standard repositories, using their most recent versions at the time of downloading (April, 2014). B.A.T.M.A.N-Adv was downloaded directly from its development website, www.open-mesh.org. The version chosen was the most recent revision at the time the system initiation script was written (January 2014).

Ubuntu 12.10 is the base operating system, B.A.T.M.A.N-Adv is the routing protocol and batctl is a tool used to configure and monitor the routing protocol.

Avahi-daemon and avahi-utils are the zeroconf implementation used and a toolset for it respectively, and were used to discover active users participating in the network.
Htop, bmon and sysstat are tools used to monitor node performance and utilization used to gauge how the network performs while running.

Build-essential is a toolset used to compile the routing protocol.

3.5 Implementation

This section is split into two parts. The first one describes how the network itself was set up and how the tests mentioned in section 3.3 were implemented. The other section describes how the security tests were set up and performed.

3.5.1 Network performance

Initially two extended lab sessions were spent testing out the network in a general sense and loosely testing the performance of the network to gather knowledge and prepare for the final tests. This included adding as many hosts as possible, doing tests where every node was connected to every other node and examining idle traffic and network strain. On the whole these sessions went well and showed that the network was generally easy to work with. One problem that was encountered though was the loss of connectivity when the amount of hops a packet had to take to get to one end of the network and back grew higher. This remained unresolved at the time and was a potential weak point of the network.

In the third and final lab session the tests were instead pre planned and carried out systematically as they were intended to be used to gather the final results of the performance tests. Initially the network was set up with one laptop connected with a direct link to a host machine containing ten VMs, each VM using two network interfaces set on different virtual networks so that each virtual machine only had direct contact with two other clients, one “ahead” of it and one “behind”. The host was in turn connected to a Cisco 3560 switch, and through that switch to another laptop, with the interfaces of the switch being configured with VLANs to ensure that these nodes had no direct contact with anything else connected to the switch through these interfaces. This laptop was finally connected through the switch to a third laptop. The first and the third laptops, placed at either end of the network, were designated as the test nodes of the network, on which the tests were performed and performance measured. The middle laptop was used as a hardware monitor to examine how a unit inside the network would be affected while the network was in use, and was deliberately chosen as a computer with relatively weak hardware.
With the environment set up, a response time test was performed by sending 10000 ICMP packets from either of the two test nodes at the same time as quickly as the application could manage. The average response time from these 10000 packets was then recorded for both of the nodes, and the tests repeated five times.

After this, the hardware performance was tested by sending a constant stream of ICMP packets with the packet size set to 1KiB was sent from either node to generate some more substantial traffic for the network. While the packets were flooding through the network the CPU, RAM and network load was measured on both the testnodes as well as the hardware monitor. The values were monitored for an extended period of time to ensure that the values were somewhat stable, and were then recorded for each of the three machines. Likewise a baseline test was performed by monitoring the three nodes in the same fashion while nothing was running on the network.

Finally the redundancy test was performed by creating a bridge that went over the host with the VMs, and establishing a connection between the two testnodes. As the bridged path was shorter, it was preferred by the hosts. After this the bridge was severed, forcing the nodes to once again start using the path through the VMs. The time taken for the testnodes to reestablish their connection was measured by hand, and the test repeated three times to make sure that it did not vary greatly from time to time.

With all three tests completed, another host with 10 VMs was then added to the network. See figure 3.2. And all the tests were repeated in the same fashion, and lastly a third host with an additional 10 VMs was added for the final round of tests.

While the tests went well with a second host and a total of 20 VMs, just like in the initial session’s problems arose when the final 10 were added for a total of 30 VMs, and connectivity could not reach through the entire network. The problem was isolated to the two last VMs in chain 3, see figure 3.3, and to Testnode02. These nodes became members of the mesh, so called Originators, but as the mesh grew they stopped responding and where flushed from the Originator list after 200 seconds. Restarted the network gave the same result, with the same nodes once again being flushed out of the mesh after timing out.

