



Escola d'Enginyeria de Telecomunicació i  
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# MASTER THESIS

**TITLE: Guifi.net - Characterization, data collection and self-management of community**

**MASTER DEGREE: Master in Science in Telecommunication Engineering & Management**

**AUTHOR: Mateu Seguí Dopazo**

**DIRECTOR: Roc Meseguer**

**DATE: February, 16<sup>th</sup> 2017**



**TITLE: Guifi.net - Characterization, data collection and self-management of community**

**MASTER DEGREE: Master in Science in Telecommunication Engineering & Management**

**AUTHOR: Mateu Seguí Dopazo**

**DIRECTOR: Roc Meseguer**

**DATE: February, 16<sup>th</sup> 2017**

## Resumen

En este proyecto, vamos a presentar una solución extremo a extremo para los principales problemas que normalmente impactan a las redes comunitarias y especialmente a Guifinet.

Para introducir nuestra solución, estuvimos investigando como funciona internamente Guifinet (su jerarquía de red, equipos usados, configuración IP, y también su sistema de financiación) y una red WISP (Wireless Internet Service Provider).

Una vez analizamos y detectamos todos los problemas potenciales, realizamos una simulación para testear dos protocolos experimentales (OLSR y BATMAN) con el fin de probar su rendimiento y calidad de servicio y así elegir el que sería mejor para nuestra solución.

Y finalmente, vamos a presentar nuestro nuevo diseño basado en una red MPLS sobre OLSR.

**TITLE: Guifi.net - Characterization, data collection and self-management of community**

**MASTER DEGREE: Master in Science in Telecommunication Engineering & Management**

**AUTHOR: Mateu Seguí Dopazo**

**DIRECTOR: Roc Meseguer**

**DATE: February, 16<sup>th</sup> 2017**

## Overview

In this project, we are going to present an E2E (end to end) solution for the principal problems that normally impact the community networks and especially Guifinet.

To introduce our solution, we were investigating how the Guifinet works internally (its network hierarchy, equipment used, IP configuration and also its financial system) and also how wireless technology works and their limitations.

Once we analysed and detected all the potential issues, we performed a routing performance and QoS (quality of service) simulation in order to test two experimental protocols called BATMAN and OLSR to find the most suitable routing protocol for our approach.

And finally, we presented our new Guifinet network concept basing in MPLS over OLSR.

# TABLE OF CONTENTS

MASTER THESIS .....	1
TABLE OF CONTENTS .....	5
CHAPTER 0. COMMUNITY NETWORKS, INTRODUCTION TO THE PROBLEM .....	8
0.1. Introduction.....	8
0.2. The reason of this project.....	8
0.3. Methodology of this memory .....	8
0.4. What is a Community network?.....	9
0.5. What is a WISP (Wireless Internet Service Provide)? .....	10
0.6. Chapter conclusions .....	11
CHAPTER 1. ANALYSING GUIFI.NET .....	12
1.0. Introduction.....	12
1.1. Guifi.net definition.....	12
1.2. Guifinet network structure .....	12
1.3. Devices and manufactures considerations .....	15
1.4. Financial aspects .....	15
1.5. Chapter conclusions .....	16
CHAPTER 2. WIRELESS AND ROUTING CONSIDERATIONS.....	17
2.1. Introduction.....	17
2.2. Wireless physical considerations .....	17
2.3. IP routing protocol considerations .....	21
2.4. Wireless mesh network meaning clarification .....	21
2.5. Experimental wireless routing protocols .....	21
2.6. Chapter conclusions .....	22
CHAPTER 3 – OLSR and BATMAN .....	23
3.1. Introduction.....	23
3.2. Setup.....	23
3.3. General protocol overview .....	24
3.4. Performance test cases .....	24
3.5. Chapter conclusions .....	27
CHAPTER 4 GUIFINET – NEW PLAN – MPLS network over OLSR .....	28
4.1. Introduction.....	28
4.2. Guifinet network structure .....	28
4.3. Current Guifinet issues .....	30
4.4. Introducing MPLS (Multi-Protocol Label Switching) networks.....	32
4.5. MPLS real case .....	35
4.6. New configuration and deployment plan .....	37
4.7. Recommended equipment .....	39
4.8. Environment impact.....	40
4.9. Conclusions.....	40
ANNEXES.....	41

<b>Network schema .....</b>	<b>41</b>
Linux VMware BATMAN and OSLR simulation .....	44
Introduction .....	44
Scenario .....	44
Creating the network schema .....	45
<b>Configuring the network.....</b>	<b>46</b>
<b>OLSR simulation .....</b>	<b>49</b>
Installation .....	49
Troubleshooting .....	53
Batman-adv simulation.....	55
Tweaking and protocol behaviour .....	60
<b>Netdata routers data .....</b>	<b>62</b>
<b>Traffic idle consumption .....</b>	<b>62</b>
OLSR .....	62
BATMAN .....	62
PER (packet error rate) first path 0% and second path 0%.....	63
BATMAN .....	64
PER (packet error rate) first path (node 2) 4% and second path (node 4) 0%:.....	66
BATMAN .....	67
PER (packet error rate) first path (node 2) 20% and second path (node 4) 0% .....	69
BATMAN .....	71
<b>Measuring the protocol behavior with RTT .....</b>	<b>72</b>
RTT (round trip time) first path (node 2) 20ms and second path (node 4) 0.5ms: .....	72
RTT (round trip time) first path (node 2) 40ms and second path (node 4) 0.5ms .....	76
RTT (round trip time) first path (node 2) 60ms and second path (node 4) 0.5ms .....	79
<b>Jitter measure .....</b>	<b>81</b>
RTT 10ms – Node 2 Jitter 3.5ms – Node 4 jitter 0.5ms.....	81
BATMAN .....	84
RTT 10ms – Node 2 Jitter 6ms – Node 4 jitter 0.5ms.....	86
BATMAN .....	88
<b>GNS3 – MPLS – OSPF – multi BGP simulation .....</b>	<b>90</b>
<b>BIBLIOGRAFY.....</b>	<b>100</b>



# CHAPTER 0. COMMUNITY NETWORKS, INTRODUCTION TO THE PROBLEM

## 0.1. Introduction

In this chapter, we are going to talk about the below topics.

- The reason of this project.
- Community networks.
- WISP (Wireless Internet Service Provider) networks.

Also, we are going to explain the methodology of this project and the steps that we have done, in order to arrive to the final network design.

## 0.2. The reason of this project

Wireless network is a new technology that was not designed as backbone (high capacity bandwidth) technology because for these purposes, we were already having technologies such as optic fiber or copper wire.

However, when the wireless technology arrived, this offered the opportunity to deploy WAN or MAN networks without having to spend a lot of money (fiber network are very expensive).

So, this was basically the reason, how the community network was born.

However, this technology (Wi-Fi) was not mature and didn't have any routing protocol designed yet. So, currently, all the community networks are deploying their backbones using conventional routing protocol designed for fiber and copper.

This causes that the performance and the reliability are very poor.

The last years, different research departments are investigating to develop new routing protocols that can adapt to the intrinsic nature of this technology. But unfortunately all these protocols are still under development, and the router main vendors (Cisco, Huawei, Alcatel, Juniper) have not already integrated into their professional equipment.

Therefore, this projects starts at this point (trying to arrive to the last mile).

## 0.3. Methodology of this memory

In this memory we have done the below task.

1. Study the real case Guifinet and extract the main issues.
2. Study the wireless routing and environment considerations.



3. Investigate the available wireless routing protocols and select the two most important ones.
4. Simulate these two protocols (OLSR and BATMAN) in a ring topology trying to reproduce a real Guifinet backbone environment.
5. Once, we know what it is the best protocol, we have implemented a MPLS network over this routing protocol that will provide better performance than a pure IP network.

#### **0.4. What is a Community network?**

As Wikipedia definition says, this is a term used broadly to indicate the use of networking technologies by, and for, a local community.

