TREBALL DE FI DE CARRERA

TÍTOL DEL TFC : Wifi mesh network nodes on guifi.net

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Eduard Duran

DIRECTOR: Roc Messeguer Pallars

DATA: 23 de juliol de 2009

Títol : Wifi mesh network nodes on guifi.net

Autor: Eduard Duran

Director: Roc Messeguer Pallars

Data: 23 de juliol de 2009

Resum

Aquest estudi presenta la situació actual de la comunitat sense fils lliure i oberta anomenada guifi.net, la seva història i els seus reptes de futur.
L'estudi s'orienta particularment a les xarxes sense fils en mode mesh, que últimament ha estat en clar creixement a la comunitat. En concret, s'estudia el rendiment dels tres protocols d'enrutament més utilitzats en aquest tipus de xarxa, B.A.T.M.A.N., BMX i OLSR.
Durant la realització d'aquest estudi s'ha utilitzat un dels CPE més utilitzats actualment, l'Ubiquiti NanoStation 2, utilitzat bàsicament pel seu preu i mida, i amb el qual voldrem esbrinar si és adequat pel tipus de xarxa mesh.
Per tot això, es compilarà una versió particular del firmware OpenWRT i es generarà tràfic real per analitzar-ne el rendiment.

Title : Wifi mesh network nodes on guifi.net

Author: Eduard Duran

Director: Roc Messeguer Pallars

Date: July 23, 2009

Overview

This thesis describes the current situation of an open wifi community called guifi.net, its history and its future challenges.
This thesis is aimed specifically at the wireless networks in mesh mode, that have been lastly in a clear growth in the community. In detail, the performance of the mostly used routing protocols will be evaluated, i.e., B.A.T.M.A.N., BMX and OLSR.
During the realisation of this study one of the most used CPE was used, Ubiquiti NanoStation 2, widely spread basically due to its price and size, and whose performance in mesh networks will be analyzed.
To do all of this, a particular version of the firmware OpenWRT will be build and compiled, and real traffic will be generated to analyze the performance.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Introduction

Imagine a world in which every single person on the planet is given free access to the sum of all human knowledge. That's what we're doing.
Jimmy Wales, founder of Wikipedia, july 2004

Hard as it is to imagine, since 2004 Wikipedia has been accomplishing the task of compiling all that human knowledge on the Internet. So there's only one major thing left, giving Internet access to every single person the planet.

Back then in 2004, there was also a not so well-known group of people who thought they could contribute at giving access to IT to everyone. With practically no political support, nor big companies', they started an open wireless community which hasn't stopped exponentially growing and counts more than 6000 active and operative nodes at the beginning of 2009. Its name is guifi.net.

Although it started with the Infrastructure wireless mode, there has been a lot of progress recently in mesh networking; there is, though, no standard way of building that mesh network, nor is it clear if is going to scale up as well as infrastructure has done.

In Chapter 1 we will describe the background of guifi.net and current technologies used by the community. We will also explain approaches to make end user's life easier when connecting to the network, and what are the future challenges of the community.

In Chapter 2 we will introduce some concepts of mesh networking, its current state and implementations, and why it will be useful for our purpose. We will also explain routing protocols useful for this type of networks.

In Chapter 3 we will follow the development of a custom and generic firmware implementing all the functions we want for our purpose, introduce the used hardware and scenario, and describe the tests to conduct.

In Chapter 4 we will use the former firmware in the described scenario, and benchmark the nodes with simulated traffic to analyse how they behave.

In Chapter 5 there is a comparison of each protocol with the tests performed on the previous chapter, analysing which one performs better in each test.

Finally, in Chapter 6 there are the final conclusions of this thesis based on the performed tests and the recollected data.

Objectives

The main objective of this thesis is to analyse and compare the one of the most routing protocols used in mesh networking, and determine which one fits better in the guifi.net community.

Specifically, the goals to be accomplished are:

- Getting a general understanding of how ad hoc networking works, and how the specifically designed routing protocols try to solve the problemes inherent to this type of network.
- Configure and build an OpenWRT firmware image, able to build a mesh network easily, using different protocols.
- Getting a small working network using this image and highly-used hardware (i.e. embedded devices).
- Test each of this protocols differently, to determine which of them perform better.
- Discussing and getting a conclusion from the performed results.

CHAPTER 1. GUIFI.NET: DEPLOYING OPEN, FREE AND NEUTRAL NETWORK

1.1. What is guifi.net

Guifi.net is an open community aiming to promote the deployment of telecommunications networks, base on peer to peer connection agreements, since 2004. With over 7000 online nodes, and 7500 km of wireless network links (as of June 2009), it has become on of the largest networks of its type in the world.

Lately, it has become a non-profit NGO (Foundation), in order to be legally constituted, though it hasn't been a priority all along the years.

Guifi.net declares its network to be open, free and neutral. Open because all the information about how it works and its components is published, and because everyone interested is allowed to participate; Free because nobody owns all the infrastructure, no single or corporatised owner can ever impose unilateral conditions on others; and Neutral, because the extent of the peer to peer agreement is limited to the terms of connectivity only, and not the content.

1.2. Philosophy

Everyone who joins the guifi.net community, or wants to connect to the network, has to agree to the terms and conditions of the network. This terms are described in The Wireless Commons License[1]and ensures the freedom of the network.

The terms of use basically claim no warranty about the contents of the network, and no liability about the user's behaviour. It also ensures everyone is free to use the network for any purpose, free to know how the network and its components work, and free to use the network for any type of communications, and promote its usage.

1.3. What services does guifi.net offer

Guifi.net itself, as a foundation, or as an open network, does not offer any service within the network itself. That would break the Neutral point of view of the network.

Instead, guifi.net offers the distinct publications and information about the own network and its typology: how is it organized, what nodes are connected and how, what are the services available on the network, statistics about availability, and so on. Every bit of that information is publicly available at the website guifi.net.

[1]http://guifi.net/WCL_EN

In addition, the website has also a management of all the nodes and its IP addresses which ease the task of joining segments of network together, so that is very easy to create new nodes and connect them with others. Also, the configuration of the devices used to connect to the network can be automatically generated, to be uploaded into the devices.

That last ”autoconfiguration” application is called unsolclic, and works for the most common devices used in the network, described in the next section. In addition, its developers are very active, and usually implement new devices quickly. If not, all the software is open source, so everyone is welcome to modify it.

And last but not least, the guifi.net application offers a MapServer, integrated within the website, which offers a visual status of the network with all the nodes and their links.

1.4. What is currently used by guifi.net networks

Obviously, everyone is free to use whatever hardware and software want, as long as it fits guifi.net's philosophy, Wireless Commons License. That said, there are 'de facto' standards at the several technical levels of the network.

1.4.1. Hardware

We must differentiate between end users and backbone network, each one with different needs.

On the end user hand, the current most used hardware is Ubiquity NanoStation[2], a low-cost CPE, featuring an easy-to-use firmware, and with a polite design suitable for everyone. Its size especifically is what makes it an attractive device, as it can be installed virtually anywhere, from balconies to rooftops. Before that, Linksys WRT54GL was extensively used.

On the backbone network, Mikrotik's RouterBoards[3]are used across the network, featuring a great performance and an affordable price. Specifically, RB600A is very suitable to extend the backbone, being able to support up to 8 antennas.

Finally, the standard antennas are sector antennas for end user's links at the band of 2.4 Ghz (or even at 5 GHz), and panel antennas for the backbone link at the band of 5 GHz, no preference for any specific brand.

1.4.2. Software

Usually, the software used in both the end user and backbone hardware is the manufacturer's software. That is, AirOS for NanoStation's, and RouterOS for RouterBoard's.

[2] http://www.ubnt.com/products/nano.php
[3] http://www.routerboard.com/

Nevertheless, a lot of people claim this is not the open way to go, and open source should be used instead. A very good alternative is OpenWRT, an open source firmware fully customizable and supporting plenty of hardware (including NanoStation's and RouterBoards). We will use this software along our development, so we will talk about it later.

1.4.3. Network configuration

Although the whole guifi.net network began at the catalan county Osona, it is now spread on quite a lot of counties along Catalonia. This means that it has to be carefully organized so that there won't be routing problems at all.

The current way of working is that each node connected at the guifi.net network (and so visible at the guifi.net website) has a public address, from the range 10.0.0.0/16, and no NAT is performed, at least not publicly available – the application doesn't support them.

Adresses are assigned to each node following the policy of `freenetworks.org`, which intends to assign private addresses in a hierarchically and topologically valid way. In addition, each guifi.net zone administrator can assign an specific range of IP's to a zone, that the application will assign on each node. Specifically, the application assigns the following ranges:

- Client's links: /27 range, 32 addresses, netmask 255.255.255.224, inside the 10.0.0.0/8 private range
- Backbone's links: /30 range, 2 addresses, netmask 255.255.255.252, inside the 172.16.0.0/12 private range
- Private uses: whatever inside the 192.160.0.0/16 private range

As far as dynamic routing is concerned, there are two major protocols in use: OSPF and BGP, both vastly supported by RouterOS and other firmwares. Despite that, there seems to be a pattern by which OSPF is usually used on isolated networks, and when the link with other isolated networks is done using BGP and route summarisation.

1.5. The future

Over the past few years, the guifi.net network has been growing at a huge rate. Each year the number of nodes is twice the previous year, so a year by now we can be able to speak of over 15000 nodes, which is a lot.

The future challenges are to find an even more autoconfigurable method of connecting nodes to the network, minimizing the work to be done by the end user, and the necessary skills to do it.

In addition, the Foundation has recently joined the RIPE, the organisation which provides global Internet resources and related services (IPv4, IPv6 and AS number resources) to its members in Europe, the Middle East and parts of Central Asia. As a result, now guifi.net is able to route directly to the global Internet, and not via proxy servers, with public IP addresses.

Also, the foundation is making steps to spread optical fiber from an Internet Exchange Point (IXP) to the end-user home, using utility poles, thus achiving really fast transfer rates.

CHAPTER 2. AD HOC ROUTING

2.1. Routing

Routing is the process of selecting a path beween to nodes in a network, from a source to a destination, along which to send network traffic.

Routing protocols specify how these routes are created, updated or removed. In other terms, how routers communicate with each other, in order to make communication possible between the source and the destination.

There are many routing protocols, which can be classified according to the route selection strategy, into two major classes. This classification is based on the algorithm applied to calculate the routes.

2.1.1. Distance vector

A distance-vector routing protocol uses the Bellman-Ford algorithm to calculate paths. The basics of the algorithm is to calculate the direction and distance to any destination in a network. The cost of reaching that destination is calculated using various route metrics, and depends on the protocol. For instance, RIP uses the hop count of the destination, whereas IGRP uses information such as node delay and available bandwidth.

Every node maintains a table, called distance vector, and performs updates periodically where all or parts of its routing table is sent to all its neighbors. Each node trusts its neighbor, and no single node has a global view of the network. As a consequence, distance vector routing protocols have a low computational complexity and message overhead. They don't require a global broadcast, and in the case of large networks (such as mesh networks) is a very important advantage.

On the other side, the convergence time for propagating routing information is slower than in link state routing protocols, specially when the cost between links is high. As a consequence, many times there are routing loops in this type of networks, because a link change happened but didn't yet propagated to the whole network.

Examples of this type of routing protocol are RIPv1 and 2, IGRP, EGP and BGP (though EGP and BGP are not pure distance-vector routing protocols). B.A.T.M.A.N. is also one of the distance vector protocols.

Figure 2.1 below shows a simple scenario as an example of a distance vector routing protocol, where every node exchanges information about its distance to a given destination.