![Diagram](image)

Figure 3.3 – The network setup used in Phase 3, where the red markings represent points in the network that do not respond.
After a bit of reasoning a “bridge” was added from the middle of chain 2 to the start of chain 3, see figure 3.4. When the bridge was active, and thus supplying the network with a new route, all 33 nodes (physical and virtual) where active in the mesh. But as soon as the bridge was shut down the same three nodes as before stopped responding and where flushed from the originator list. Furthermore, nodes in the middle of the network could communicate with either of the two testnodes with or without the bridge. These two things confirmed that the problem was not with the nodes themselves, and that there must be a reason that the network would always break at the same spot. This lead to the realization that the problem was likely to be located somewhere in the protocols configuration.

Examination of the B.A.T.M.A.N-Adv documentation eventually lead to the conclusion that the protocol had a Time To Live (TTL) that could not handle the entire width of the network. After modifying it communication worked once more through the network and the third phase of tests could be performed.

As a final step, a large mesh network was constructed in order to verify that the network worked as intended with a realistic mesh topology. See figure 3.5. Similar tests were conducted on the mesh topology, though because of how the network functions the test results where difficult to interpret, as there was not any way of knowing which path the test traffic took. Aside from that, no significant observations were made about the network except that it worked as intended, with the nodes being able to freely communicate with each other without any issues.
3.5.2 Network security

A few basic tests were constructed and carried out in order to gauge how the network responds to malicious usage according to the threats discussed under section 3.3.4 Threats. As these tests where aimed at testing software related behavior of the network, the environments used were entirely virtual. Two different test environment were used for the tests.

For tests regarding IP and Hostname duplication, a pure mesh network consisting of four nodes in direct contact with each other over the same virtual network was used. See figure 3.6. Tests were carried out by changing hostname and IP by hand and then monitoring traffic, the host list and the originator list for inconsistencies. The results were then cross referenced to minor tests conducted on the full scale test environment in order to make sure that nothing was overlooked.
For the tests conducted on hardware address spoofing one node was added to the virtual network and used for IP and hostname duplication. The added node did not connect directly to the mesh, but to one of the meshed nodes, forcing its data to travel through the bridging node. See figure 3.7. This forced data to travel a specific path to reach the new node, enabling monitoring of the data. Once established, the hardware addresses were spoofed manually, one node at a time, and traffic was sent between different nodes all throughout the tests, which was monitored to interpret how the flow of data was affected.
3.6 Reliability

In order to eliminate potential sources of inconsistency, an environment where as few outside factors as possible could interfere with the test results was created. The network used was self-contained and did not have any contact with the internet or with any device not specifically used to build the network. All network clients ran the same software and used the same scripts for initialization, and all virtual machines ran with the same virtualization software. The end points of the network chain were always the same, with only the network in between them growing in incremental steps between tests. With the exception of the meshed tests which measured how the network could handle multiple paths in regards to redundancy and load balancing, the network was always configured in such a way that all traffic was forced to travel through the entire network through a sequential chain of clients to ensure that the full width of the network was tested.

As for limitations, a wider range of hardware, both in terms of performance and architecture, would preferably have been tested in order to ensure the reliability of the results obtained through the tests. It is also difficult to definitively say what influence using mainly virtual clients might have had on the networks behavior and performance, though it should not be of great significance as they only acted as a medium for traffic and routing information to spread through, with all data being recorded on physical hardware. Some test results are also subject to a certain level of human error.

There was no way to record exact numbers for CPU usage and data transfer speed over time during hardware test as it had to be read and written down in real-time. The results therefore had to be interpreted individually as some fluctuation of the values had to be accounted for, though in the end the margin of error should be very small. Likewise the time it took for the network to find and start using a new route after the previously used path had been broken had to be timed manually, but considering that a clear pattern can still be observed from the tests millisecond precision does not appear to be of great importance.
4 Results
This section reviews the scripts that have been constructed to set up the network in terms of design and function, and presents the results gathered from all the performed tests on the network.