Free-nets and civic networks indicate roughly the same range of projects and services, whereas community technology centers (CTCs) and tele-centers generally indicate a physical facility to compensate for lack of access to information and communication technologies (ICTs):

In our definition, Community network is a tool to spread the world knowledge and free Internet around the world in order to create a social network that everybody can use and collaborate.

Normally, the Community networks are basically a WISP (Wireless Internet Service Provider) built with cheap devices providing low performance and reliability using the common wireless standard IEEE 802.11.

Normally, all the nodes are fixed radio-stations connected between themselves forming a mesh network with enough redundant paths in order to provide reliability (if one node fails, the network have additional paths to achieve the same destination).

So, in order to manage (in as efficient way), all these redundant paths, we need to use intelligent and fast routing protocols.

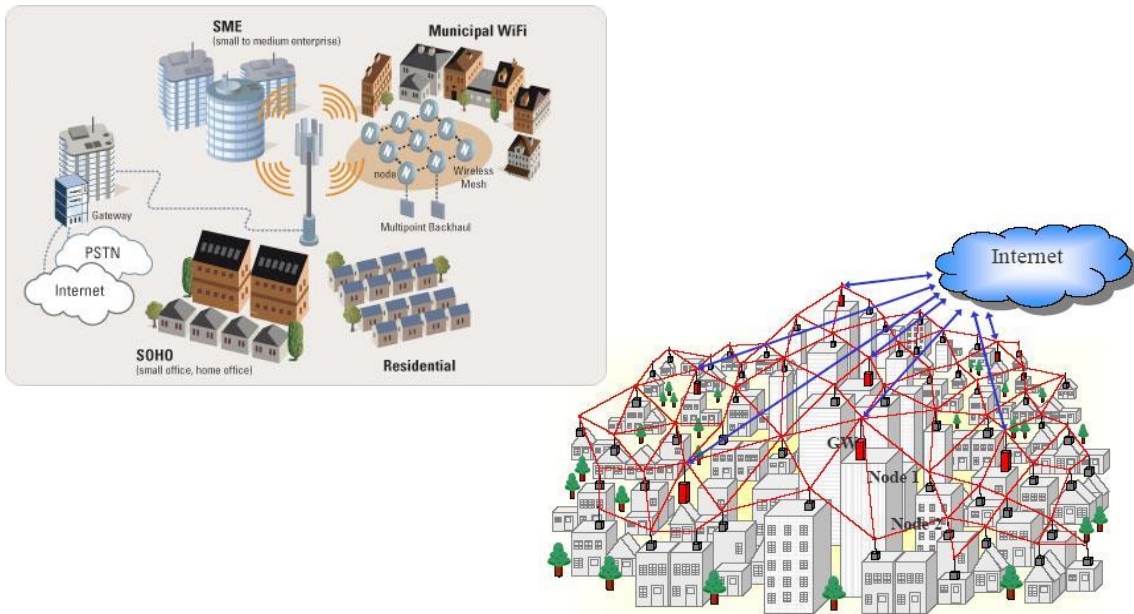


Figure 1 - WISP network schema

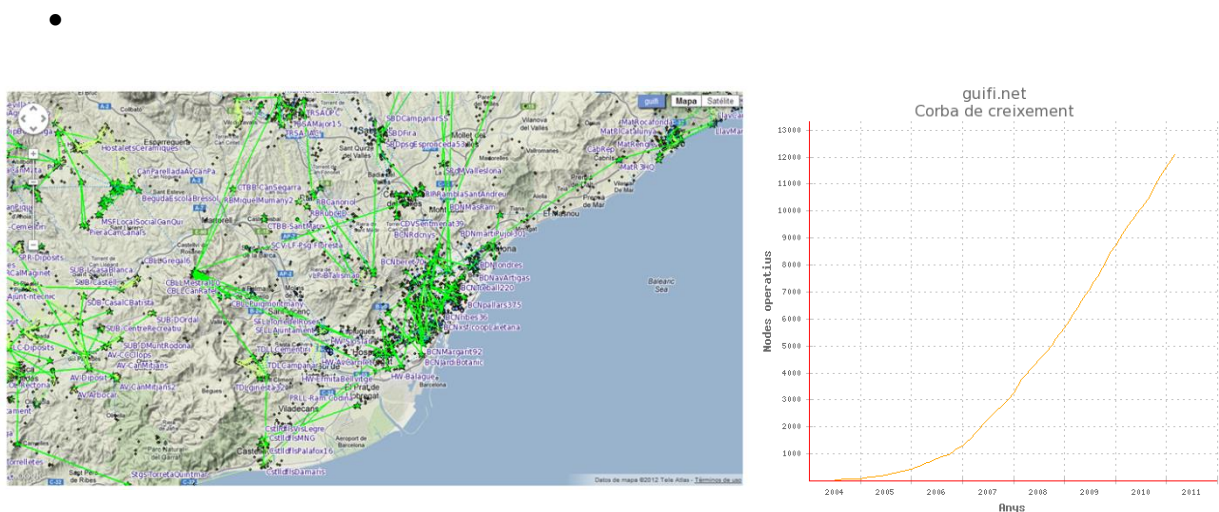


Figure 2 - Guifinet Catalonia map

## 0.5. What is a WISP (Wireless Internet Service Provide)?

WISP is a MAN or WAN network that uses Wi-Fi as backbone technology.

Normally these networks use long distance radio links forming a mesh or ring topology to provide redundant paths in order to improve its network reliability.

The most complexes WISP work with MPLS and multi-protocol BGP services.

## 0.6. Chapter conclusions

In this chapter, we have learnt the below points.

- The reason of this project.
- Community network is a volunteer network that handle very little budget that uses only wireless equipment.
- Wireless networks were not designed to carry high capacity traffic in backbone schema (mesh network with multi redundant paths to achieve the same endpoint).
- The current router vendors are not giving support to wireless routing protocols.
- WISP is the standard that community networks should follow.

# CHAPTER 1. ANALYSING GUIFI.NET

## 1.0. Introduction

After introducing the community network, WISP concept and the reason of this project, now in this chapter, we are going to talk about the real case Guifinet.

In this chapter, we are going to introduce a small summary about its internal structure.

## 1.1. Guifi.net definition

Guifi.net is a free community network that everybody can participate.

It was born in 2004, focused to provide a broadband connection (public and free) to “last mile” (network access) where normally the normal ISPs (Internet Service Providers) never arrive.

Currently it has around 26600 nodes and 17000 are permanently active.

This network is especially based in Spain (mainly in Catalonia and Valencia) and also, has other countries such as Argentina, Nicaragua, Bolivia, and Portugal.

The main financial medium is the donation - Guifinet's projects are published in its own webpage, and anyone can make a money contribution.

In 2011, Guifi.net was connected to CATNIX and since then, Guifinet concept changed, becoming now a WISP network with IANA AS (autonomous system) code.

## 1.2. Guifinet network structure

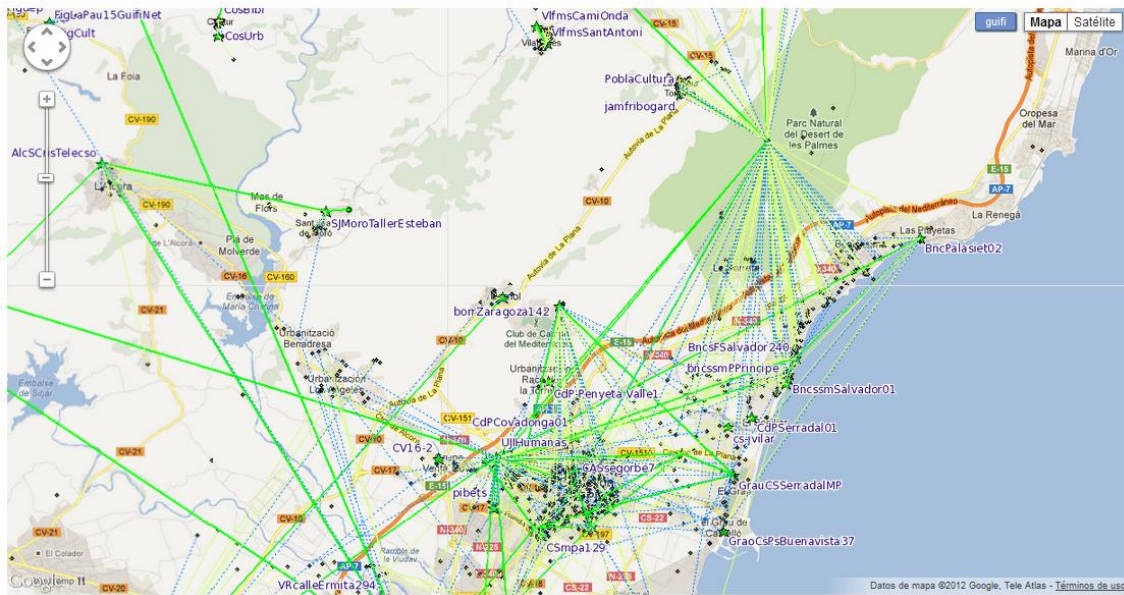
The network is a pure WISP network with two node categories:

### 1.2.1. Primary nodes (backbone nodes):

Also called super-nodes, these are the critical elements that support the whole backbone network stability.