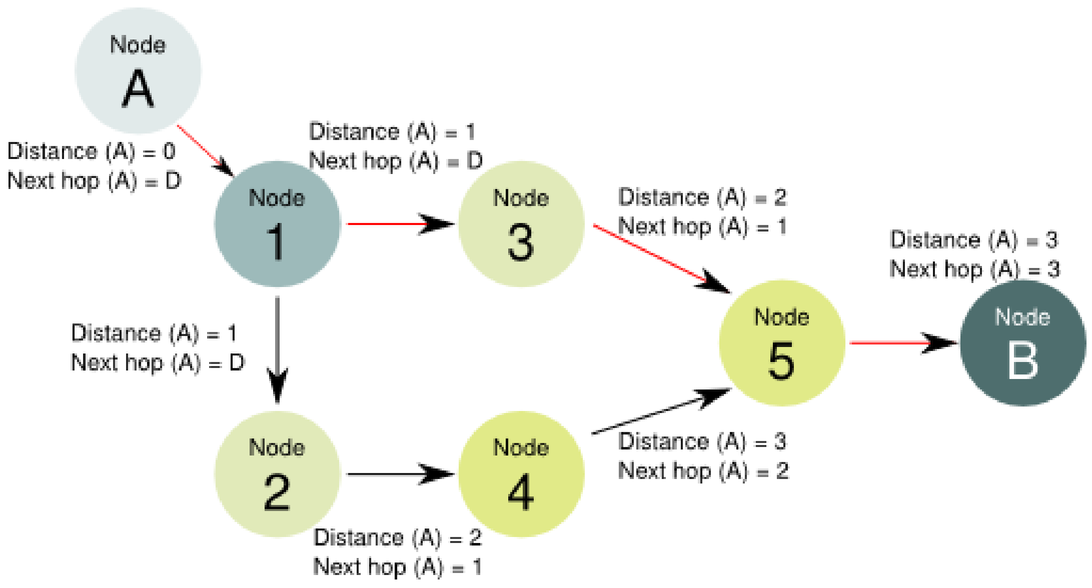

Figure 2.1: Diagram of an example distance vector routing protocol

As seen, for A to reach B the path with a lower distance is chosen, suposing the same cost for each link between nodes.

2.1.2. Link state

The basic concept of link-state routing is that every node maintains a full copy of the network topology, including the costs for all known links. This information is transmitted periodically to all nodes by flooding, and each node forms the routing table from the collection of best next hops.

Link-state algorithms solve looping problems associated with distance vector protocols, but require more complexity and the computational cost is higher.

Examples of this type of routing protocol are OSPF and IS-IS. OLSR is also one of the link-state protocols.

Figure 2.2 below shows a simple scenario as an example of a link state routing protocol, where every node exchanges information about its neighbours and the costs to getting to each. For the sake of simplicity, costs are not shown on this example, and are considered equal for each link.

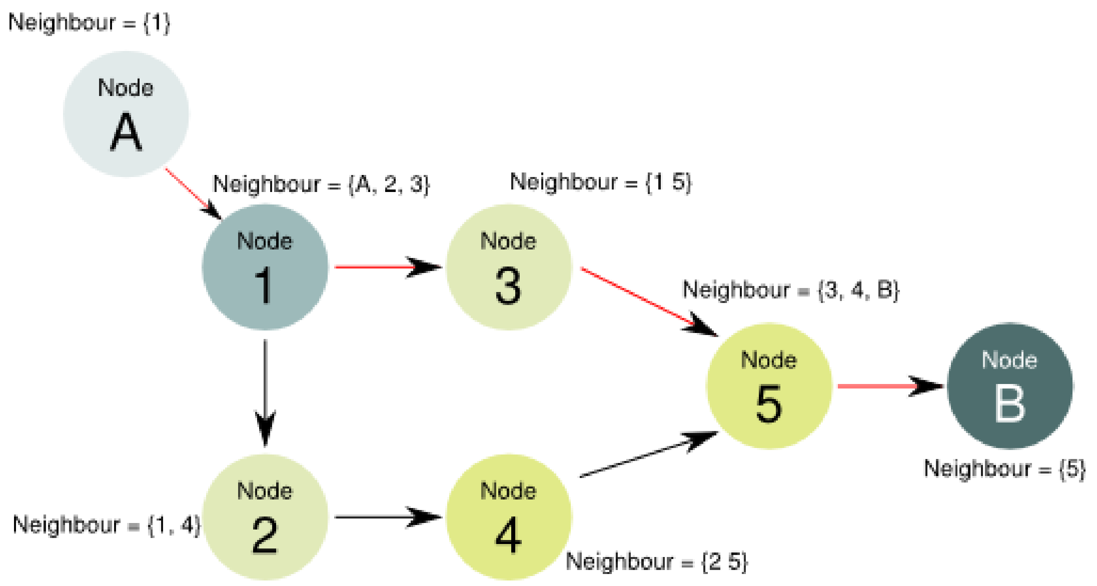

Figure 2.2: Diagram of an example link state routing protocol

Each node in this figure broadcasts its neighbours and costs to the network, and each node then forms its routing table considering the whole network.

2.2. Ad hoc networking

We can define wireless ad hoc network as a decentralized wireless network, where each node is able to forward data for other nodes equally, and so the paths this data will take are dynamically calculated based on the network connectivity.

This type of networks is in contrast to wired networks, in which there are special devices (called routers) which perform the task of routing. It also contrasts to managed wireless networks, in which it is the access point the device managing communications among nodes.

Wireless ad hoc networks can be further classified into two main types:

- Mobile Adhoc Network (MANET), in which the nodes conforming this type of network don't usually have a fixed place.
- Wireless Mesh Network (WMN), usually consisting of fixed nodes.

We will focus on the second type, wireless mesh networking.

2.3. Characteristics and limitations of mesh networks

In this section a summary of the main characteristics of mesh networks is presented. Mesh networks have several peculiar features which make them vastly useful at the expense of some limitations.

Self-organization: ad hoc network infrastructure lacks a centralized administration whose functioning is vital for the well-being of the network. Instead, all the nodes self-organize in a distributed way by which the addressing and routing of the network is self-determined. This fact, helps to improve the reliability of the network.

Multi-hop: A multi-hop network is one in which the path from source to destination goes through several nodes. Each node in a wireless ad hoc network communicates with the others via radio links. Therefore, multiple hops may be needed to reach others nodes due to their limited radio propagation range.

Wireless: Every comunication within an ad hoc network is carried over the air. Therefore, there exists a bandwidth restriction and variable link capacity, compared to wired networks. The changing wireless medium can lead to variable network characteristics (such as delays and bandwidth), and so it decreases reliability of the network.

Resource limitations: Usually, nodes used in an ad hoc network consist of embedded systems with little CPU, memory or storage capacity. That may decrease bandwidth capacities, and forces protocols used in ad hoc networks to keep processing of routes and packages as low as possible.

2.4. Mesh networking

Mesh networking is a relatively new technology originating out of ad hoc networking research from the early 90's. As a consequence, there is still an ongoing effort to find routing protocols which perform best in large static or quasi-static wireless mesh networks.

Most of the protocols used for mesh networking grew directly out of protocols used for MANETs, designed with mobility in mind. Examples of these protocols are Optimized Link State Routing (OLSR), Dynamic Source Routing (DSR) and Ad-hoc On Demand distance Vector (AODV).

Those protocols were designed, as we said, considering a constantly changing topology due to mobility of the nodes, and losses over the wireless medium. These special characteristics don't occur on mesh networks, where little or no mobility is expected and very little route fluctuation should happen.

Keeping that in mind, two protocols are being used here, B.A.T.M.A.N. and OLSR-NG, both specially designed to be fit in large wireless mesh networks. Below there is a brief description of what each protocol does and how they work.

2.5. B.A.T.M.A.N.

2.5.1. Introduction

As we said, B.A.T.M.A.N. is a routing protocol specifically built to solve the problem of large wireless mesh networks, and was born out of a response to the shortcomings of OLSR. It belongs to the proactive and distance vector type of wireless routing protocols.

The strategy of the B.A.T.M.A.N. 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 in the mesh. Thereby it is unnecessary to notify globally the local topology changes on the network.

In our tests, we will use the version 0.3.2, which uses the B.A.T.M.A.N. IV/TQ algorithm, featuring better handling for asymetric links and packet aggregation.

2.5.2. Overview

All nodes periodically broadcast hello packets, also known as OGMs, to its neighbors, informing its link-local neighors about its existence. Each originator message consists of an originator address, sending node address, and an unique sequence number. Each neighbor then changes the sending address to its own address and re-broadcasts the message, thus flooding the network with OGMs. These OGMs are very small, typically of 52 byte including IP and UDP overhead. The message keeps being broadcasted until it gets lost or its TTL value has expired.

In practice, OGM packet loss is significant, and is used to estimate the quality of a route. In order to be able to find the best route to a certain originator, B.A.T.M.A.N. counts the originator-messages received and logs which link-local neighbor relayed the message. Using this information B.A.T.M.A.N. maintains a table with the best link-local router towards every originator on the network.

2.5.3. B.A.T.M.A.N. OGM

The Originator Message format in acsB.A.T.M.A.N. IV is described at the table 2.1 below.

+	**00**	**01**	**02**	**03**
00-03	Version	Flags	TTL	GW Flags
04-07	Sequence Number		GW Port	
08-11	Originator Address			
12-15	Previous Sender Address			
16-19	TQ	HNA length	(...)	

Table 2.1: B.A.T.M.A.N. IV Packet format

2.5.4. B.A.T.M.A.N. IV/TQ algorithm

The main problem in previous versions of B.A.T.M.A.N. is asymetric links. For example, there is an asymetric link joining node A with node B: while the link from A to B is perfect (0% wrong packets), 95% of the packets in the link from B to A get lost. With B.A.T.M.A.N., this whole link is dismissed, while there is one usable link (from A to B).

In order to solve the problem with asymetric links and B.A.T.M.A.N. III algorithm, the Transmit Link Quality (TQ) calculates both the receiving and trasmitting link quality.

B.A.T.M.A.N. IV/TQ aknowlodges 2 parts in a given link quality: receiving and transmit link quality. Receiving link quality is the probability of a successful packet transmission towards the node. Transmit link quality is the probability of a successful transmission towards a neighbour node.

The more interesting part is transmit link quality, as we are trying to build the route tables, and the receiving link quality is pointless for this purpose. However, the Receiving Link Quality (RQ) is needed to calculate the Transmit Link Quality (TQ). We will also need the Echo Quality (EQ). All of them are schematized in the figure 2.3 below.

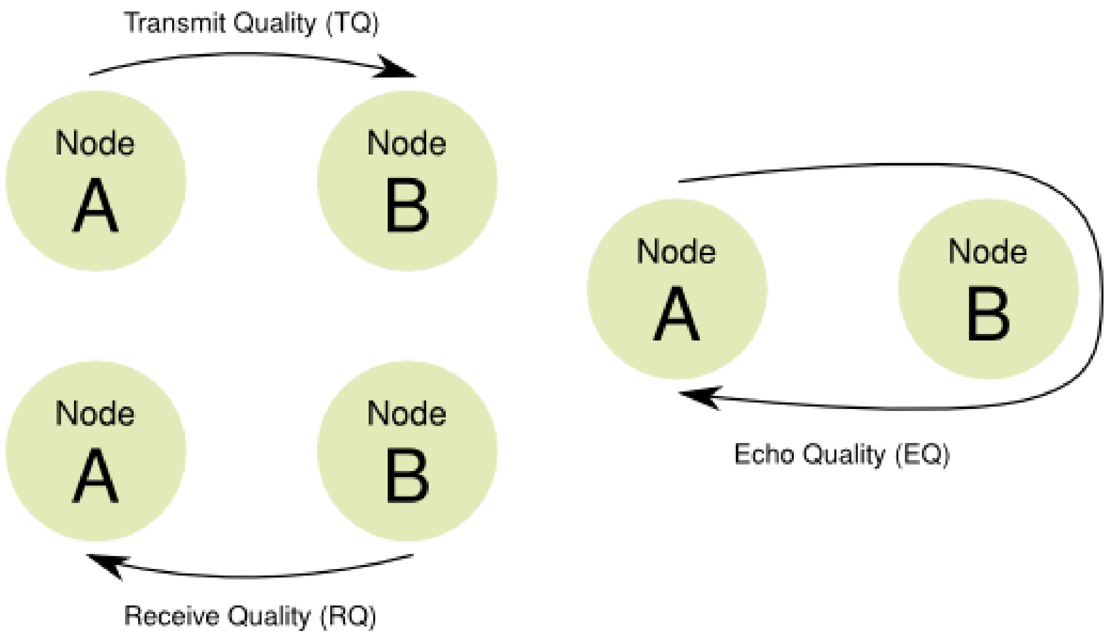

Figure 2.3: Different types of link quality

As the figure shows, the RQ is measured by counting the packets node A receives from node B. EQ is measured by counting rebroadcasts of its own OGMs from node B. Finally, B.A.T.M.A.N. calculates the Transmit Link Quality (TQ) dividing the echo quality by the receiving link quality. So we have: $TQ = EQ/RQ$

2.5.4.1. *Transmitting TQ*

As seen on the table 2.1, there exists a byte in the B.A.T.M.A.N. IV/TQ OGMs to transmit the TQ value throughout the mesh network. This way, each node rebroadcasts the TQ considering the local TQ with its neighbour. In addition, in the case that one node received two TQ to one node, it rebroadcasts only the best TQ to reach the node: that way, only the best path is transmitted.