4.1 Script and network
In order to achieve zero configuration and system autonomy two scripts were devised. The first script is called “init”. The function of this script is to prepare a node for running the components needed to participate in the network. This script is written for Debian based Linux distributions though as it is written in the BASH language it is compatible with all distributions of Linux with minimal changes. Init works by retrieving all of the applications that are needed to run the network, as well as downloading and compiling the B.A.T.M.A.N-Adv kernel module and configuring the system to use that kernel module.

Once init has been executed, the system is ready to enter the network. This is managed by the script “bat”. Bat handles addressing, adding interfaces on the node to B.A.T.M.A.N-Adv, setting up Avahi to report neighbors on the network and place those neighbors in the local hosts file, so that they can be accessed without using DNS-services.

![Figure 4.1](image)

Figure 4.1 – Host file is backed up and then rewritten with hostnames found through Avahi.

Bat can be executed with one of two options:
- By supplying a username
- By running in anonymous mode

In username mode the node’s hostname becomes “username”.local. In this mode the IP is set based on the nodes hardware address. Once a connection to the network has been established this hostname will be found by Avahi on the other nodes, and through a function in the script added to their hostfiles. These nodes will then be able to identify the new node by the supplied username, and achieving this will thereafter be able to communicate freely as they would in any conventional network topology.
Anonymous mode works in a similar manner but with some key differences. First off, before ever entering the network, all hardware addresses are spoofed to random ones, generated from entropy in the local node as seen in figure 4.2.

![Figure 4.2 – The hardware, or MAC, address is set to a randomly generated number.](image)

After that the hostname is set according to the generated hardware address by appending it to anon-"address”, so for instance “anon-c84b08127e9c.local”. The IPv6 address made for the hosts is also generated from system entropy in a similar fashion, as seen in figure 4.3.

![Figure 4.3 – The creation of a random IPv6 address.](image)

Bat also features a restore section used to exit the network and restoring the system to its previous state. This is done automatically when the script is shut down. Both scripts in their entirety can be found in Appendix A.A and A.B respectively.

### 4.2 Test results

This subsection presents the results from the two main areas of testing. The results from the network performance tests and from the network security tests.

#### 4.2.1 Network performance results

The results of the hardware tests, which can be read in their entirety in Appendix B.A, show that CPU and RAM usage was largely unaffected by the traffic and the size of the network. While there was some variation in the baselines for the tests due to outside factors that would have been difficult to completely control, the relative results from when the network was under load shows that there was not a significant amount of additional strain that was put on the network when it was in use. Even the so called hardware monitor which was running on very old hardware did not have any problems
keeping up, and the strain put on the testnodes did not noticeably increase as the network grew larger, with the RAM usage staying consistent and the CPU usage remaining at a few percent at the most throughout.

There was a noticeable drop in the traffic that ran through the network during the tests despite the same ICMP-commands being run each time, as seen in Appendix B.A under the Network Load column. Seeing as the drop was only noticeable when going from 10 to 20 VMs, and being about the same for 20 and 30 VMs, this is likely to be because of outside factors and not a limitation of the network. In particular when paired with the fact that none of the nodes was shown to be otherwise affected by the growth of the network.

The results from the tests performed to measure response time in the network, available in its entirety in Appendix B.B, show fairly low response times throughout. As the network grew from 10 to 20 VMs, the response time went up by about 2ms, and then an additional 3ms when going from 20 to 30 VMs, reaching a response time of about 8-9ms at most. This shows that there was no dramatic increase as the network grew. The results show that the amount by which the response times increased grew when going from 20 to 30 compared to 10 and 20, though the increase was not big enough to suggest an exponential growth were it would quickly become unsustainable.