### 1.2.2. Secondary node (access nodes):

Usually, they are built using low cost devices without redundancy and obviously the quality and reliability are not guaranteed.



**Figure 3 - Barcelona Guifinet backbone network**

### 1.2.3. Guifinet IP addressing

It uses only private IP ranges and depending of the type of nodes or services, these are split into these three IP sets.

- Public layer (class A - 10.0.0.0 - 10.255.255.255):

These sets are reserved for servers and end customer applications.

- Manage layer (class B - 172.16.0.0 – 172.31.255.255):

These sets work in the backbone area and are responsible to transport the traffic between different access networks.

- Private layer (class C - 192.168.0.0 - 192.168.255.255):

This range work in the end customer network (normally they work behind a NAT router).

## 1.2.4. IP management design

Guifinet works using IP routing (layer 3). So, this means that, each node works as a IP router and every interface a different IP sub-net.

The Guifinet IP addressing uses geographic approaches (routing table efficiency – “supernetting”), and in Catalonia, we have around sixty networks.

Catalonia topology example

**Table 1 - Catalonia main IP networks**

Network	host	type	IPs used	Used %
10.139.0.0/16	65,534	public	5616	9%
172.25.0.0/16	65,534	backbone	3287	5%
10.140.0.0/16	65,534	public	1321	2%

**Table 2 - Catalonia branch IP networks (information uncompleted)**

Network	initial / final	hosts	type	min / max	IPs used	used %
<b>Alt Penedès</b>						
<b>10.138.46.0/23 (255.255.254.0)</b>	10.138.46.1/ 10.138.47.254	510	public	10.138.46.1 10.138.47.97	/9	2%
<b>Anoia</b>						
<b>10.145.0.0/16 (255.255.0.0)</b>	10.145.0.1/ 10.145.255.254	65,534	public	10.145.0.1 10.145.60.129	/675	1%
<b>172.17.16.0/20 (255.255.240.0)</b>	172.17.16.1/ 172.17.31.254	4,094	backbone	172.17.16.1 172.17.21.2	/299	7%
<b>Bages</b>						
<b>172.25.128.0/22 (255.255.252.0)</b>	172.25.128.1/ 172.25.131.254	1,022	backbone	172.25.128.1 172.25.129.122	/162	16%
<b>10.228.0.0/17 (255.255.128.0)</b>	10.228.0.1/ 10.228.127.254	32,766	public	10.228.0.1 10.228.32.1	/808	2%

<b>Baix Empordà</b>						
<b>10.155.0.0/20</b> <b>(255.255.240.0)</b>	10.155.0.1/ 10.155.15.254	4,094	public	10.155.0.1 10.155.15.230	/416	10%
<b>Baix Llobregat</b>						
<b>10.138.9.0/24</b> <b>(255.255.255.0)</b>	10.138.9.1/ 10.138.9.254	254	public	0.0.0.0 / 0.0.0.0	0	0%
...						

The table shows “partially” this IP network hierarchy which depends of the initial network showed in the previous table. For every geographic district, it has at least one network that may be a public or backbone network.

### 1.3. Devices and manufactures considerations

High performance telecommunication equipment is expensive. Whereby the main Guifinet nodes were designed using cheap wireless devices that obviously have performance limitations.

However, the last years, new providers such as Ubiquiti or Mikrotik have appeared on the market, revolutionizing the Wi-Fi technology.

Thanks to them, now it’s possible to build wireless nodes with a cost-effective much cheaper than the normal manufactures such as Cisco or Alcatel.

In fact, Ubiquiti is offering a cost-effective radio link with a similar performance than a fiber link called (Ubiquiti - Air Fiber product).

And also, Mikrotik is offering high performance cost-effective modular routers with advance routing and switching protocols such as BGP, OSPF, MPLS, Layer 2 aggregation.

So, Guifinet is migrating their old equipment to the vendors.

### 1.4. Financial aspects

Guifinet is financed by donation and voluntarism.

In Guifinet webpage, anyone can publish his node projects and post the amount of money which is necessary to build it and people can donate.

For instance, if someone is living in a low population density region where the national ISPs (Internet Service Providers) cannot or don’t want to provide Internet connection, this person and his own community would be interested to deploy their own node.

And, if there is no backbone access, they would be also interested to extend the backbone scope.

So, the way to get funds and coordination would be using the Guifinet crowdfunding portal.



**Figure 4 - Guifinet webpage - Financial project example**

## 1.5. Chapter conclusions

In this chapter, we have learnt the below points.

1. Guifinet is a community networks with a mesh configuration.
2. They use private IP addressing.
3. Their financial system is the donation.
4. Due to their budget limitations, and lack of coordination, this network is not offering an end to end user experience.



# CHAPTER 2. WIRELESS AND ROUTING CONSIDERATIONS

## 2.1. Introduction

As we saw in the previous chapters, Guifinet is basically a wireless network. So, in order to understand its limitations, we present this chapter to provide a summary about this technology.

Here, we are going to explain the important physical, access and routing considerations that we have to take into account to deploy a wireless backbone network (WISP).

And finally, we are going to explain the routing considerations in Wi-Fi.

## 2.2. Wireless physical considerations

As Community networks are wireless networks, it's important to introduce important physics concepts in order to understand their intrinsic limitations.

### 2.2.1 Wireless medium behavior

Wireless medium is a no guide medium, therefore, it is not possible to control "how clean the transmitter channel is". So, we are in front of an unpredictable medium that depends of many random variables that unfortunately, are impossible to control.

Here we have some example of variables that we should bear in mind:

- Variability of refraction index (Snell's law) produced by humidity, temperature, sun radiation, refraction, kind of land (forest, mountains, lakes, sea...)
- Multi-propagation is another important and well-known value that degrades the wireless communication, especially in high speeds.
- Fresnel radius is an important value to keep in mind, especially if there is not a LOS (line of sight) connection.

### 2.2.2. Wireless layer link considerations

In this section, we explain how the IEEE 802.11 (layer access) works internally (layer 2).

Wireless network can work (OSI layer 2) in three modes:

1. Infrastructure
2. WDS
3. Ad-hoc

### 2.2.3. Infrastructure mode

Centralized scheme, where one master device manages and controls the access medium of the rest of devices (slaves). In this schema, the master device is the responsible to control the well-behaviour of the whole Wi-Fi network.

This means, that the master device sends periodically beacon frames in order to announce its presence and its RF configuration. Thanks to this, the rest of devices can synchronize their transmission time applying the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) access protocol and its RF parameters such as modulations, encryption feature, etc...

In this case, the backbone network is always a wired network, where the master device (access point) works as bridge between wireless and wire medium (this is the classic configuration in Wi-Fi end customer access networks).