2.6. B.A.T.M.A.N. Advanced

B.A.T.M.A.N. is usually a layer 3 protocol, but there is a different approach working at layer 2, called B.A.T.M.A.N. Advanced. Using the layer 2 approach, we will be able to run DHCP and retreive network configuration automatically.

Since it works using the layer 2, this protocol will not be object of the study, as we just want to test the other protocols, which are more similar between each other. B.A.T.M.A.N.-advanced is briefly explained here as a reference and because it is from the same family as B.A.T.M.A.N. and BMX.

2.6.1. Overview

B.A.T.M.A.N. Advanced is a protocol working on top of a standard layer 2 Ethernet header as described in the table 2.2 below.

	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**
00-07	Destination MAC						Source MAC	
08-15	Source MAC				Ethernet Type		Batman Type	(...)

Table 2.2: Ethernet Embedding

Where Ethernet Type is the Batman Ethertype (0x0842), Batman Type is one of the different Batman messages (unicast, broadcast, originator, message system, visualization), destination MAC is the MAC of the neighbour or broadcast and source MAC is the MAC of the NIC the packet is sent out from.

2.7. BMX (B.A.T.M.A.N. Experimental)

BMX is another branch of the B.A.T.M.A.N. protocol, and can generally be user in the same way as the batmand-0.3 branch. The concepts of the underlying algorithm are the same as all the B.A.T.M.A.N. family protocols, but it incorporates a number of extensions, and a complete rewrite of the data structure maintained by each node to keep track of received originator messages and identified routes.

It also offers a few additional information to a node when trying to find optimal metrics for selecting the best next hop towards the final destination of a packet. That can be easily seen in the output of the debug level one, which provides more information about the currently best-ranking neighbor.

Finally, the core routing algorithm of BMX can be fully parametrized, which enables developers to do some experimentation or fine tuning the algorithm. The so many options to parametrize the algorithm escape the aim of this text, and can be found at [2]

2.8. OLSR-NG

OLSR is another routing protocol, optimized for MANETs, which can also be used on WMNs. It belongs to the proactive and link-state type of wireless routing protocols. OLSR-NG (or OLSRv2) is the latest version of the OLSR protocol, nad includes several improvements since the very first version which help the MANET scale. The larger and more dense a network is, the more optimization can be achieved compared to the classic link state algorithm.

As all link-state routing protocols, a designated router is chosen on every link in order to perform flooding of topology information, called Multi-Point Relay (MPR). This router is responsible for re-transmitting all the broadcast messages that receives from its selectors, provided that the message is not a duplicate, and that the hop limit field of the message is greater than one.

2.8.1. Overview

OLSRv2 has the stability of every link state algorithm, in addition to the advantage of immediately available routes when needed because of its proactive nature. Some of the main optimizations of OLSRv2 against traditional link state protocol are:

- Partial topology maintenance: each router knows only a subset of the links in the network, enough to reach all destinations with a minimum hop route.
- MPR flooding for MANET-wide link state information distribution.
- All control messages are transmitted unacknowledgely and periodically, but may also be sent in response to changes in the local neighborhood.

The first and second improvements are based on the concept of MultiPoint Relays (MPRs). Each router in the network selects a set of MPRs. That MPRs may be any subset of symmetric 1-hop neighbors, such that every router in the symmetric 2-hop neighborhood has also a symmetric link to at least one of ours MPRs. That is graphically explained at figure 2.4 below:

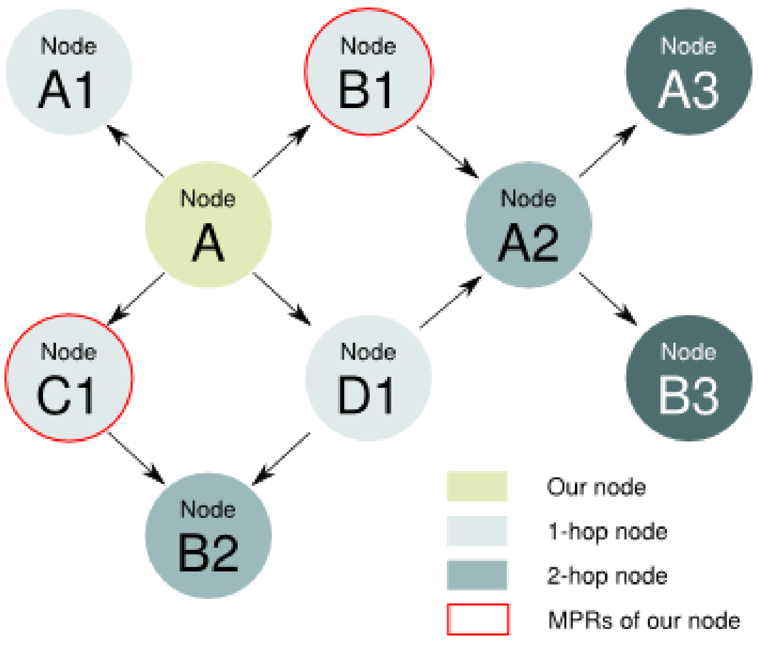

Figure 2.4: Explanation of different elements in an OLSR network

As seen, our router (Node A) chooses 2 MPRs, Node B1 and Node C1, such as any 2-hop node in the network has a symmetric 1-hop neighbor to at least 1 MPR. In the example, Node B2 has Node C1, and Node A2 has Node B1.

Thus, the MPRs of our router may be said to *cover* the router's symmetric 2-hop neighborhood. Each router also maintains information about the set of symmetric 1-hop neighbors that have selected it as an MPR, i.e. its MPR selectors.

Finally, a popular feature of the OLSR protocol is the ability to include external plugins, which enhance the protocol itself. Examples of plugins are multicast[1], dyn_gw[2], httpinfo[3], secure[4] or nameservice[5].

[1]http://sourceforge.net/projects/olsr-bmf/
[2]To dynamically add uplink gateways
[3]A tiny webserver for information purposes
[4]To secure OLSR routing with shared key
[5]To announce hostnames and DNS servers

CHAPTER 3. USED TECHNOLOGIES AND SCENARIO

3.1. Hardware

3.1.1. Ubiquiti NanoStation

As we said before, Ubiquiti NanoStation is used as a CPE for end users, and that hardware is what we are going to use. It is chosen because of the great popularity across the community: its price and even its *beauty* (it is a small device and fits well in many places) are two important factors for its popularity.

Figure 3.1: Picture of a NanoStation 2

Specifically, we will be using Ubiquiti NanoStation 2 (picture at figure 3.1), operating at the 2,4 GHz frequency. Its main specifications are, according to the manufacturer's datasheet[1]

- Processor: chip Atheros AR2315, 180 MHz MIPS
- Memory: 16 MB SDRAM, 4 MB Flash
- Wireless: Atheros 802.11 b/g, 400 mW
- Antenna gain: 10 dBi
- Power: passive PoE 12V, 1A

[1] `http://www.ubnt.com/downloads/ns2_datasheet.pdf`

3.2. Software

3.2.1. OpenWRT

As taken from its website[2], OpenWRT is a Linux distribution for embedded devices.

It has become the most popular Linux distribution, and nowadays it supports many hardware devices, including the one we will use, Ubiquiti Nanostation 2.

Due to its popularity, there are many packages included which can be easily compiled to generate an specific firmware image. Along these packages, there are the B.A.T.M.A.N. and OLSR-ng routing protocols, which we'll be using in the paper.

3.3. Scenario

Our common scenario will be 5 NanoStation with static IPs, and the different routing protocols to compare. Specifically, the tested versions of each protocol are:

- B.A.T.M.A.N. 0.3.2, from the trunk, revision 1289
- BMX, taken from a latest version available at the GraciaSenseFils website[3]
- OLSR-NG 0.5.6-r4

3.4. Tests

Using our scenario, we will perform some tests to measure the effectivity of each protocol, as well as some aspects such as the convergence time of a node to the mesh. The list of performance metrics is below:

- Round-trip time
- Jitter
- Probability of error
- Bandwidth test
- CPU utilization

[2] http://www.openwrt.org
[3] http://merry.biruji.org/gsf/bmx-180.tar.gz

They are briefly described below, if necessary. Our main testing tool will be iperf. Bandwidth tests include UDP and TCP throughtput. The same tests will also offer us UDP and TCP packet loss, UDP jitter and TCP unordered packets and re-transmissions. We will also use the popular *ping* utility and *sar*, a system activity collection and reporting tool found in the sysstat utilities package, able to collect and report information on CPU and network interface activity over a period of time.

The tools, commands and parameters used to perform these tests are described at Annex C on page 63.

3.4.1. Round-trip time

The round-trip time (RTT) is the time it takes for a packet to reach a remote host and return back. It is related to the latency of the connection. Low RTT is better for these metrics.

The tests specifically measure the two-way **ICMP!** RTT

3.4.2. Jitter

Jitter is the variation in the latency of packets received by a remote host. For applications with streaming connections, jitter can be alleviated by buffering the stream. However, this adds delay in the connection which is intolerable for some applications such as Voice over Internet Protocol (VoIP). Low jitter is better for these metrics.

The tests specifically measure the one-way UDP jitter.

3.4.3. Probability of error

Errors can sometimes occur in network communication causing packets to be lost, corrupted, duplicated, or out of order. When an error occurs, it is important to know the probability of it happening again, and the time between errors. A related metric is packet loss which gives the percentage of packets that were lost or corrupted. No errors are ideal, but low error rate is acceptable.

CHAPTER 4. TESTS AND RESULTS

4.1. Scenario

The used scenario is described at the figure 4.1 below.

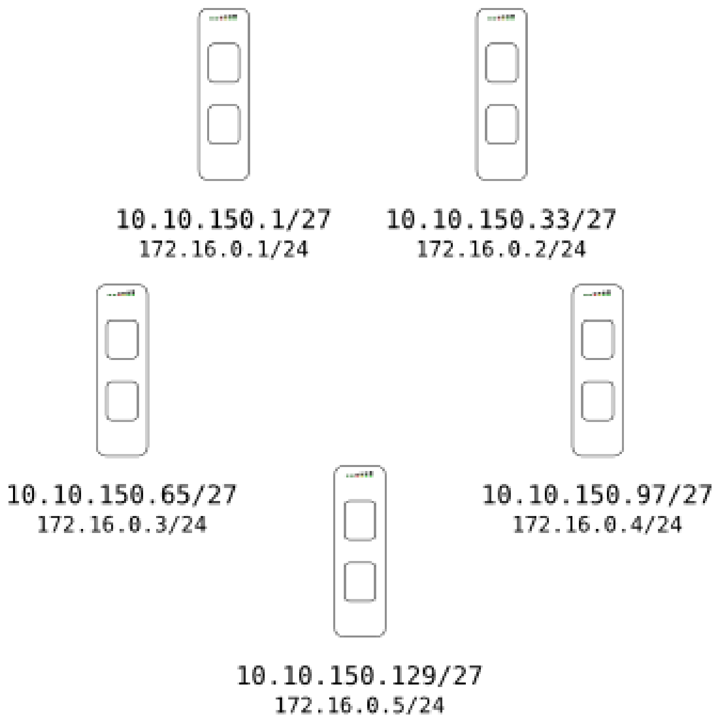

Figure 4.1: Testing scenario with static IPs

As we can see, each NanoStation has 2 IP addresses, a public (from the 10.0.0.0/8 range) and a private one (from the 172.16.0.0/12 range).

The public address is the globally unique to the guifi.net network, announced by each protocol, and the private address is the address which the routing protocol uses. Although it may not be necessary when just testing the efficiency of routing protocols, it is just a good habit to do it this way.

4.2. Real capacity of the network

All the performed tests were done with the instances of iperf running on the NanoStation2 routers. This means that the first NanoStation2 was always playing the role of the iperf server, while depending of the number of hops each NanoStation2 of the network was the iperf client as well.

This means that in addition to the routing tasks, these 2 NanoStation in particular have also the task of generating and receiving traffic.

Therefore, prior to testing the scenario the first test will be of how the network performs in a 1-hop mesh network and the iperf client and server outside the mesh network, connected via cable with the NanoStation2.