The redundancy tests, the results of which can be seen in Appendix B.C, show that the time it took for the network to calculate a new route was not noticeably affected by the size of the network. The median time was around 24 seconds, with a lowest and highest time of about 15 and 30 seconds respectively. The occasional variance in the time it took might have been due to the timing of Avahis Hello-packets which are sent out at fixed times to update the networking tables for all the hosts.

Appendix B.D shows that in the meshed topology that was set up, the test nodes could communicate with each other almost directly, making the response times very low. The results in the appendix also makes it clear that there was no measurable difference in the response time from adding more nodes to the network.

### 4.2.2 Security results

The tests that were conducted on IP duplication found that this form of threat was highly ineffective, since the IPv6 “Duplicate Address Detection” handles duplicate IP addresses [23]. Multiple tests found that if a duplicate address was assigned to the network, only the preexisting addresses would be accessible.

For hostname duplication, that is to say adding new nodes to the network with already known hostnames, tests found that this would not be an issue, as duplicate host names were handled by Avahi, the zeroconf daemon. If a duplicate host name were to be introduced to the network it will simply get a
“-2” appended to its name. If the same name enters the network yet again, a 
“-3” will be appended, and so on.

Finally, the tests for hardware address spoofing found that this type of attack did not work as intended in the network. Seeing as routing is performed at the addresses being spoofed and that the network is aware of where the spoofed addresses really are, communication may get disrupted by such an attack but only if the attacker is directly connected to its target, in which case the attacker will get all sent data from the target but will not be able to forward it further, effectively ending the targets communication and rendering the attack unsuccessful.
5 Discussion

The purpose of this section is to provide discussion and analysis on the research and results of the paper. This includes the perceived usability of the network and the ease of which it can be established, possible applications for it as well as things future research to be done.

5.1 Usability

In section 2 “Background and theory”, it was mentioned that mesh networks are perceived to be complex to set up, manage and maintain, and that they do not scale well. The method of setting up the network that was created in this paper through the use of two scripts proved that it would not necessarily have to be complicated to set up a mesh network. While at this stage it would still be difficult to use for someone without any knowledge at all, even an introductory amount of knowledge would make the setup fairly easy. And the experiences gathered about the usage suggests that when the network is up, it is easy to use as it handles addition and subtraction of units on its own without interfering with the core functionality of the network.

However, scaling was somewhat problematic to measure. As mentioned in section 3 “Method” the network had some issues when the number of hops from one node to another started to grow. This was found to be caused by the TTL function in B.A.T.M.A.N-Adv. It is designed so that no packets can travel further than the TTL. The perceived purpose of this is to limit broadcast domains. In practice, this means that to a single node the network will never be wider in regards to number of nodes than the set TTL, but the network itself can grow beyond this limit.

Since any number of nodes can be confined inside the broadcast domain, congestion could potentially occur, but this will only be of local significance. This does however make it difficult to measure or predict what the exact limitations of the network would be.

5.2 Possible applications

There is a number of possible applications for a network like this. One of the originally thought of applications is a proximity based social networking function. That is, connecting and communicating with other network participants in the direct vicinity of your node. This could be combined with grouping functions similar to those of conventional social networks, enabling participants to more easily meet up in person, than if they were communicating over larger distances, such as over the Internet.

Friend lists could be formed by sharing each other’s encryption keys, thus enabling safe encrypted communication, such as in the case of PGP webs of trust. [20]
Home appliances could be connected to this kind of network as well, giving control and communication with them from any number of trusted devices with great ease.

Furthermore, home Internet connections could easily be shared through the routing protocols integrated Gateway mode, thus extending Internet access across potentially great distances. [21]

As this network can run on such a multitude of different devices, battery powered devices and ad-hoc Wi-Fi can enable the network to stay alive even during a power outage or natural catastrophes. Communication even with just other nodes in proximity is sure to be of great advantage during such an event.

5.3 Conclusion
The goal of this paper was to create an autonomous network without any need for infrastructure, that was relatively easy to configure, use, and which performed well. The implementation which has been put forth, while still proof of concept has been successful at these goals.