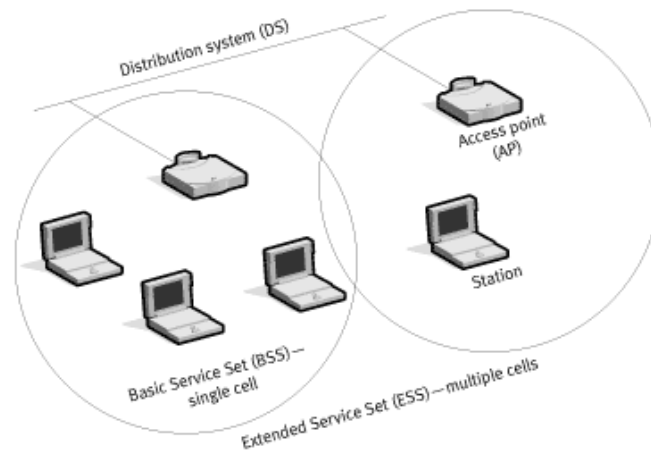


Figure 5 - Wi-Fi infrastructure mode

## 2.2.4. Wireless distribution system (WDS) mode

In this case, N master devices use the same wireless medium as backbone network and instead to send the traffic to the wire network, retransmit via Wi-Fi too, to other master devices.

Obviously, this configuration provides a lower performance comparing with the previous configuration (infrastructure mode). This is due to that the two transportation layers (access and backbone network) are sharing the same medium splitting the speed.

However, WDS is a great solution in environments that are not able to deploy a wired backbone network and the performance is not an important value to take into account.

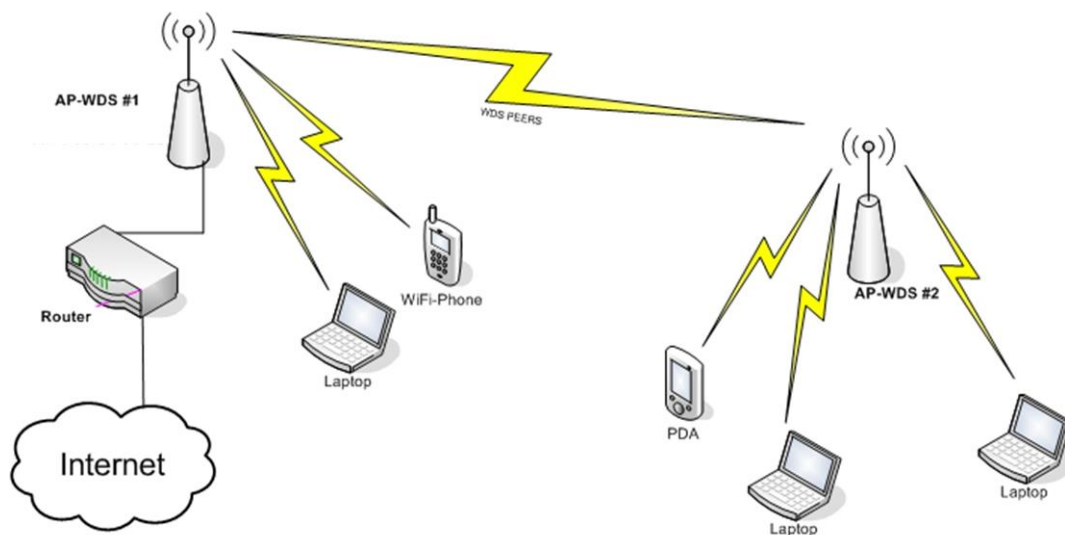


Figure 6 - Wi-Fi - WDS mode

## 2.2.5. Ad-hoc mode

In Ad-hoc networks, all devices transmit via broadcasting without hierarchy participating in the frame forwarding via flooding mechanism (pure ad-hoc without mesh routing protocols).

Obviously, this mechanism is very inefficient, because it causes high level of redundant information.

However, for specific solutions where the nodes are constantly in movement and the final throughput is not an important topic, are suitable.

These solutions could be a system to share information between cars, firefighters, policemen...

Now, IEEE is trying to standardize the mesh protocol with the draft 802.11s.

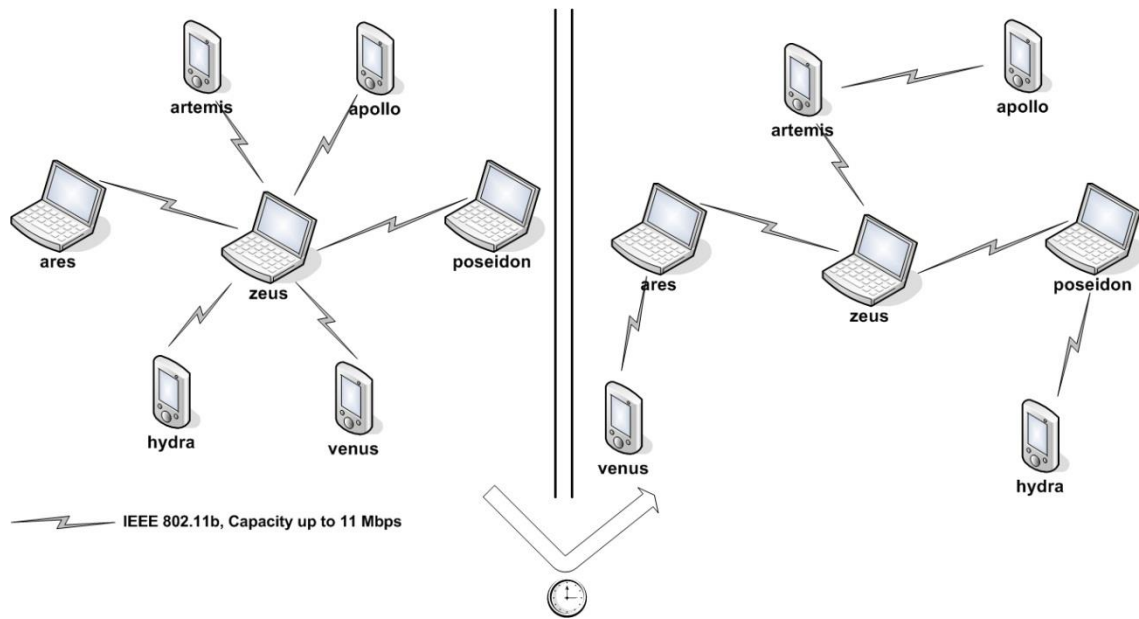


Figure 7 - Wi-Fi Ad-Hoc schema

## **2.3. IP routing protocol considerations**

Nowadays, the routing protocols (OSPF (open shortest path first), RIP (routing information protocol) and BGP (border gateway protocol)) are massively used in IP networks. These protocols work on third OSI layer and provide different capabilities that work quite well on wired network.

However, these routing protocols were not designed for handling wireless network, because in their internal logic, are not taking into account the packet errors, jitter and RTT rate that, are much higher and more unpredictable than on wired networks. In fact, these protocols only have two statuses (UP or DOWN).

However, in wireless network, we have a third status (DEGRADED) and it's here where these protocols fail.

Therefore, in order to improve the network quality and scalability, the new community networks should incorporate a new IP routing protocol that should be suitable for wireless considerations.

## **2.4. Wireless mesh network meaning clarification**

Before explaining wireless IP routing protocol in detail, there is a popular misunderstanding concerning mesh networks that would be required to clarify before continuing with this memory.

Normally, when people speak about mesh networks, they think about ad-hoc networks with no-fixed nodes like we described before.

However, although this is a valid definition, there is another definition, that for our approach is much more interesting.

WISP is a mesh network, however their nodes are not acting in ad-hoc mode, but infrastructure mode.

In fact, regarding our approach (due to that our nodes are completely fixed), the ad-hoc configuration would be absolutely useless.

## **2.5. Experimental wireless routing protocols**

As we could see before, the actual IP routing protocols are not suitable in WISP networks.

So, it's important to find a protocol that can take into account this third status (degraded) and can also react rapidly against a topology change.

Currently the list of protocols available can be classified into two categories:

### **2.5.1. On demand or reactive protocols:**

The routing tables are auto generated once one node wants to send traffic and these are also deleted once the transmission has been already completed. This kind of is suitable for low consumption nodes that are fed by batteries and the time response and throughput are not critical.