The scenario is shown in the figure 4.2 below:

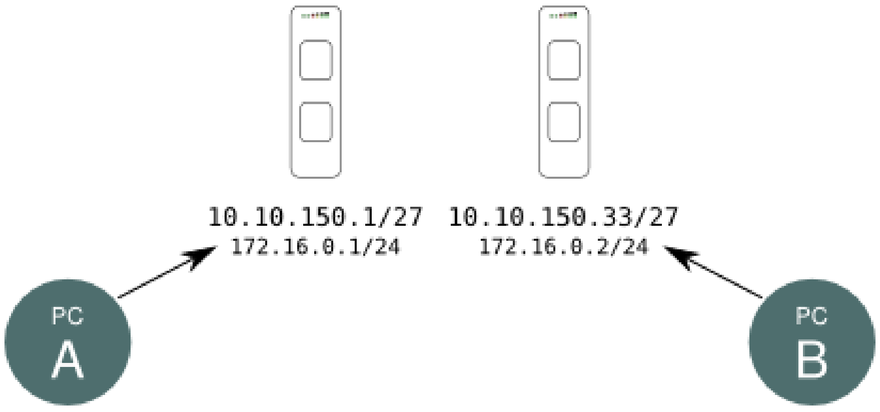

Figure 4.2: Testing scenario with external iperf client and server

where node A is the iperf server and node B is the iperf client. This way, the only task of NanoStation2 is to maintain the routing table and the daemon process (for each protocol).

When performing both UDP and TCP bandwidth tests, the results where almost the same in each tested protocol (i.e. batman, bmx and olsr).

In the case of UDP bandwidth, the maximum bandwidth the network allows is 26.5 Mbps

On the other hand, with the TCP bandwidth tests the results are of 16.5 Mbps.

4.3. Tests results

4.3.1. ICMP RTT

To measure the RTT value of each protocol we will use the ping utility, described on section C.1.. We will measure the RTT of different packet sizes (64 and 1024 bytes) from one node to each of its neighbours, evaluating the impact of the number of hops in each test.

4.3.1.1. B.A.T.M.A.N.

Figure 4.3 below shows the results of the obtained RTT values using the B.A.T.M.A.N. protocol. The results show a pattern where RTT increases with both size and number of hops.

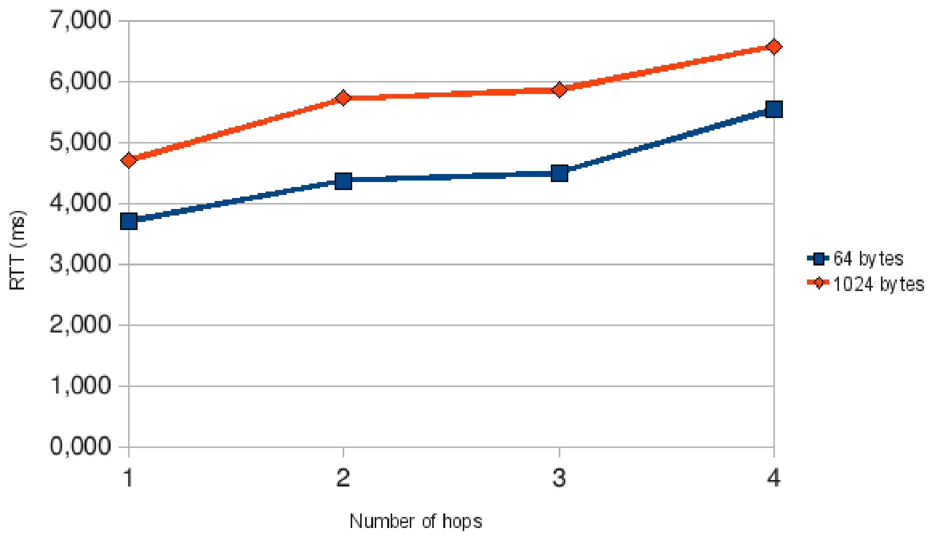

Figure 4.3: B.A.T.M.A.N. evaluation of Round-trip Time

4.3.1.2. BMX

Figure 4.4 below shows the results of the obtained RTT values using the BMX protocol. The results show, as with the B.A.T.M.A.N. protocol, a pattern where RTT increases with both size and number of hops.

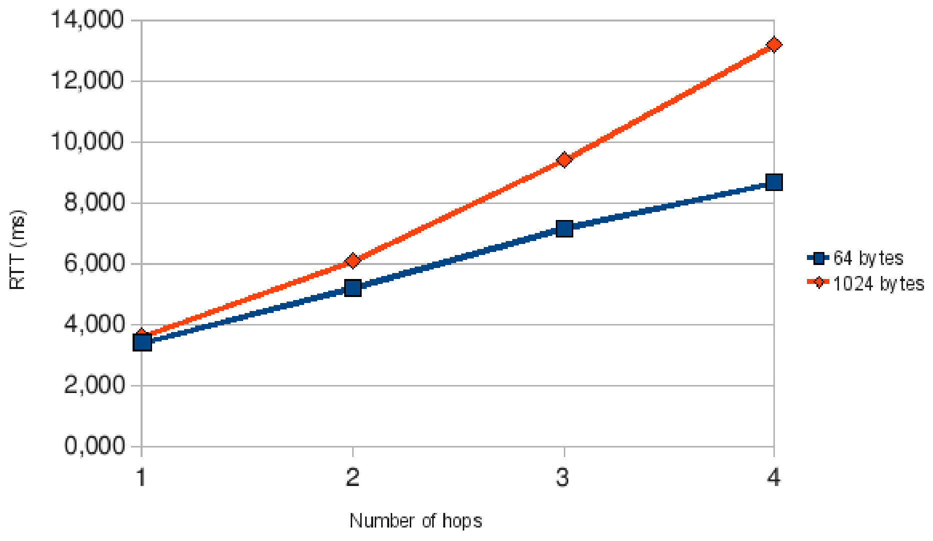

Figure 4.4: BMX evaluation of Round-trip Time

4.3.1.3. OLSR

Figure 4.5 below shows the results of the obtained RTT values using the OLSR protocol. The results also show, as with the previous protocols, a pattern where RTT increases with both size and number of hops.

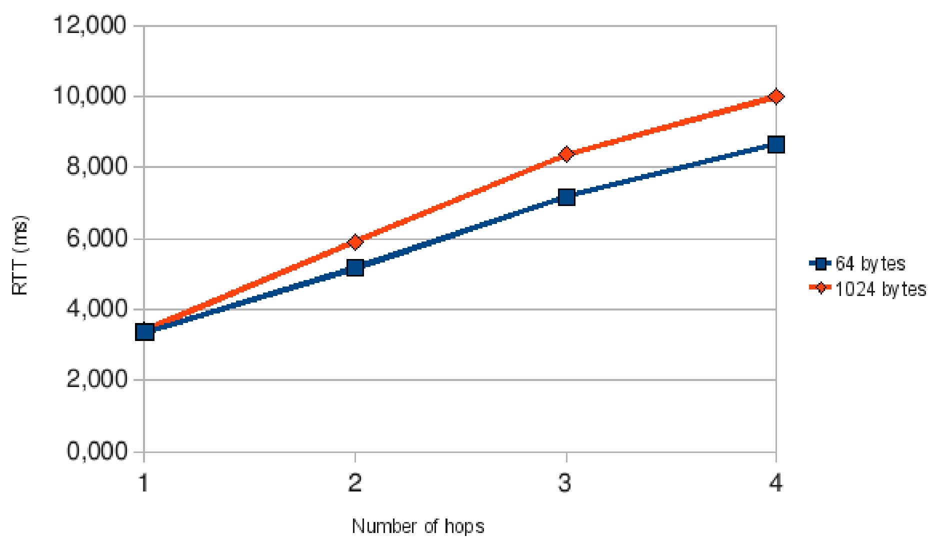

Figure 4.5: OLSR evaluation of Round-trip Time

4.3.2. UDP bandwidth

To measure the UDP bandwidth of each protocol, we will use the iperf tool as described on section C.2.1.. From one node we will offer bandwidths of 0.1 Mbps, 1 Mbps, 2 Mbps, 4 Mbps and 8 Mbps to each of its neighbours, evaluating the impact of the offered load and number of hops to the received throughput.

4.3.2.1. B.A.T.M.A.N.

Figure 4.6 below shows the results of the obtained UDP tests using the B.A.T.M.A.N. protocol.

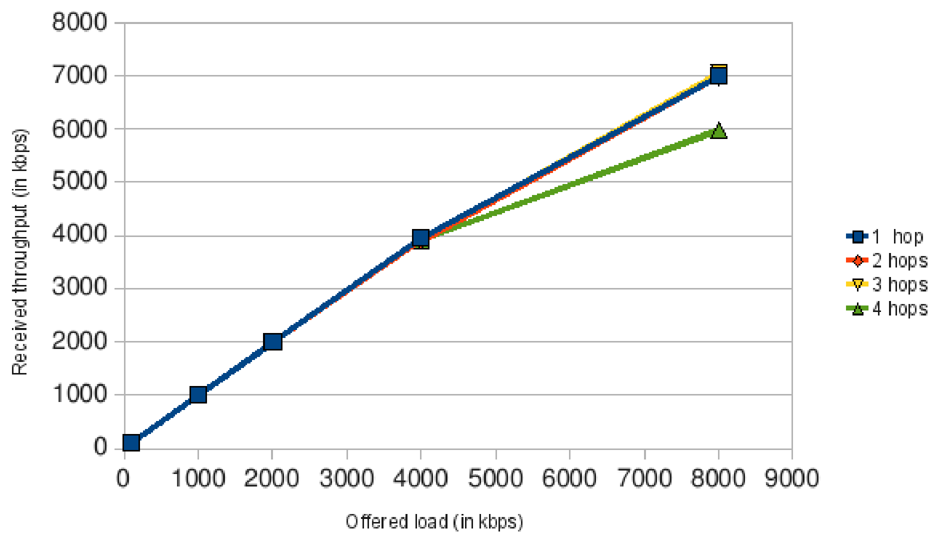

Figure 4.6: B.A.T.M.A.N. evaluation of UDP bandwidth

As we can see, there are no major differences until reaching the 8 Mbps load. With that load, the farest node experiments significant losses with respect of the other neighbors. Also, all the 8 Mbps tests show that the received throughput is lower than the offered load.

4.3.2.2. BMX

Figure 4.7 below shows the results of the obtained UDP tests using the BMX protocol.

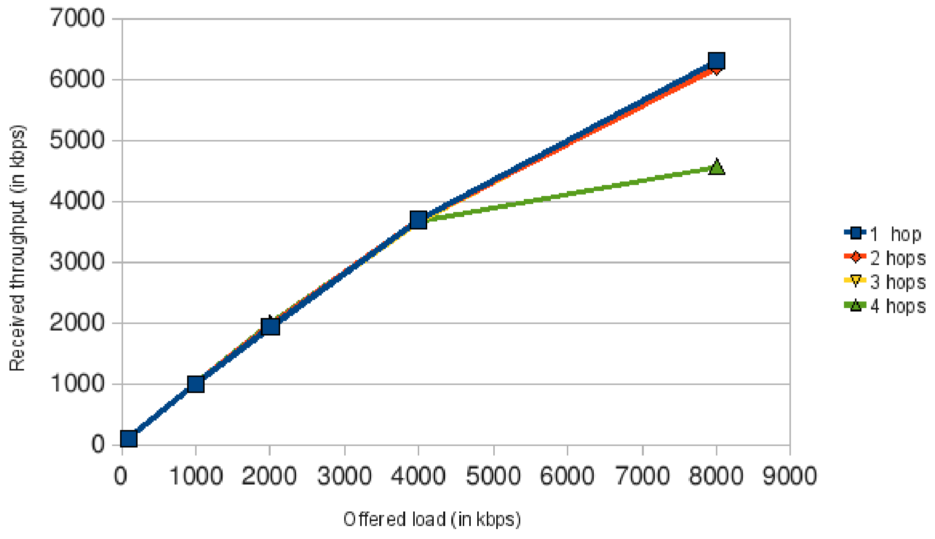

Figure 4.7: BMX evaluation of UDP bandwidth

As the figure shows, and similarly to what happened with the B.A.T.M.A.N. protocol, the bandwidth decreases significantly at the point of 8 Mbps offered load. This time, there is also a decrease in bandwidth when transmitting 4 Mbps.