The script and environment that was constructed makes it easy to set up and join nodes into the network, and the network can increase and decrease in size without affecting the core functionality of the network. The implementation for automatic host discovery makes it simple for anyone with a small amount of knowledge to find and communicate with other hosts, and the network has proven to be resilient to some common ways of tampering. Furthermore the response time was shown to scale well with the size of the network, and the strain on the host’s hardware proved to be within reason.

As wireless technologies become better and more reliable, this is a networking concept that shows a lot of promise for the future, with many possible applications and areas of usage both big and small. And the implementation presented in this thesis could be a stepping stone towards making it a reality.

5.4 Future research
This is a proof of concept solution to the kind of zero configuration autonomous network that was defined in this paper, where functionality has been achieved by piecing together readily available applications and doing some basic solutions to make them work together through the bat script. Future versions should focus on reducing the number of needed applications and removing unused functionalities that are generating network traffic, specifically tailoring the system to function with this type of network. Another point of further interest that was touched on in section 1.5 is the scalability of the physical size of the network, which could not be tested due to a limitation of resources. Furthermore, a significant future step would be to make it available on completely different platforms, such as windows,
thereby enabling a more widespread usage. Lastly the fluctuation in Network Load during the hardware tests which was mentioned in section 4.2.1 is something which could be re-tested in order to make sure that it had no noticeable impact on the results, or potentially to find out the reason from it happening in the first place.
References


A Scripts
This appendix contains the two scripts which were used to set up and configure our mesh network.

A.A Bat
#!/bin/bash

#Test if user is wrong and weather batman-adv kernel module exists or not
if [ -z $1 ] || [ $1 == "-h" ] || [ $1 == "h" ] || [ $1 == "help" ]; then
    echo -en "Usage: Call bat and supply your username\n"
    echo -en "Example: bat myusername\n"
    echo -en "Or: bat anon (for anonymous mode)\n"
    exit 1
fi

if [ ! -z $2 ]; then
    echo -en "Your username cannot contain spaces\n"
    exit 1
fi

if [ $1 == "anon" ]; then
    anon=True
else
    anon=False
fi

modprobe batman-adv > /dev/null 2>&1
if [ $? == 1 ]; then
    echo -en "Your device is not properly equiped to run this application.\n"
    exit 1
fi

#Functions
function setip6 () {
    mac=$(cat /sys/class/net/bat0/address)
    ifconfig bat0 inet6 add ba70::$mac/32
}

function addbat () {
    ifconfig $1 mtu 1532
    wless=$(iwconfig $1 2> /dev/null )
    if [[ $wless == *IEEE* ]]; then
        ifconfig $1 down
        iwconfig $1 mode ad-hoc essid outernet ap
    fi
}
fi
ifconfig $1 up
batctl if add $1
}

function randv6 () {
    randnum=( 1 2 3 4 5 6 7 8 9 0 a b c d e f )
    nw=ba70:0000 #ipv6 network prefix
    a=${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}
    b=${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}
    c=${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}
    d=${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}
    e=${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}
    f=${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}${randnum[$RANDOM%16]}
    ifconfig bat0 inet6 add $nw:$a:$b:$c:$d:$e:$f/32
}

function randmac () {
    ifconfig $1 down
    while true; do
        macaddr=$(dd if=/dev/urandom bs=1024 count=1 2>/dev/null|md5sum|sed 's/^\([\w]{2}\)\([\w]{2}\)\([\w]{2}\)\([\w]{2}\)\([\w]{2}\)\([\w]{2}\)\([\w]{2}\)/\1:\2:\3:\4: \5:\6/')
        ifconfig $1 hw ether $macaddr > /dev/null 2>&1
        if [ $? == 0 ]; then
            ifconfig $1 up
            break
        fi
    done
}