As examples we have these:

AODV - Ad-hoc On-demand Distance Vector  
DSR - Dynamic Source Routing  
TORA - Temporary Ordered Routing Algorithm

Regarding our project, these protocols are obviously not suitable.

### **2.5.2. Table driven or proactive protocols:**

Nodes are constantly sending signalling traffic to keep the routing tables synched consuming more energy and bandwidth than the reactive protocols, and providing better performance against topology changes and suddenly radio link degradations.

As examples, we have these:

BATMAN (better approach to mobile ad-hoc networking)  
OLSR (Optimized Link State Routing)

## **2.6. Chapter conclusions**

In this chapter we have learnt the below topics:

1. Wireless technology is very sensitive to the atmosphere changes and external interferences.
2. Wi-Fi has different operation modes.
3. The current IP routing protocol are not taking into account the third state degraded.
4. We have to two kind of mesh protocol (reactive and proactive).
5. Basing in our requirements, we need to use a proactive mesh protocol.

# CHAPTER 3 – OLSR and BATMAN

## 3.1. Introduction

As we explained in the previous chapter, OLSR and BATMAN are two IP routing protocols that were designed to work in wireless mesh network. Due to the wireless condition, these two protocols have mechanisms to monitor the state link in order to detect spontaneous error and performance degradations.

In this chapter, we are going to test how sensitive and accurate these two protocols are when one link has packet loss, high jitter and delay.

## 3.2. Setup

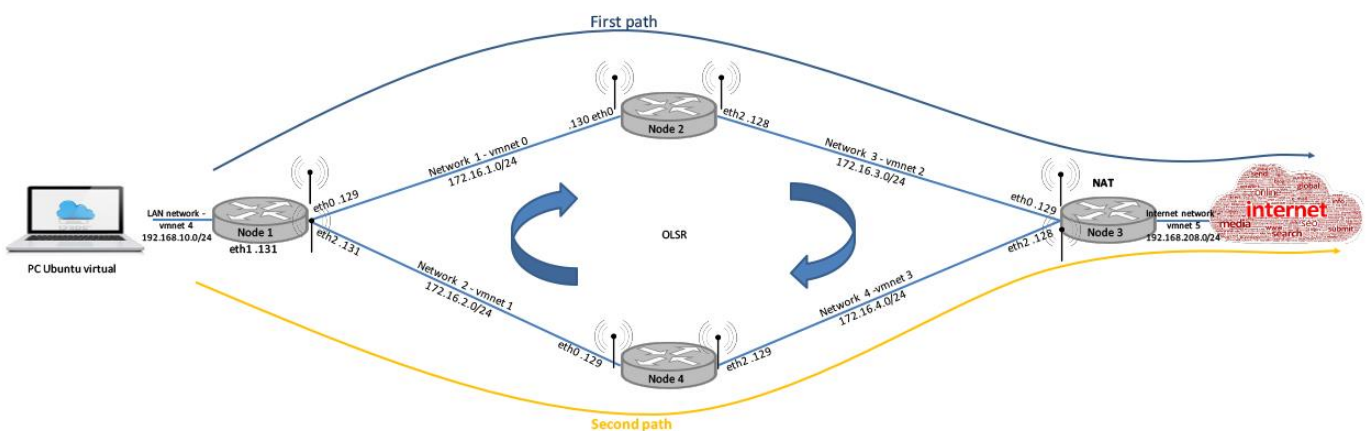


Figure 8 - Simulation network schema

In order to get these results, we have created a virtual wireless ring with two identical paths to achieve the same destination (for further information how to install, go to annexes).

In the picture, we have the whole setup that is composed for the below elements:

Every node works as an IP router that have Wi-Fi interfaces forming four point-to-point simulating the Wi-Fi schema point to point (one device master and one slave) infrastructure mode) wireless networks.

In order to simulate the wireless degradation, we used the Linux command tc.

4 routers (Linux machines) connected themselves via IP subnetting.

- Node 1 has the PC client connected.
- Node 2 forms the first path.
- Node 3 has Internet with an NAT interface.
- Node 4 forms the second path.

### 3.3. General protocol overview

BATMAN wins all the points to OLSR (see the table). This protocol has a better Linux implementation, the topology convergence time is faster and the traffic consumption is lower.

So, in environments with low consumption (small Wi-Fi mobile devices), BATMAN is more suitable than OLSR.

**Table 3 - OLSR vs BATMAN comparison**

	<b>OLSR</b>	<b>BATMAN</b>
<b>Protocol OSI layer</b>	3 (network layer)	2 (link layer)
<b>Linux integration</b>	init child process (daemon)	Kernel module
<b>Topology convergence time (one path is down)</b>	25s	1s - 10s
<b>Routing standby data traffic</b>	symmetric 3,5kbps	symmetric 1,88kbps

### 3.4. Performance test cases

In order to perform the tests, we have done the below tasks sequentially (one time per test):

- Initially, second path is always down (so, all the traffic must go to the first path).
- Node 3 has an Iperf server session listening.
- Client PC uses Iperf to connect to Node 3 to perform a TCP upstream connection.
- We add to Node2 interface some PER, jitter and RTT using the Linux command TC (depending on the test case tested).
- Client PC starts to send traffic to Node 3 via path 1 (Node 2).
- Monitor the Node 2 traffic with Netdata graphs checking the traffic degradation due to the PER, jitter and delay (RTT)
- We turn the second path on and wait for the reaction (this second path is without artificial channel degradation).
- Monitor the Node 4 traffic via Netdata to check if the protocol (BATMAN or OLSR) switches to this path.
- Collect the results and print in the test cases grid.



So, the tests have been divided in the below areas:

1. PER (packet error rate)
2. Delay or RTT (round trip time)
3. Jitter (delay variability)

The expected results would be that the tested protocol should use the path with better PER, jitter and RTT. So, our expectation is that both protocols should switch to the second path as soon as possible in order to avoid the degradation.

When the protocol passed the test, we put in green. Otherwise red.

### 3.4.1. PER (packet error rate)

BATMAN and OLSR worked very well. Both protocol moved all the traffic to the second path after 42 – 57s.

However, BATMAN tries to balance the traffic to all the available paths (when these are not degraded). For our point of view (for WISP networks) this is not a good practise for the below reasons:

This causes jitter impacting the TCP throughput and the real time application (VoIP or IPTV obligating to add buffer in the receivers).  
 IP packets balancing causes high computational cost limiting the maximum packet per second that a router can handle (fast switching mechanisms).  
 So, for our approach this is not advantage.

**Table 4 - BATMAN vs OLSR - packet error rate comparison**

Packet error rate (first and second path)	OLSR	BATMAN
0%/0%	The traffic goes to one path	The traffic is rounded (balanced) to both paths
4% / 0%	The protocol switches after 57s to the second path, and keeps there.	The protocol switches after 55s to the second path, and keeps there.
20% / 0%	The protocol switches after 42s to the second path, and keeps there.	The protocol switches after 46s to the second path, and keeps there.

### 3.4.2. Delay

In this case, we need to verify if these protocols are sensitive to the delay.

In these tests, we saw that BATMAN is not sensitive to the delay. This protocol does not take into account this parameter.

However, this parameter is very important for high TCP throughputs and real-time applications such as VoIP or IPTV.

**Table 5 - BATMAN vs OLSR - delay comparison**

RTT (first and second path)	OLSR	BATMAN
20ms / 0.5ms	The protocol switches after 45s to the second path, and keeps there.	It does not switch to 2n path (it keeps in the high RTT path)
40ms / 0.5ms	The protocol switches after 44s to the second path, and keeps there.	It does not switch to 2n path (it keeps in the high RTT path) and flaps
60ms / 0.5ms	The protocol switches after 42s to the second path, and keeps there.	It does not switch to 2n path (it keeps in the high RTT path) and flaps

### 3.4.3. Jitter

Jitter is another important value to bear in mind when you want to carry high throughputs and real time services.

Unfortunately, the experience was the same than the delay. BATMAN does not take into account the parameter.