4.3.2.3. OLSR

Figure 4.8 below shows the results of the obtained UDP tests using the OLSR protocol.

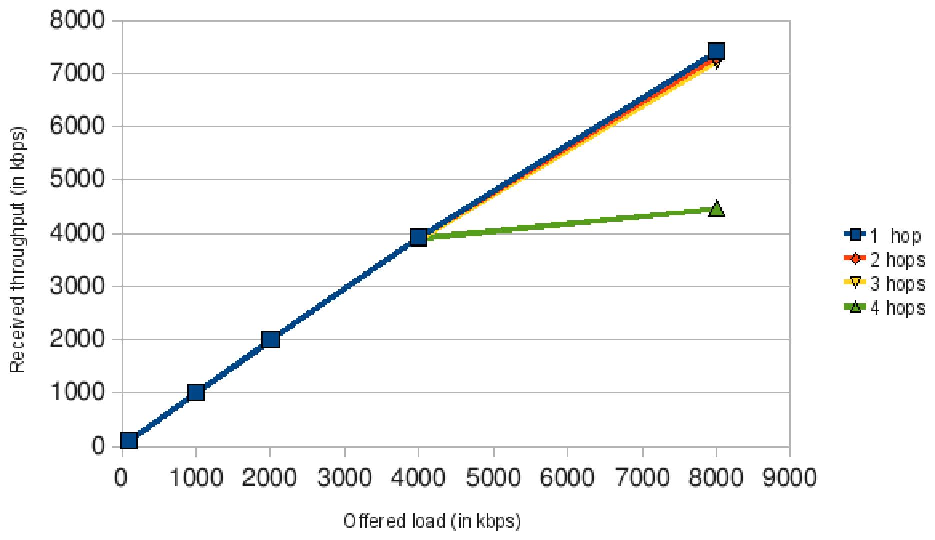

Figure 4.8: OLSR evaluation of UDP bandwidth

Again, the figure shows how the 8 Mbps looses efficiency. In this case, the 4-hop neighbour is the one suffering a bigger loss, but the 1, 2 and 3-hop neighbours receive almost the same bandwidth as the offered load.

4.3.3. TCP bandwidth

To evaluate the TCP bandwidth of each protocol, we will use the iperf tool as described on section C.2.2.. In this case, we can't offer a fixed load, so the tests will be limited to how much can TCP offer.

4.3.3.1. B.A.T.M.A.N.

Figure 4.9 below shows the results of the obtained TCP tests using the B.A.T.M.A.N. protocol.

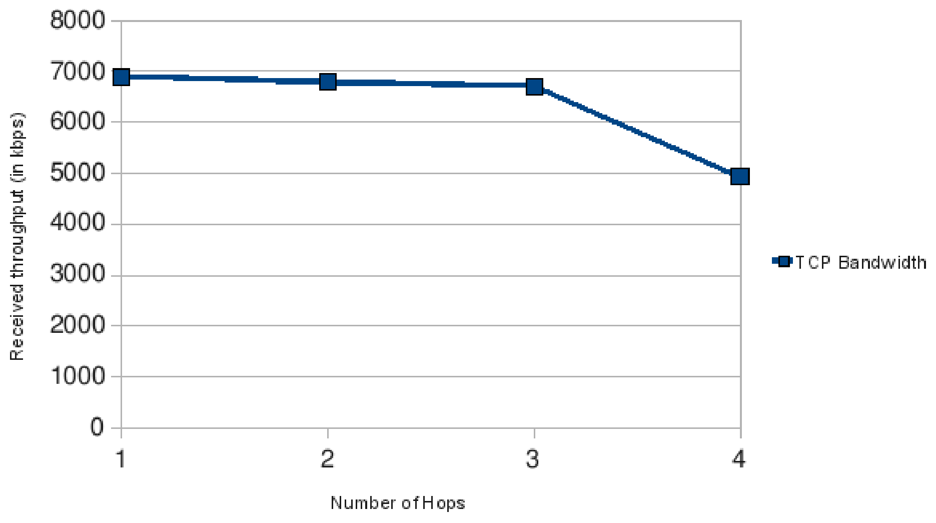

Figure 4.9: B.A.T.M.A.N. evaluation of TCP bandwidth

Just like what UDP bandwidth tests showed, the 4-hop neighbour suffers a decrease in the available TCP bandwidth, while the other neighbours all have a similar value.

4.3.3.2. BMX

Figure 4.10 below shows the results of the obtained TCP tests using the BMX protocol.

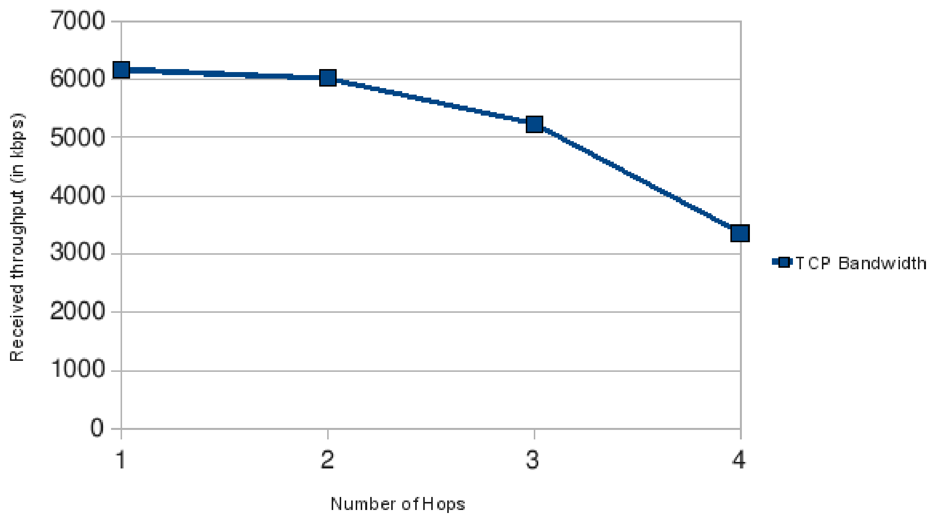

Figure 4.10: BMX evaluation of TCP bandwidth

Again, the bandwidth decreases as the number of hops increases. This time, the 3-hop neighbor experiences a decrease in bandwidth too.

4.3.3.3. OLSR

Figure 4.11 below shows the results of the obtained TCP tests using the OLSR protocol.

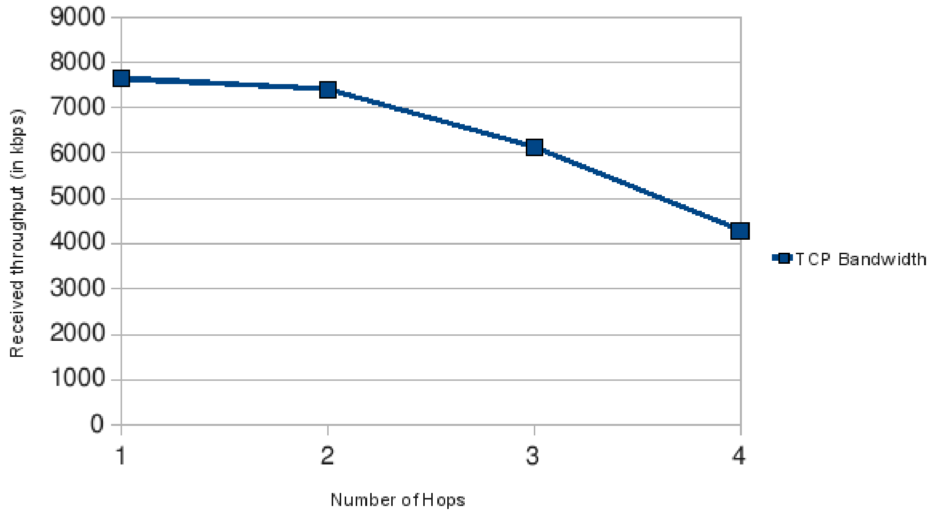

Figure 4.11: OLSR evaluation of TCP bandwidth

And again, TCP bandwidth decreases as the number of hops increases. Similar to the BMX behaviour, the 3-hop neighbor is the first to suffer a decrease in bandwidth.

4.3.4. Jitter

To evaluate the jitter each protocol, we will use the iperf tool as described on section C.2.1.. The same tests as with the UDP bandwidth tests are used, as iperf also shows the jitter of each test. Therefore, the measured jitter is dependant of the offered (UDP) load.

4.3.4.1. B.A.T.M.A.N.

Figure 4.12 below shows the results of the jitter value using the B.A.T.M.A.N. protocol.

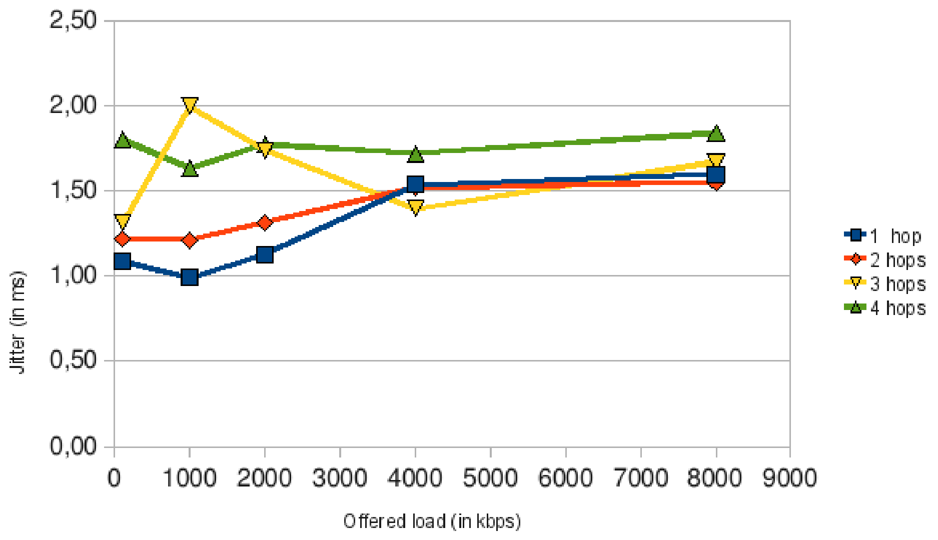

Figure 4.12: B.A.T.M.A.N. evaluation of Jitter

As the figure shows, there is no clear pattern with the jitter values. It seems that, mostly, the bigger the number of hops, the bigger the value of jitter. However, the maximum value of the measured jitter is 2 ms, which is low enough for all uses.

4.3.4.2. BMX

Figure 4.13 below shows the results of the jitter value using the BMX protocol.

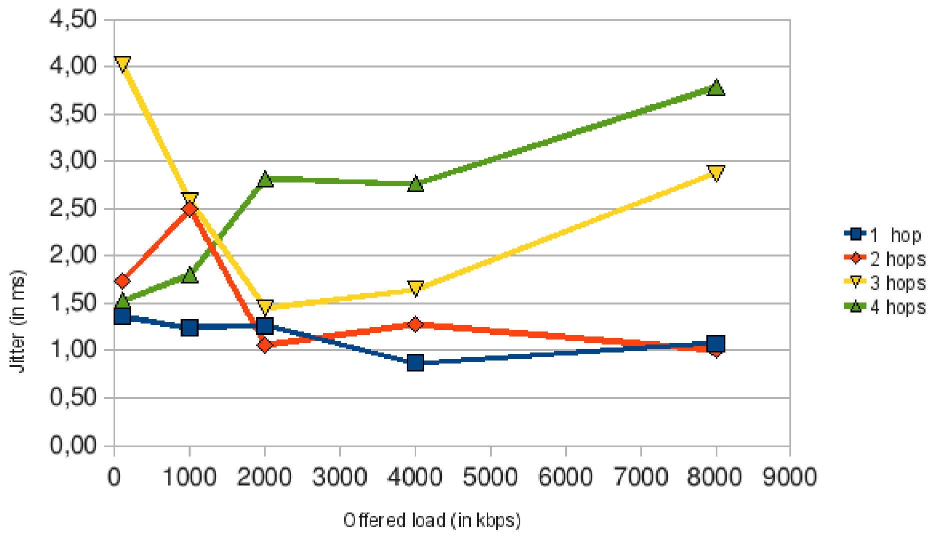

Figure 4.13: BMX evaluation of Jitter

Again, there isn't any clear way of relating the number of hops or offered load to the jitter value. As before, it seems that there is a tendency where the bigger the load and offered load, the bigger the jitter value is.