###Begin
#Get local interfaces capable of running batman
interfaces=$(ifconfig -a | sed 's/[^t].*/;\^\(lo\|bat0\|\)\$/d')
for inter in $interfaces; do
    echo -en "Adding $inter to the mesh\n"
    if [ $anon == "True" ]; then
        randmac $inter
        addbat $inter
    else
        addbat $inter
    fi
done

#Bring up bat interface
anybats=$(batctl if)
if [ -z "$anybats" ]; then
echo -en "None of your interfaces are connected to the mesh\n"
exit 1
else
    ifconfig bat0 up
    if [ $anon == "True" ]; then
        randv6
    else
        setip6
    fi
fi

#Update hostname and backup hosts file
orghostname=$HOSTNAME
if [ $anon == "True" ]; then
    mac=$(cat /sys/class/net/bat0/address | sed 's/://g')
    hostname anon-$mac
else
    hostname $1
fi
HOSTNAME=$(hostname)
cat /etc/hosts > orghosts
cp orghosts hosts
echo "#End of regular hosts file" >> hosts
sed -i "s/localhost/localhost $HOSTNAME/g" hosts
sed -i "s/ip6-loopback/ip6-loopback $HOSTNAME/g" hosts
/etc/init.d/avahi-daemon restart > /dev/null 2>&1
sleep 5

#Daemon to update hosts file with mesh participants
while true; do
    echo "" > bathosts
    #avahi-browse -arpt | grep bat0 | grep -v + | cut -s -f8,7 -d";" | sed 's:// /g' | awk '{ print $2 " "$1} | awk '!(a[$0]+) > bathosts
avahi-browse -arpt | grep "=;bat0" | cut -s -f8,7 -d";" | sed 's;/ /g' | sort | uniq | awk '{ print $2 " " $1}'} > bathosts
    cat hosts bathosts > allhosts
    cat allhosts > /etc/hosts
    sleep 20
done &
hostupdate="$!"

# Kill and restore
while true; do
    read -p "Press any button to quit " q
    kill $hostupdate
    wait $hostupdate 2>/dev/null
    cat orghosts > /etc/hosts
    rm hosts bathosts allhosts orghosts
    hostname $orghostname
    HOSTNAME=$(hostname)
    service avahi-daemon restart > /dev/null 2>&1
    ifconfig bat0 down
    interfaces=$(ifconfig -a | sed 's/[^t].*//;/^lo|bat0|$/d')
    for inter in $interfaces; do
        echo -en "Removing $inter from the mesh\n"
        batctl if del $inter
        ifconfig $inter down
        ifconfig $inter mtu 1500
        sleep 2
        ifconfig $inter up
    done
    break
done


A.B Init

#!/bin/bash
#Download prerequisites, compile and load batman-adv
#Run as root

if [ $UID -ne "0" ]; then
  echo -en "Run as root. Issue command 'sudo passwd root; su'\n"
  exit 0
else
  apt-get update && apt-get upgrade -y
  apt-get install openssh-server batctl build-essential bmon avahi-utils avahi-discover htop screen ethtool vlan -y
  tar -xf batman-adv-2014.0.0.tar.gz
  rm batman-adv-2014.0.0.tar.gz
  cd batman-adv-2014.0.0/
  make
  make install
  depmod
  modprobe batman-adv
  cd ..
  rm -rf batman-adv-2014.0.0/
  echo "batman-adv" >> /etc/modules
  reboot
fi
**B Test results**

This appendix contains the results from our second lab session. Complete with short summaries on how the tests were designed and executed.

**B.A Hardware usage**

We measured hardware usage by sending a large amount of data and observe how the physical and virtual hardware in our hosts were affected. To observe the load on our test nodes we used the built-in System Monitor combined with the command `watch -d 'iostat -c'`. This command gave us a summarized percentage of the CPU-usage of the nodes.

The values inside the brackets are the idle value, i.e. the value recorded with no load.