**Table 6 - BATMAN vs OLSR - jitter comparison**

Jitter (first and second path) - RTT 10ms	OLSR	BATMAN
3.5ms / 0.5ms	The protocol switches after 40s to the second path, and keeps there.	It does not switch to 2n path (it keeps in the high RTT path) and flaps

10ms /0.5ms	The protocol switches after 42s to the second path, and keeps there.	Path 1 - Strong degradation and flaps to the second one. I don't know the reason.
-------------	--	---

### 3.5. Chapter conclusions

Here, we present our final conclusions.

Principal ideas:

1. OLSR takes always the best path (PER, jitter and RTT).
2. OLSR never balance the traffic using both paths.
3. BATMAN is not sensitive to RTT and jitter.
4. BATMAN tries to balance the traffic to both channels.
5. BATMAN convergence time is 10s and OLSR convergence time is 25s.
6. BATMAN signaling traffic is 1.8Kbps, and OLSR 3.5Kbps.

So, we have arrived the below conclusions:

1. For pure ad-hoc networks with mobile nodes that need fast convergence and low throughputs BATMAN is more suitable.
2. However, for WISP purposes, OLSR wins.

# CHAPTER 4 GUIFINET – NEW PLAN – MPLS network over OLSR

## 4.1. Introduction

Once we have studied the Guifinet issues, the wireless routing and atmosphere considerations and we simulated the two mesh routing protocols, now we are capable to offer a new design.

1. In this chapter, we are going to present the below topics.
2. Introducing the current Guifinet network structure with its main issues.
3. Introducing the new MPLS over OLSR network.
4. The final project conclusions.

## 4.2. Guifinet network structure

Guifinet works with a WISP network schema using cost-effective equipment with low performance and features that work in the standard IEEE 802.11.

### 4.2.1. Hierarchy

All nodes are fixed devices installed in towers and building roofs and they work as IP routers.

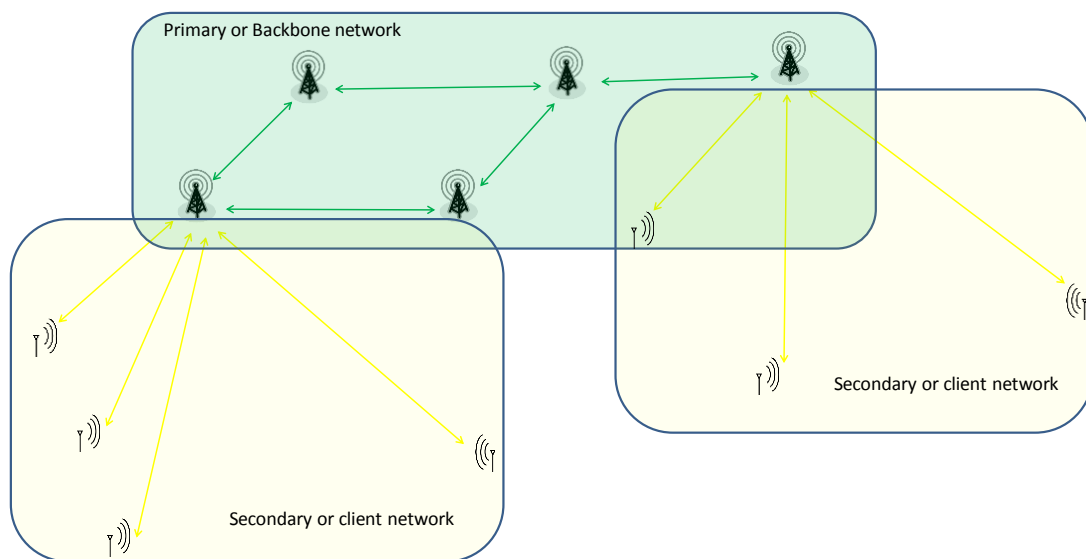


Figure 9 - Current Guifinet network

Depending the duty and topology, every node can be classified in one of these two hierarchies:

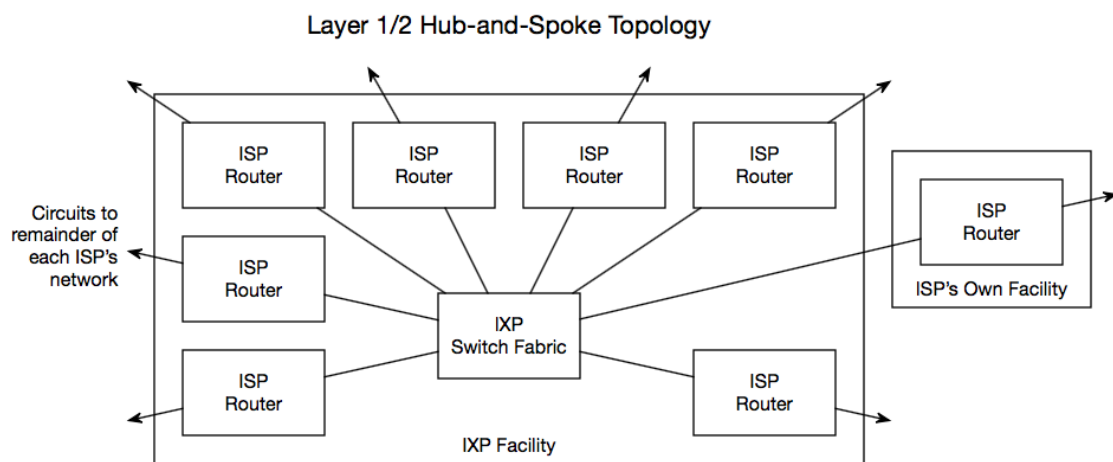
- Primary hierarchy (backbone network) work forming a mesh or ring topology using the routing protocols OSPF and OLSR. So, the whole network stability depends exclusively to this hierarchy.

- Secondary hierarchy (access networks) does not have redundant paths and the connection with the super-nodes (first hierarchy) is normally a point-to-point.
- These nodes build the access network forming a star topology where the end customer connect with their own equipment - CPEs (customer provider equipment)
- The CPEs are small Wi-Fi router with NAT capabilities that receives a WAN private IP.
- In this hierarchy (because there are no redundant paths) is not necessary to run complex IP routing protocols (normally the routing is provided by static routes or simple protocols such as RIP).

#### 4.2.2. Interconnection

To have Internet connection, Guifinet connects to the Catalonia NIX (Internet exchange point). So, the network has its IANA autonomous system number like any normal ISP (Internet Service Provider).

This means that now the network has its own public IP addresses (provided by IANA, instead of a ISP reseller), and also it now has to share its public IPs dealing with the rest of autonomous systems (that are connected to the same NIX) using the routing IP protocol eBGP.



**Figure 10 - Layer 2 ISP interconnection**

### 4.2.3. Public IP addressing

As the IPv4 addressing are more and more harder to get (there are not more available IP ranges), Guifinet works with NAT routing. So, this means that no all the end-customers have its own public IPs, because these are reserved for important tasks such as servers or NAT routers (a group of customers are sharing the same public IP).

Regarding IPv6, currently Guifinet is not supporting.

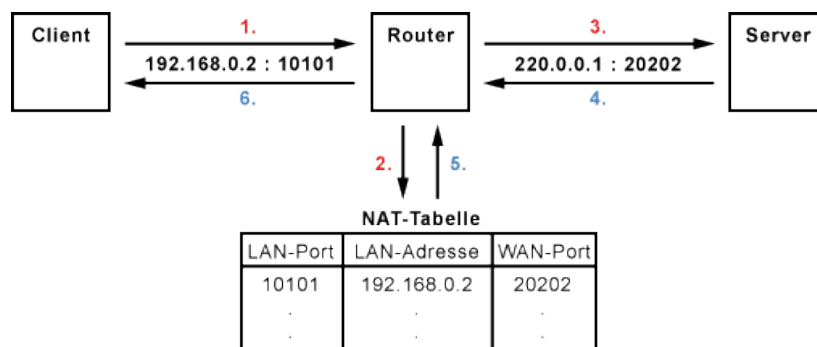


Figure 11 - NAT (Network Address Translation) routing technology

### 4.3. Current Guifinet issues

Here, we have a list with the main issues:

- Backbone low capacity – as the equipment is cheap, the real throughput is very low with channel degradation (PER, jitter and high RTT).
- No reliable backbone network – The number of redundant paths are low and they do not have mechanisms to detect degradations.
- Lack of central network design – this is due to that community networks work with volunteers and low budget, and there is not a responsibility hierarchy well defined. This causes inefficiencies that are quite expensive to fix.
- No monitoring or management network – Guifinet does not have a management network in parallel to monitor the nodes. This management network should work with external connection (for example, external low xDSL connection).
- No wireless IP routing protocols – This is the main issue and the reason of this project. The routers work without wireless IP routing protocols

causing that if one path has a degradation, the IP routing protocol does not switch to another path.