4.3.4.3. OLSR

Figure 4.14 below shows the results of the jitter value using the OLSR protocol.

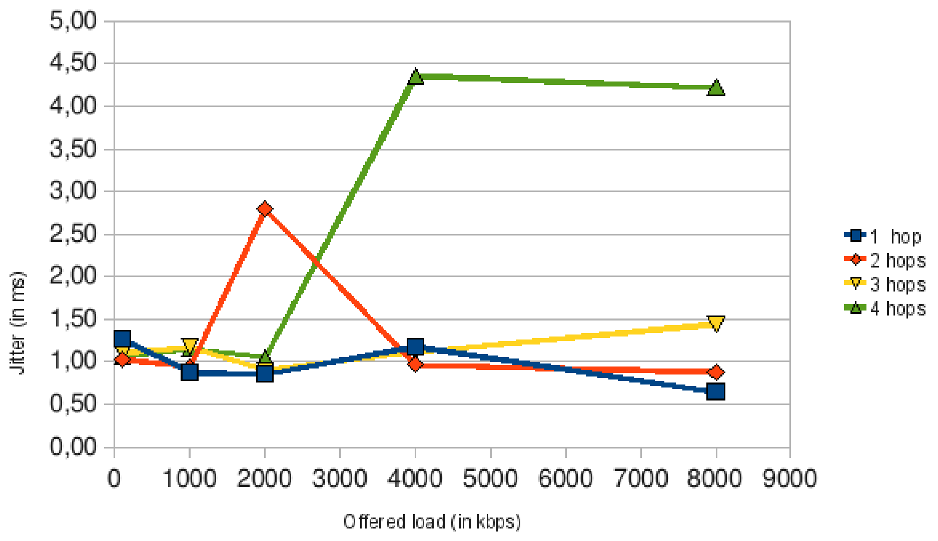

Figure 4.14: OLSR evaluation of Jitter

This time again, as the figures before, the jitter value seems to depend more on the instantanious environmental conditions that the number of hops or offered load.

4.3.5. Probabilty of error

As with the jitter tests, we will use the iperf tool to evaluate the probability of error, as described on section C.2.1.. The same tests as with the UDP bandwidth tests are used, as iperf also shows the jitter of each test. Therefore, the probability of error is dependant of the offered (UDP) load.

4.3.5.1. B.A.T.M.A.N.

Figure 4.15 below shows the results of the probabilty of error value using the B.A.T.M.A.N. protocol.

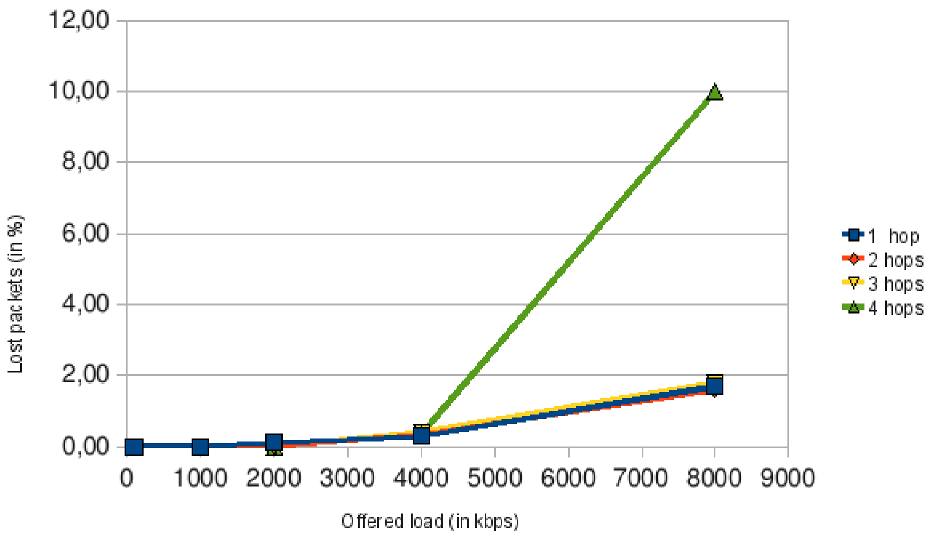

Figure 4.15: B.A.T.M.A.N. evaluation of Probability of errors

As the figure shows, there is almost no lost packets until the offered load is 4 Mbps, and specially 8 Mbps. The figure also shows that the lost packets are significantly higher with the 4-hop neighbor.

4.3.5.2. BMX

Figure 4.16 below shows the results of the probabilty of error value using the BMX protocol.

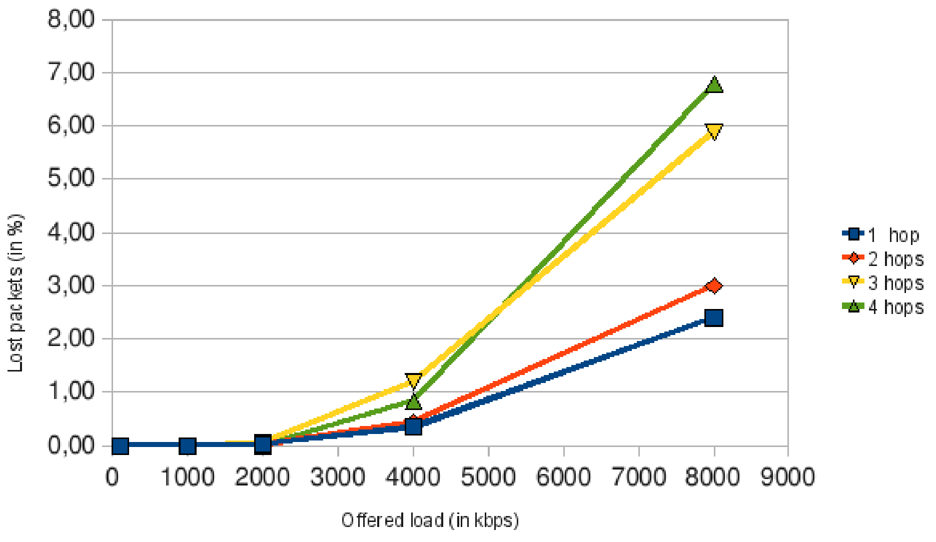

Figure 4.16: BMX evaluation of Probability of errors

This time, the results of the tests show that the 3 and 4-hop neighbours have bigger lost packets when the offered load is 8 Mbps. In general, the probabilty of errors is null until the offered load overcomes 4 Mbps.

4.3.5.3. OLSR

Figure 4.17 below shows the results of the probabilty of error value using the OLSR protocol.

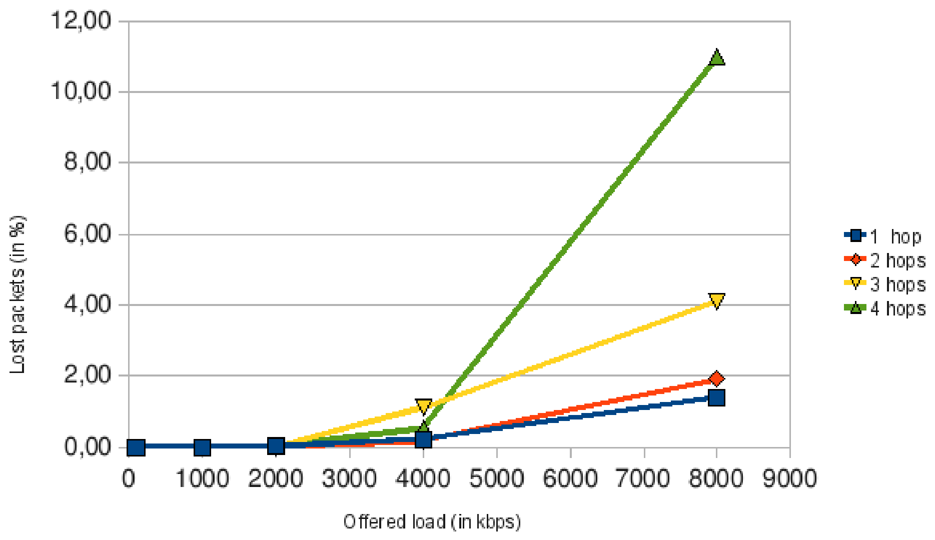

Figure 4.17: OLSR evaluation of Probability of errors

As before, the lost packets rate is significant when the offered load is 4 Mbps or above. Again, the worst result is with the 3 and 4-hop neighbours.

4.3.6. CPU utilization

To evaluate CPU utilization, we will use several tools, described at section C.3.. Unfortunately, each protocol is evaluated differently, so comparisons can't be made confidently.

Therefore, the performed tests are based uniquely on the routing daemon CPU utilisation, regardless of other processes.

4.3.6.1. B.A.T.M.A.N.

In this case, we just have the `top` tool output, which shows a CPU usage between 4% and 5% most of the time.

4.3.6.2. BMX

Running `bmxd -lcd8` shows the instant CPU time. This time changes from 50/1000 to 80/1000, so it never reaches the 1% usage of CPU.

However, `top` tool output is considerably higher and shows a CPU usage between 5% and 8%.

4.3.6.3. OLSR

Running `time` for 10 minutes and 13 seconds, gives the following results:

```
User time (seconds): 1.63
System time (seconds): 6.80
Percent of CPU this job got: 1%
Elapsed (wall clock) time (h:mm:ss or m:ss): 10m 13.00s
```

This matches the `top` tool output, which shows an average of 1% CPU usage.

CHAPTER 5. COMPARISON OF RESULTS

This chapter presents the comparison of results between each protocol, in addition to the discussion of the results in general described in the previous chapter.

5.1. Comparison between B.A.T.M.A.N., BMX and OLSR

This sections contains all the interessant comparisons of results between each protocol. Not all the results are worth comparing to. For instance, jitter is not important enough, and as it was shown it doesn't follow a clear pattern. Also, the UDP bandwidth tests are not relevant, since they show the same results for each protocol.

5.1.1. UDP bandwidth

To compare the UDP bandwidth, we only take into account the 3 and 4-hop neighbour, which are the ones which showed more differences. We also just want the 4 and 8 Mbps offered load, which we saw in the previous chapter were the interesting results.

Figure 5.1 below shows the differences between protocols in UDP bandwidth in a 3-hop neighbour transfer.

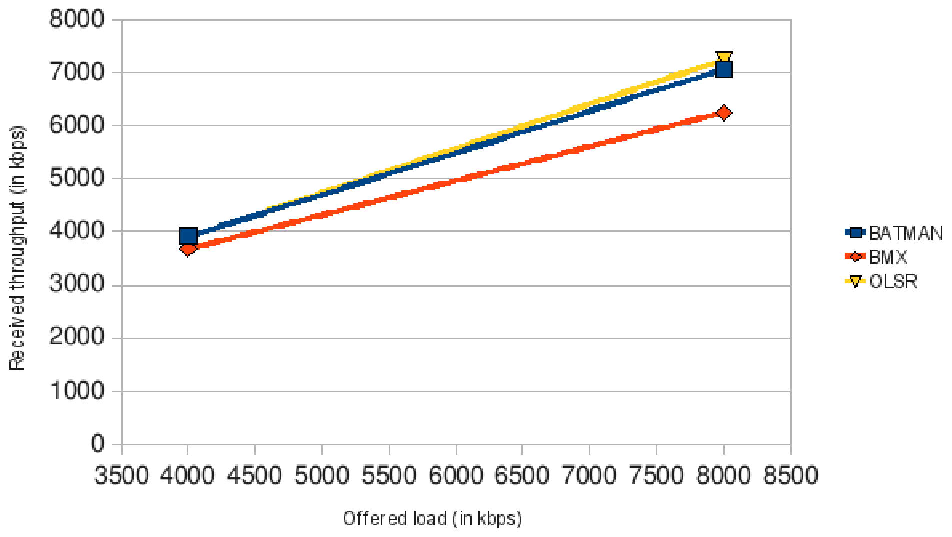

Figure 5.1: Comparison of UDP bandwidth to a 3-hop neighbour for each protocol

As the figure shows, both B.A.T.M.A.N. and OLSR perform very similarly, and above BMX.

On the other hand, figure 5.2 shows the same differences now in a 4-hop neighbour transfer.