<table>
<thead>
<tr>
<th>ICMP command</th>
<th>-i 0.001</th>
<th>-s 1016</th>
</tr>
</thead>
<tbody>
<tr>
<td>ping6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sends an ICMP-package with IPv6</td>
<td>Sets the interval between each packet.</td>
<td>Sets the size of the package to 1016 bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 1, 10VMs</th>
<th>Host</th>
<th>CPU</th>
<th>RAM</th>
<th>Network load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Testnode01</td>
<td>1.62% (1,68%)</td>
<td>17,1% (17,1%)</td>
<td>S: 563KiB/s (262b/s) R: 568KiB/s (2,3KiB/s)</td>
</tr>
<tr>
<td></td>
<td>Testnode02</td>
<td>6% (5,09%)</td>
<td>15,1% (14,5%)</td>
<td>S: 591KiB/s (192b/s) R: 591KiB/s (2,5KiB/s)</td>
</tr>
<tr>
<td></td>
<td>HWmon</td>
<td>3% (1,4%)</td>
<td>16,8% (16,2%)</td>
<td>S: 595KiB/s (3KiB/s) R: 594KiB/s (3KiB/s)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 2, 20VMs</th>
<th>Host</th>
<th>CPU</th>
<th>RAM</th>
<th>Network load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Testnode01</td>
<td>0,66% (0,67%)</td>
<td>8,6% (8,5%)</td>
<td>S: 380KiB/s (0 KiB/s) R: 375KiB/s (0 KiB/s)</td>
</tr>
<tr>
<td></td>
<td>Testnode02</td>
<td>0,21% (0,19%)</td>
<td>7% (7%)</td>
<td>S: 383KiB/s (0 KiB/s) R: 383KiB/s (0 KiB/s)</td>
</tr>
<tr>
<td></td>
<td>HWmon</td>
<td>1,6% (1,4%)</td>
<td>17% (16%)</td>
<td>S: 410 KiB/s (3KiB/s) R: 410 KiB/s (3KiB/s)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 3, 30VMs</th>
<th>Host</th>
<th>CPU</th>
<th>RAM</th>
<th>Network load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Testnode01</td>
<td>0,32% (0,23%)</td>
<td>10,3% (10,3%)</td>
<td>S: 372KiB/s (1,6 KiB/s every 20s) R: 368KiB/s (3,2 KiB/s)</td>
</tr>
</tbody>
</table>
### Phase 1, 10 VMs

<table>
<thead>
<tr>
<th>Host</th>
<th>CPU</th>
<th>RAM</th>
<th>Network load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testnode02</td>
<td>0,36% (0,26%)</td>
<td>8,5% (8,5%)</td>
<td>S: 377KiB/s (1,6 KiB/s every 20s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R: 374KiB/s (3,2 KiB/s)</td>
</tr>
<tr>
<td>HWmon</td>
<td>2% (1,5%)</td>
<td>16,8% (16%)</td>
<td>S: 403 KiB/s (3 KiB/s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R: 402 KiB/s (3 KiB/s)</td>
</tr>
</tbody>
</table>

### B.B Response time

We measured response time by sending the following ICMP-package between our test nodes (Testnode01 and 02). Then we recorded the average response time in the ICMP end message.