- No IPv6 – Currently, the IPv4 is disappearing and the ISP are fixing this issue migrating to IPv6 and installing massive ISP NATs called CGNATs.
- No end to end QoS (Quality of Service) – Guifinet is a best effort network without different traffic priorities with different paths (PBRs) Policy base routing or MPLS (Multi-Protocol Label Switching).

## 4.4. Introducing MPLS (Multi-Protocol Label Switching) networks

In this section, I am going to introduce this ISP technology that is not well-known using my MPLS simulation performed with GNS3 and the Cisco IOS c7200.

MPLS is a technology that substitute the IP forwarding for labels. This means that when a packet gets into a MPLS network, instead to be forwarded reading its IP header, this is forwarded via labels.

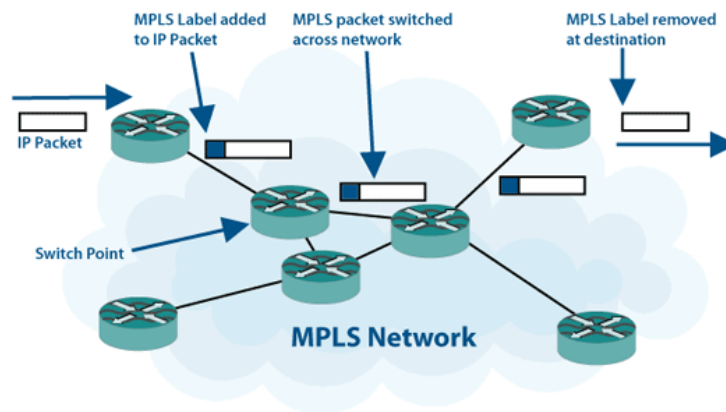


Figure 12 - MPLS switching

The main benefits are the below:

- Reduce the forwarding computational cost (the router can handle more packets per second).
- End to end quality of service – MPLS can create permanent paths for a specific traffic (VoIP, HTTP, RTP...). (Without MPLS, you can do this using PBRs (Policy Base Routing) without performance limitations).
- With multi BGP, MPLS can provide network VPN (virtual private network) using VRF (virtual routing forwarding). This is very useful to limit the visibility between customer networks without using firewalls, increasing the final throughput.
- MPLS can work with any IP routing protocol.



In MPLS the routers are classified in three categories:

- LSR (label switch router) – These are the routers that all its interfaces work with MPLS. This means that they never deliver IP packets without label.
- LER (label edge router) – These are the border routers that joins the IP network with MPLS network. Its principal function is adding (push) or removing (pop) the MPLS labels to the IP packets.
- PER (provider edge router) – They are LER routers that work with multi BGP offering VPN services.

MPLS network is itself an IP network. This means that mpls needs a IP routing protocol underneath that creates the routing table to reach all the interfaces. This protocol can be any protocol, but normally is OSPF. In our case, this protocol will be OLSR.

Once the IP routing protocol is running and the router tables have synched, we can start MPLS.

```
R3#sh ip route ospf
Gateway of last resort is 192.168.2.2 to network 0.0.0.0

 1.0.0.0/32 is subnetted, 1 subnets
O   1.1.1.1 [110/2] via 10.0.1.2, 00:02:31, FastEthernet0/1
 2.0.0.0/32 is subnetted, 1 subnets
O   2.2.2.2 [110/3] via 10.0.1.2, 00:02:31, FastEthernet0/1
 4.0.0.0/32 is subnetted, 1 subnets
O   4.4.4.4 [110/4] via 10.0.1.22, 00:02:21, FastEthernet1/0
    [110/4] via 10.0.1.2, 00:02:31, FastEthernet0/1
 5.0.0.0/32 is subnetted, 1 subnets
O   5.5.5.5 [110/2] via 10.0.1.22, 00:02:21, FastEthernet1/0
 6.0.0.0/32 is subnetted, 1 subnets
O   6.6.6.6 [110/3] via 10.0.1.22, 00:02:21, FastEthernet1/0
 7.0.0.0/32 is subnetted, 1 subnets
O   7.7.7.7 [110/3] via 10.0.1.22, 00:01:58, FastEthernet1/0
    [110/3] via 10.0.1.2, 00:01:58, FastEthernet0/1
10.0.0.0/8 is variably subnetted, 12 subnets, 2 masks
O   10.0.1.4/30 [110/2] via 10.0.1.2, 00:02:31, FastEthernet0/1
O   10.0.1.8/30 [110/3] via 10.0.1.2, 00:02:31, FastEthernet0/1
O   10.0.1.12/30 [110/3] via 10.0.1.22, 00:02:21, FastEthernet1/0
O   10.0.1.16/30 [110/2] via 10.0.1.22, 00:02:21, FastEthernet1/0
O   10.0.1.24/30 [110/2] via 10.0.1.2, 00:02:08, FastEthernet0/1
O   10.0.1.28/30 [110/2] via 10.0.1.22, 00:02:08, FastEthernet1/0
O   10.0.1.32/30 [110/3] via 10.0.1.22, 00:02:08, FastEthernet1/0
    [110/3] via 10.0.1.2, 00:01:58, FastEthernet0/1
O   10.0.1.36/30 [110/3] via 10.0.1.22, 00:01:58, FastEthernet1/0
    [110/3] via 10.0.1.2, 00:02:08, FastEthernet0/1
```

To create the mpls paths, this technology uses its own protocol called LDP (Label Distribution Protocol). With this protocol, (using the IP routing tables), create the router label table.

In this table, you have the IP destinations with its pop and push label, and the layer two next hop (MAC address) that this has been collected thanks the IP router protocol.

When this mapping has been performed, this creates a FEC (Forwarding Equivalence Class) that is how it is called this relationship.

```
R3#sh mpls forwarding-table
Local  Outgoing Prefix  Bytes Label  Outgoing  Next Hop
Label  Label   or Tunnel Id  Switched    interface
16     16     4.4.4.4/32    0           Fa0/1     10.0.1.2
17     17     4.4.4.4/32    0           Fa1/0     10.0.1.22
18     18     2.2.2.2/32    0           Fa0/1     10.0.1.2
19     Pop Label 1.1.1.1/32  127        Fa0/1     10.0.1.2
20     19     10.0.1.36/30 0           Fa0/1     10.0.1.2
21     21     10.0.1.36/30 0           Fa1/0     10.0.1.22
22     Pop Label 10.0.1.24/30 0           Fa0/1     10.0.1.2
23     27     10.0.1.12/30 0           Fa1/0     10.0.1.22
24     22     10.0.1.8/30  0           Fa0/1     10.0.1.2
25     Pop Label 10.0.1.4/30  0           Fa0/1     10.0.1.2
26     16     6.6.6.6/32    0           Fa1/0     10.0.1.22
27     Pop Label 5.5.5.5/32   348        Fa1/0     10.0.1.22
28     25     10.0.1.32/30 0           Fa0/1     10.0.1.2
29     22     10.0.1.32/30 0           Fa1/0     10.0.1.22
30     Pop Label 10.0.1.28/30 0           Fa1/0     10.0.1.22
31     Pop Label 10.0.1.16/30 0           Fa1/0     10.0.1.22
32     28     7.7.7.7/32   0           Fa0/1     10.0.1.2
33     28     7.7.7.7/32   0           Fa1/0     10.0.1.22
34     No Label 10.10.10.0/24[V] 0           aggregate/Internal
```

If the mpls network has some path with degradation, OLSR would change the next hop, and this will recalculate the label table.