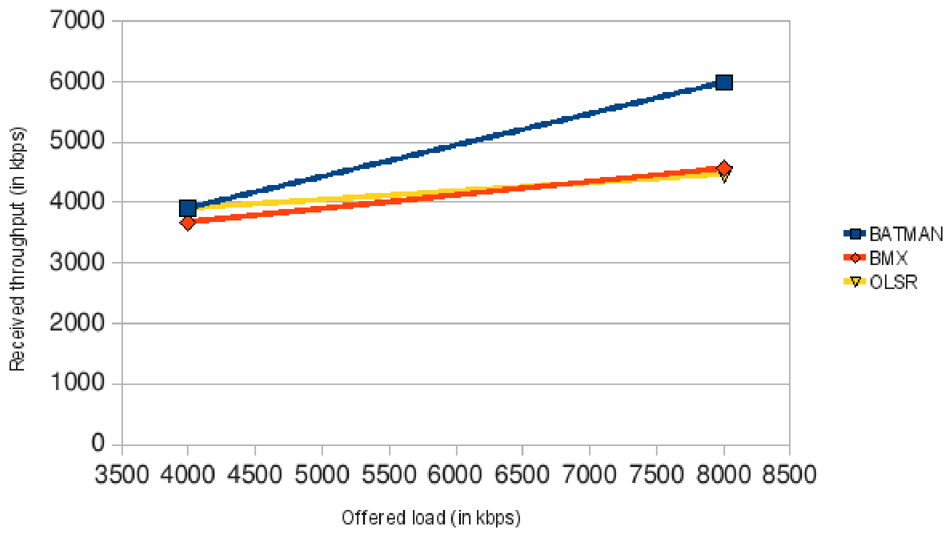

Figure 5.2: Comparison of UDP bandwidth to a 4-hop neighbour for each protocol

In this case, only the B.A.T.M.A.N. protocol outstands in efficiency, and shows better results than BMX and OLSR protocols.

5.1.2. Probability of error

Again, as before, we only take into account the 3 and 4-hop neighbours in order to compare the distinct probabilities of error between protocols. As before, only the 4 and 8 Mbps are shown. We also use 2 different figures.

Figure 5.3 below shows the probabilities of error of each protocol using an UDP transfer.

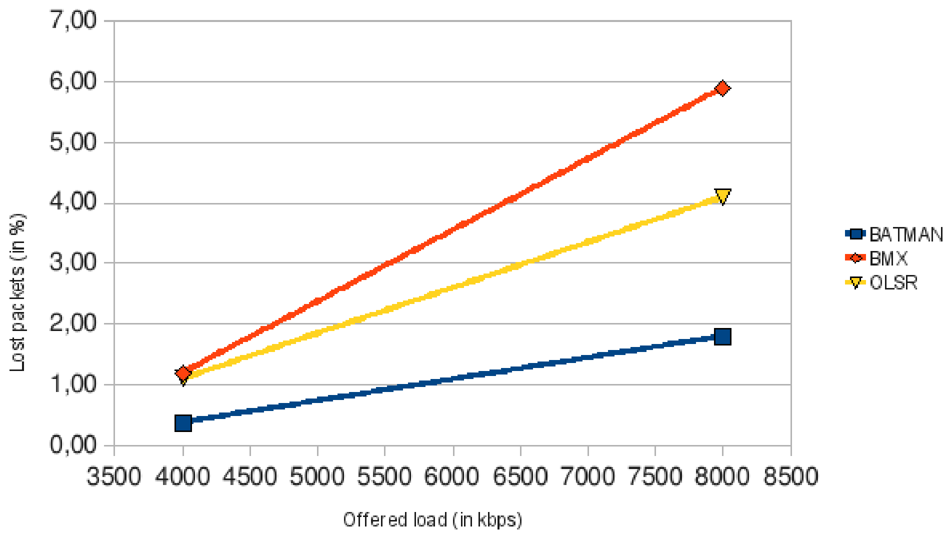

Figure 5.3: Comparison of lost packets to a 3-hop neighbour for each protocol

As figure above shows, in this case B.A.T.M.A.N. is the protocol which behaves better, with little more than 2% of the packets lost. On the other hand, BMX is the worst, followed by OLSR, with nearly 6% and 4%, respectively.

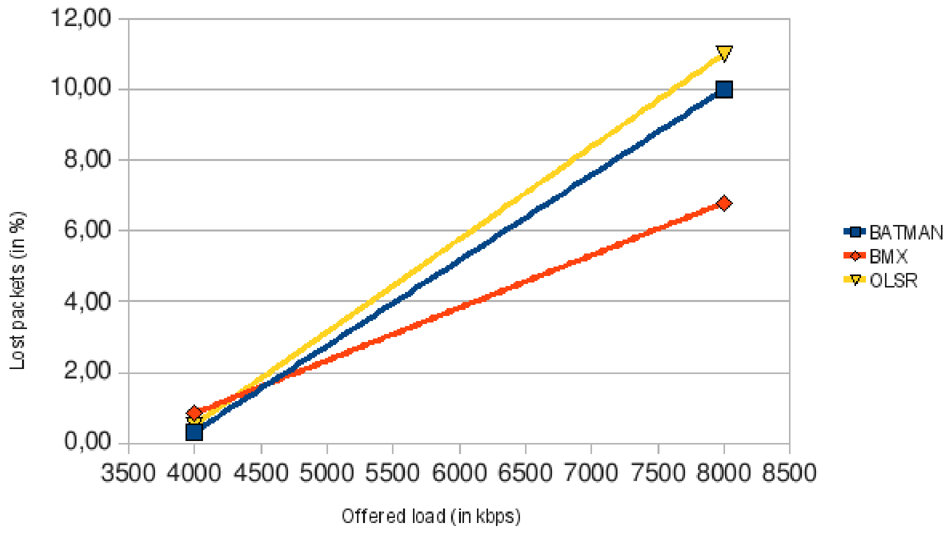

Figure 5.4: Comparison of lost packets to a 4-hop neighbour for each protocol

In this case, and as figure above shows, it is BMX the best protocol, with nearly 6% of lost packets. Again, OLSR is the worst, now followed by B.A.T.M.A.N. with 11% and 10%, respectively.

5.1.3. TCP bandwidth

Easier than before, we now take into account all the hops and just focus on the received throughput for each protocol.

For that, we use figure 5.5 below.

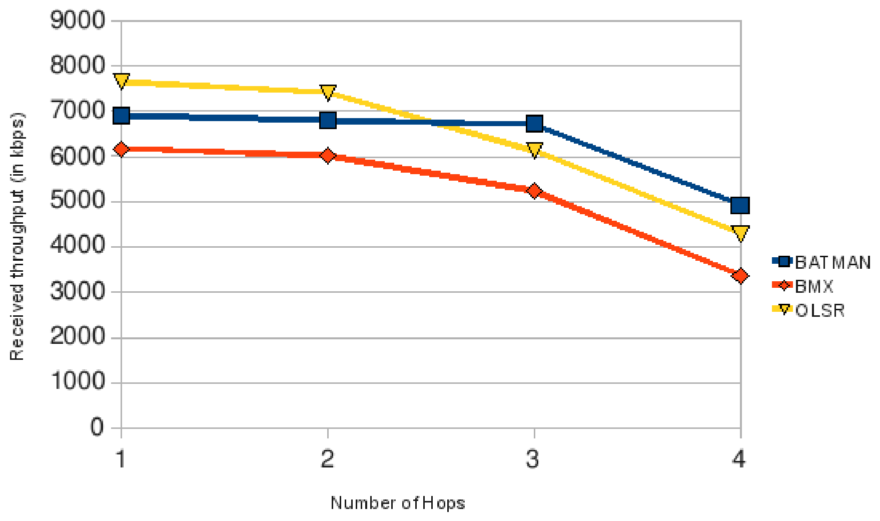

Figure 5.5: Comparison of TCP bandwidth for each protocol

As we can see, BMX is clearly the protocol which behaves worst of all, whatever number of hops it is.

On the other hand, B.A.T.M.A.N. behaves better than OLSR in 3 and 4-hop neighbours, and viceversa, OLSR has better received throughput when it comes to 1 and 2-hop neighbours.

CHAPTER 6. CONCLUSIONS

This thesis presents a study of the performance of several protocols in multi-hop mesh networking, i.e., B.A.T.M.A.N., BMX and OLSR. It presents the empirical results of several metrics using the same scenario.

The presented scenario is not a very realistic real world network, since it is formed of just 5 nodes. It is, however, real tests and not simulations the ones which were carried out, so the results, small as they can be, intend to be as reliable as they can be. When dealing with such environments, the experiments are more difficult to repeat each time exactly as the one before.

Therefore, there is not really a winning protocol, i.e. the one performing better, as in a realistic real world implementations many factors should be also considered, such as the size of the network, the interferences of other radio signals, and the different hardware used in those networks.

However, in the light of the results there indeed are some interesting findings, maybe not directly connected to the main purpose of this document – comparing protocols.

First, OpenWRT is a perfectly valid software which can be used in this and many hardware to support whatever services we want. It is perfectly suitable in the guifi.net community, and very interesting to explore deeply. It can also be easily extended to perform community-specific functions which could not be done using private software.

Second, Ubiquiti NanoStation2 is a perfectly valid – community proved that – hardware when it comes to Infrastructure (Server-client) model. However, there is a major inconvenient which is the CPU utilisation. As seen, the bandwidth decreases drastically when the iperf is running on the same NanoStation, so the hardware is not able to support the bandwidth and the routing at the same time.

Third, and last, each protocol performs well in the studied scenario. Main limitations in the scenario come from the hardware. Even it is true some protocols behave better than others (B.A.T.M.A.N. being the best), it is also true that we couldn't test on bigger scenarios, where specially CPU consumption in calculating the routes is very important.

BIBLIOGRAPHY

[1] Simon Wunderlich Marek Lindner Wesley Tsai, *B.A.T.M.A.N. Advanced*
http://gitorious.org/batman-adv-doc

[2] Axel Neumann, *Reference Manual of B.A.T.M.A.N Experimental*
December 10, 2007 http://downloads.open-mesh.net/batman/misc/bmx.pdf

[3] *IETF MANET OLSRv2-08 draft, The Optimized Link State Routing Protocol version 2*, March 09, 2009
http://tools.ietf.org/pdf/draft-ietf-manet-olsrv2-08.pdf

[4] *B.A.T.M.A.N.: Getting behind the routing vodoo*
http://open-mesh.net/wiki/RoutingVodoo

[5] Joan Llopart, *Com fer un OpenWRT per la nanostation2 des de zero*
http://www.guifi.net/node/14805

[6] Alexandra Villagrasa Batalla, Esunly Medina Medina, *A real-world implementation and parametrization of mobile ad–hoc networks*
http://docencia.ac.upc.edu/EPSC/XSS/memoria.pdf

[7] GràciaSenseFils, *Compilar BMX*
http://graciasensefils.net/doku.php?id=desenvolupament:bmx:compilar

[8] GràciaSenseFils, *Compilar kamikaze*
http://graciasensefils.net/doku.php?id=desenvolupament:kamikaze:compilar

APPENDIX A. USING OPENWRT

In order to perform the tests and configure our nodes properly, we need an own firmware image, based as we said on the OpenWRT Linux distribution. The latest version at the time of writing this paper is Kamikaze 8.09.1, released on Jun 01, 2009.

A.1. Compiling OpenWRT

To build a customized firmware image, we basically need to compile the sources, following the next steps.

The first step to be done is to download the current release (8.09.1) via subversion, and update the feeds (packages) from the repositories:

```
# svn co svn://svn.openwrt.org/openwrt/tags/8.09.1
# cd 8.09.1/
# ./scripts/feeds update
# make package/symlinks
```

Before we move on, we need to apply a patch for the signal LEDs of the Nanostation to work properly, due to a bug in the OpenWRT sources.

```
# patch -p0 -i gpio_leds.patch
```

At this point, we have the fully downloaded sources (and packages) of the latest OpenWRT release. The second step is to configure which packages we will use for our tests. We need to keep the image size low, because the NanoStations only has a Flash memory of 4 MB.

To select the needed packages, and unselect the ones which are not needed, the following lines apply:

```
# make menuconfig
Target System -> Atheros 231x/5312 [2.6]
Network -> batmand [*]
Network -> hostapd-mini [ ]
Network -> iperf [*]
Network -> olsrd [*]
Network -> ppp [ ]
```

After that, we compile the source code with:

```
# make world V=99
```

where V=99 will verbose all the output of what is doing.