<table>
<thead>
<tr>
<th>ICMP command</th>
<th>ping6 -c 1000</th>
<th>-i 0.001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sends an ICMP-package with IPv6</td>
<td>Sets ICMP to send 1000 packets and then stop.</td>
<td>Sets the interval between each packet.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase</th>
<th>Between</th>
<th>Response time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 VMs + 3 mobile</td>
<td>Testnode01 Testnode02</td>
<td>L1 - 2.872</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L2 - 2.878</td>
</tr>
<tr>
<td></td>
<td>Testnode01 Testnode02</td>
<td>L1 - 2.887</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L2 - 2.948</td>
</tr>
<tr>
<td></td>
<td>Testnode01 Testnode02</td>
<td>L1 - 2.781</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L2 - 2.835</td>
</tr>
<tr>
<td></td>
<td>Testnode01 Testnode02</td>
<td>L1 - 2.747</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L2 - 2.793</td>
</tr>
<tr>
<td></td>
<td>Testnode01 Testnode02</td>
<td>L1 - 2.897</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L2 - 2.980</td>
</tr>
<tr>
<td>Phase 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 VMs + 3 mobile</td>
<td>Testnode01 Testnode02</td>
<td>L1 - 4.836</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L2 - 4.877</td>
</tr>
<tr>
<td></td>
<td>Testnode01 Testnode02</td>
<td>L1 - 4.732</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L2 - 4.825</td>
</tr>
<tr>
<td></td>
<td>Testnode01 Testnode02</td>
<td>L1 - 4.740</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L2 - 4.812</td>
</tr>
<tr>
<td>Phase</td>
<td>Between</td>
<td>Response time (ms)</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td>Testnode01</td>
<td>L1 - 4.748</td>
</tr>
<tr>
<td></td>
<td>Testnode02</td>
<td>L2 - 4.779</td>
</tr>
<tr>
<td>Phase 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 VMs + 2 mobile</td>
<td>Testnode01</td>
<td>L1 - 4.805</td>
</tr>
<tr>
<td></td>
<td>Testnode02</td>
<td>L2 - 4.845</td>
</tr>
<tr>
<td></td>
<td>Testnode01</td>
<td>L1 - 9.251</td>
</tr>
<tr>
<td></td>
<td>Testnode02</td>
<td>L2 - 9.333</td>
</tr>
<tr>
<td></td>
<td>Testnode01</td>
<td>L1 - 8.323</td>
</tr>
<tr>
<td></td>
<td>Testnode02</td>
<td>L2 - 8.445</td>
</tr>
<tr>
<td></td>
<td>Testnode01</td>
<td>L1 - 7.940</td>
</tr>
<tr>
<td></td>
<td>Testnode02</td>
<td>L2 - 8.070</td>
</tr>
<tr>
<td></td>
<td>Testnode01</td>
<td>L1 - 7.730</td>
</tr>
<tr>
<td></td>
<td>Testnode02</td>
<td>L2 - 7.714</td>
</tr>
<tr>
<td></td>
<td>Testnode01</td>
<td>L1 - 7.998</td>
</tr>
<tr>
<td></td>
<td>Testnode02</td>
<td>L2 - 8.027</td>
</tr>
</tbody>
</table>
B.C Full mesh

Our response time tests in the full mesh topology were executed in the same way as in the chain mesh.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Between</th>
<th>Response time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1 10 VMs + 3 mobile</td>
<td>Testnode01 Testnode02</td>
<td>L1 - 1.152</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 2 20 VMs + 3 mobile</td>
<td>Testnode01 Testnode02</td>
<td>L1 - 1.115</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 3 30 VMs + 3 mobile</td>
<td>Testnode01 Testnode02</td>
<td>L1 - 1.211</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase 4 40 VMs + 3 mobile</td>
<td>Testnode01 Testnode02</td>
<td>L1 - 1.120</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B.D Redundancy

We measured redundancy by adding a bridge in the network, and thus creating a shorter path to the end of the network. We then removed the bridge and measured the time it took to converge and find a new route.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>T1 (sec)</th>
<th>T2 (sec)</th>
<th>T3 (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1 10 VMs + 2 mobile</td>
<td>24.1</td>
<td>23.9</td>
<td>19.8</td>
</tr>
<tr>
<td>Phase 2 20 VMs + 2 mobile</td>
<td>29.9</td>
<td>24.8</td>
<td>24.5</td>
</tr>
<tr>
<td>Phase 3 30 VMs + 2 mobile</td>
<td>15.2</td>
<td>30.3</td>
<td>24.4</td>
</tr>
</tbody>
</table>