That process takes a little bit, as it needs to download all the source code of the packets needed to compile (including the Linux kernel, and some libraries), and compile them.

The first time to compile it can last for approximately 2 hours, depending on the Internet connection and the processor capacity.

A.2. Uploading the firmware

After the process, the image for the Nanostation 2 has been created, and is located at the directory

```
bin/openwrt-atheros-ubnt2-squashfs.bin
```

With the original Ubiquiti firmware, AirOS, the firmware can be uploaded via the web administrative panel. At the System tab, Firmware section, just clicking at the ”Upgrade...” button and following the steps it can be easily uploaded.

If we already uploaded an OpenWRT image before, the way to upload the newly created image via tftp. For that, we need to reboot the device, and press the reset button while booting. By default, the NanoStation is configured to run at 192.168.1.20, and supports the image upload via tftp:

```
# cd bin/
# tftp 192.168.1.20
# binary
# put openwrt-atheros-ubnt2-squasfs.bin
```

Once the image is uploaded, the new firmware will boot and listen by default at the address 192.168.1.1

A.3. Configuring OpenWRT

Once the image has been uploaded, several configuration files need to be changed. Configuration in the OpenWRT environment is done via UCI. For example, to list the network configuration parameters we could enter the following:

```
# uci show network
network.loopback=interface
network.loopback.ifname=lo
network.loopback.proto=static
network.loopback.ipaddr=127.0.0.1
network.loopback.netmask=255.0.0.0
[...]
network.wlan0.ifname=ath0
network.wlan0.proto=static
network.wlan0.ipaddr=172.16.0.1
network.wlan0.netmask=255.255.255.0
```

So, changing an IP address is as easy as doing:

```
# uci set network.wlan0.ipaddr=172.16.0.2
# uci commit
# /etc/init.d/network restart
```

UCI will write the configuration parameters to the needed package, in this case `/etc/config/network`.

As all the NanoStation have very similar configuration files, a common strategy is to include these files in the firmware images. For that, a directory `files/` is created at the root of the OpenWRT source code, and all the files there will be merged with the base file system.

Listed below are some of the files included in the firmwares:

- `/bin/quality-LEDs.sh`: script to switch on or off the LEDs in the NanoStation depending on the quality of the signal the device received.
- `/bin/batctl`: script to output the B.A.T.M.A.N. Advanced variables, including originators and interfaces.
- `/etc/config/batmand-adv-kernelland`: configuration file for B.A.T.M.A.N. Advanced.
- `/etc/config/batmand`: configuration file for B.A.T.M.A.N.
- `/etc/config/bmx`: configuration file for BMX.
- `/etc/config/network`: configuration file of all the interfaces, either real or virtual, of the router.
- `/etc/config/olsrd`: configuration file for OLSR.
- `/etc/config/wireless`: configuration file with the wireless settings, such as the channel, protocol or SSID.
- `/etc/dropbear/authorized_keys`: authorized public keys to access to the NanoStation via SSH using public key authentication.
- `/etc/init.d/nano-leds`: init script to enable the Quality LEDs script.
- `/etc/passwd`: passwords file, to set an specific password to access the router via SSH.
- `/www/cgi-bin/cgi-bin-dev-zero.bin`: simple script which outputs /dev/zero and is used to determine simple TCP throughput using web access. Taken from GraciaSenseFils[1].

Some of these files are shown at chapter D on pag 67.

[1]`https://rilat.guifi.net/svn/mesh-gracia/0.2/files/www/cgi-bin/cgi-bin-dev-zero.bin`

APPENDIX B. USING B.A.T.M.A.N.

The generated firmware image will include the B.A.T.M.A.N. binaries, so there is no need to install it afterwards.

Running B.A.T.M.A.N. on the default wifi interface (ath0) is as easy as executing:

```
# batmand ath0
Using interface ath0 with address 172.16.0.1 and broadcast address 172.16.0.255
```

B.A.T.M.A.N. offers debugging info so that an observer is able to know what the protocol is doing. For that, execute:

```
# batmand -c -d 1
```

Where the flags

- -c: connects to the B.A.T.M.A.N. daemon via the unix socket.
- -d: verbose level of the output, from 0 to 5. The verbose levels are:
  - 0: disabled debugging, enabled by default.
  - 1: shows the neighbours list in the B.A.T.M.A.N. mesh network.
  - 2: shows the gateways to the Internet in the B.A.T.M.A.N. mesh network.
  - 3: observes B.A.T.M.A.N., showing more information about what the protocol is doing, ie, the routes it adds and deletes, new neighbors appearing and disappearing, etc.
  - 4: observes B.A.T.M.A.N. very verbosely, each action the protocol performs is shown in this mode, ie, each sent and received packet, what functions are called, etc.
  - 5: show CPU and memory usage.

Taking a look closer to the level of verbosity 1, the output is something like the following:

```
  Originator  (#/255)        Nexthop [outgoingIF]:   Potential nexthops ... [B.A.T.
172.16.0.4      (252)      172.16.0.4 [      ath0]:      172.16.0.4 (252)      172.1
172.16.0.3      (247)      172.16.0.3 [      ath0]:      172.16.0.3 (247)      172.1
172.16.0.2      (255)      172.16.0.2 [      ath0]:      172.16.0.2 (255)      172.1
172.16.0.5      (244)      172.16.0.5 [      ath0]:      172.16.0.5 (244)      172.1
```

As can be seen, the originator nodes are shown on the first column. The second column (the number in parenthesis) indicates the quality of the link, over 255.

The second column indicates the next hop node, the next node where our packets are headed in order to reach the node. In parenthesis the outgoing interface where the packets will be sent.

Finally, the following columns show the potential next-hops, that is, the nodes in the network which are also able to reach the node in the first column, but have a not so good quality link.

APPENDIX C. PERFORMING TESTS

Tests on the scenario described at section 3.4. on page 32 are performed using different tools which includes ping, iperf, and sar. Below there is a brief overview of how to use each of the tools.

C.1. ping

Ping is a popular network tool user for testing the network connectivity. This tool is already included in the OpenWRT firmware image by default, and is used to estimate the round-trip time.

The RTT can be estimated by:

```
# ping 172.16.0.1 -s 1016 -c 61
```

Where:

- -s: the size of the packet date, excluding the headers. By default is 56.
- -c: the number of *echo request* packets.

C.2. iperf

The iperf tool must be included in the firmware image, at the *make menuconfig* menu, by choosing:

```
# make menuconfig
Network -> iperf [*]
```

iperf is used to perform the UDP and TCP tests, including throughput, jitter and probabilty of error. Either with UDP or TCP there are two nodes where the test is performed, a server and a client.

C.2.1. UDP

The used commands are:

```
Server: iperf -s -u -i 5
Client: iperf -c 172.16.0.1 -u -b 1M -t 60 -i 5
```

Where:

- -s: run iperf in server mode.
- -c: run iperf in client mode, connecting to the IP address of the server.
- -u: use UDP.
- -b: offered bit rate (only valid when using UDP).
- -t: duration of the transmission in seconds.
- -i: seconds between periodic bandwidth reports.

These two commands will output the transfer rate, the jitter of the link, and the lost packets of the transmission (probability of error). The final output looks like:

```
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  0.0-60.0 sec  14.3 MBytes  2.00 Mbits/sec  1.006 ms    1/10187 (0.0098%)
```

C.2.2. TCP

The used commands are:

```
Server: iperf -s -i 5
Client: iperf -c 172.16.0.1 -t 60 -i 5
```

Where each flag is already explained at section C.2.1.. Notice that in TCP mode the client can't offer a fixed bit rate.

These two commands will output the transfer rate during the time of the transmission, and the transmitted data. The final output looks like:

```
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.1 sec  52.5 MBytes  7.33 Mbits/sec
```

C.3. Processor time

To measure the processor time each protocol, there are several methods. We chose the `time` tool for the OLSR. It is very easy to use, and the used command is:

```
# time -v olsrd -f /var/etc/olsrd.conf -nofork
```

The tool is runned for 10 minutes, and during that 10 minutes we reboot the neighbours of the mesh network, and move them from place to place, to cause changes in the network to happen.

Unfortunately, this tool can't be used with B.A.T.M.A.N. and BMX, because they daemonize to the background, and the time `tool` exits.

Instead, we will make use of the B.A.T.M.A.N. and BMX tools theirself, which provide ways to show the CPU usage.

For B.A.T.M.A.N. the following command:

```
# batmand -cd 5
```

will show information about CPU time in this protocol.

On the other hand, the following command:

```
# bmxd -lcd8
```

will directly show CPU usage for the BMX protocol.

Finally, the `top` utility can also measure the CPU usage time.

APPENDIX D. CONFIGURATION FILES

D.1. /etc/config/network

```
config 'interface' 'loopback'
        option 'ifname' 'lo'
        option 'proto' 'static'
        option 'ipaddr' '127.0.0.1'
        option 'netmask' '255.0.0.0'

config 'interface' 'lo_bmx'
        option 'ifname' 'lo:bmx'
        option 'proto' 'static'
        option 'ipaddr' '10.10.150.1'
        option 'netmask' '255.255.255.255'

config 'interface' 'lan0'
        option 'ifname' 'eth0'
        option 'proto' 'static'
        option 'ipaddr' '192.168.1.1'
        option 'netmask' '255.255.255.0'

config 'interface' 'wlan0'
        option 'ifname' 'ath0'
        option 'proto' 'static'
        option 'ipaddr' '172.16.0.1'
        option 'netmask' '255.255.255.0'
```

D.2. /etc/config/wireless

```
config wifi-device  wifi0
        option type     atheros
        option channel  1
        option diversity 0
        option antenna auto
        option sw_merge 1
        option country 724
        option outdoor 1

config wifi-iface
        option device   wifi0
        option network  wlan0
        option mode     adhoc
        option ssid     guifi-mesh-node
        option encryption none
        option protmode 0
        option bgscan   0
        option uapsd    0
        option rssi11a  9
        option rssi11b  9
        option rssi11g  9
        option bintval  1000
```

D.3. /etc/config/batmand

```
config batmand general
        option interface                "ath0"
        option announce
        option gateway_class
        option originator_interval
        option preferred_gateway
        option routing_class
        option visualisation_srv
        option policy_routing_script
```

D.4. /etc/config/bmx

```
config 'bmxd' 'general'
        option 'base_port' '16305'
        option 'prio_rules_offset' '400'
        option 'rt_table_offset' '40'
        option 'ogm_interval' '500'
        option 'announce_ifs' '1'
        option 'routing_class' '3'
        option 'one_way_tunnel' '4'

config 'plugin' 'plugin_0'
        option 'plugin' 'bmxd_config.so'

config 'dev' 'dev_0'
        option 'dev' 'lo:bmx'
        option 'ttl' '50'

config 'dev' 'dev_3'
        option 'dev' 'ath0'

config 'throw_rule' 'throw_rule_0'
```

```
        option 'throw_rule' '172.16.0.0/14
```

D.5. /etc/config/olsrd

```
config olsrd
        option IpVersion '4'

config Interface
        list interface 'wlan0'
```

APPENDIX E. ACRONYMS AND ABBREVIATIONS

AODV	Ad-hoc On Demand distance Vector
AS	Autonomus System
B.A.T.M.A.N.	Better Approach To Mobile Adhoc Networking
BGP	Border Gateway Protocol
BMX	B.A.T.M.A.N. Experimental
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DSR	Dynamic Source Routing
EGP	Exterior Gateway Protocol
EQ	Echo Quality
IP	Internet Protocol
IGRP	Interior Gateway Routing Protocol
IS-IS	Intermediate System to Intermediate System
IXP	Internet Exchange Point
LED	Light-Emitting Diode
MAC	Media Access Control
MANET	Mobile Adhoc Network
MPR	Multi-Point Relay
NAT	Network Address Translation
NIC	Network Interface Card
NGO	Non Government Organisation
OGM	Originator Message
OLSR	Optimized Link State Routing
OLSR-NG	Optimized Link State Routing - Next Generation
OSPF	Open Shortest Path First
RIP	Routing Information Protocol

RIPE	Reseaux IP Europeens
RQ	Receiving Link Quality
RTT	Round-trip Time
SSH	Secure Shell
SSID	Service Set Identifier
TCP	Transport Control Protocol
TQ	Transmit Link Quality
TTL	Time to live
UCI	Unified Configuration Interface
UDP	User Datagram Protocol
WMN	Wireless Mesh Network