## 4.5. MPLS real case

To understand better, I am going to perform a ping between two customer network A - E that are attached between a MPLS network.

To perform this, I am going to use my MPLS GNS3 network created with Cisco router 7200.

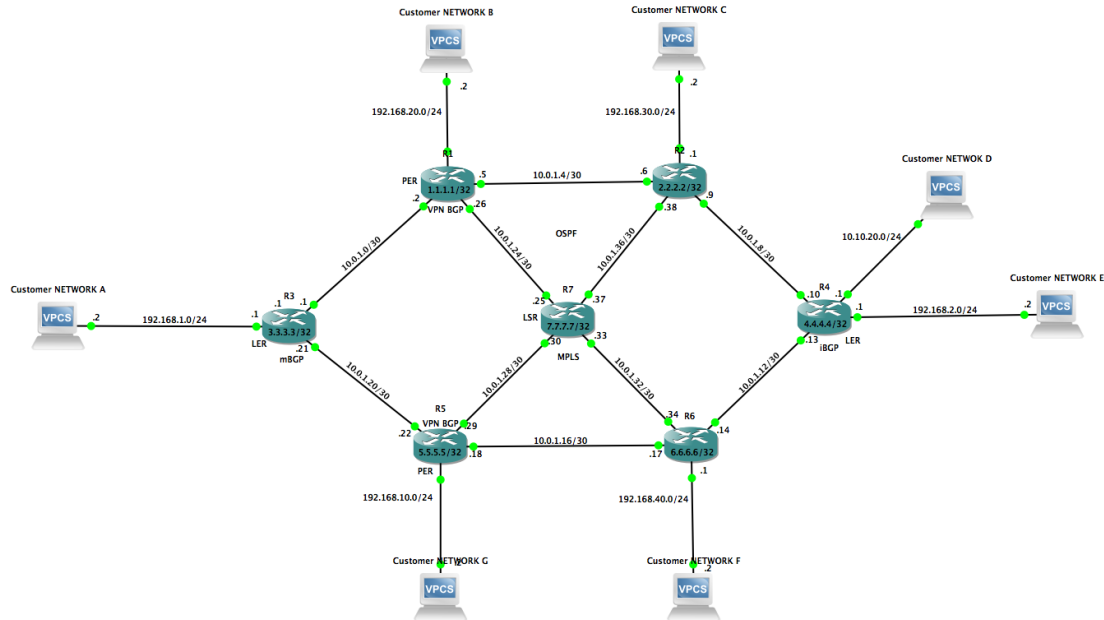


Figure 13 - GNS3 MPLS simulation

Customer network A (192.168.1.2) sends a ICMP to 192.168.2.2 (that belongs to network E).

When the ICMP packet arrives to R3 this requests the next hop:

```
R3#sh ip route 192.168.2.2
Routing entry for 192.168.2.0/24
  Known via "bgp 10", distance 200, metric 0, type internal
  Last update from 4.4.4.4 00:23:33 ago
  Routing Descriptor Blocks:
    * 4.4.4.4, from 4.4.4.4, 00:23:33 ago
      Route metric is 0, traffic share count is 1
      AS Hops 0
      MPLS label: none
```

This replies saying that we have to send the packet to 4.4.4.4 (multi BGP)

Finally, we request in the mpls forwarding table.

```
R3#sh mpls forwarding-table 4.4.4.4
Local  Outgoing Prefix      Bytes Label  Outgoing  Next Hop
Label  Label  or Tunnel Id  Switched    interface
16     16     4.4.4.4/32    0           Fa0/1     10.0.1.2
17     17     4.4.4.4/32    0           Fa1/0     10.0.1.22
```

Next hop has two possibilities:

- To 10.0.1.2 with MPLS labelled with 16 – MAC address ca01.05bf.0008
- To 10.0.1.22 with MPLS labelled with 17 – MAC address ca05.05ff.0006

As you can see in this Wireshark screen-shoot, the router chose the second path and sends the mpls packet with the label 17 to the MAC address ca05.05ff.0006.

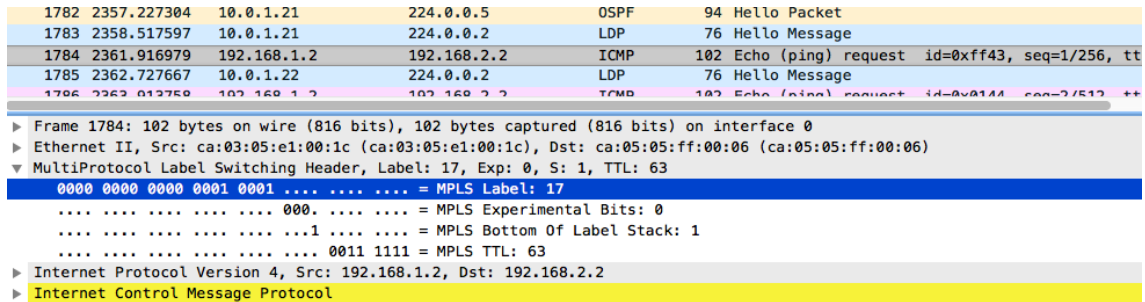


Figure 14 - MPLS label - Wireshark screen-shoot

When this MPLS packet arrives to the router 5, this will only read the mpls forwarding table.

```
R5#sh mpls forwarding-table
Local  Outgoing Prefix      Bytes Label  Outgoing  Next Hop
Label Label  or Tunnel Id      Switched    interface
16  Pop Label 6.6.6.6/32      0           Fa0/0     10.0.1.17
17  17        4.4.4.4/32      52482      Fa0/0     10.0.1.17
18  Pop Label 3.3.3.3/32      0           Fa0/1     10.0.1.21
.
.
.
```

The local column shows the inbound mpls label (in our case is 17) and says that you have to send with the same label (outgoing label) to the interface Fa0/0 to the mac address of the IP 10.0.1.17 (ca06.060e.0006).

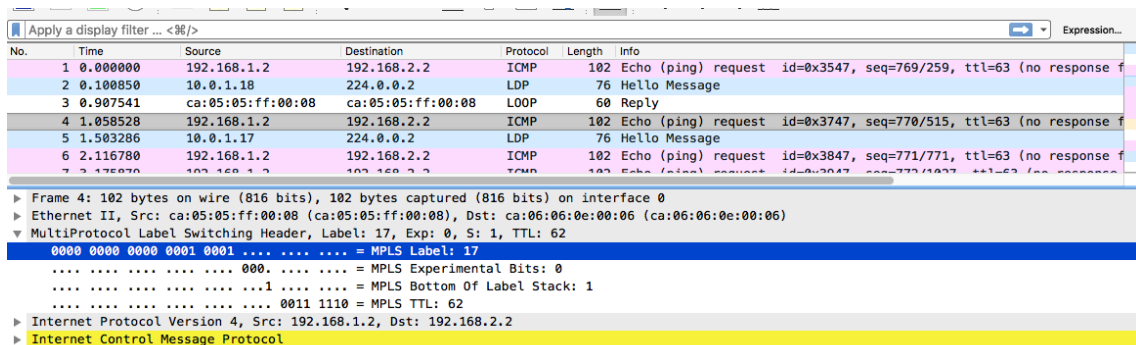


Figure 15 - MPLS label - Wireshark screen-shoot

This would continue until to arrive to the last MPLS router (PER or LER) that would perform the pop label (removing the label).

## 4.6. New configuration and deployment plan

After performed the below task:

- Testing BATMAN and OLSR behavior with a real simulation
- Testing MPLS network with the GN3 simulation with real Cisco IOS
- Screening the wireless cost-effective equipment and available routers.

I am willing to offer a final proposal.

My recommendation is to create the network with the below configuration:

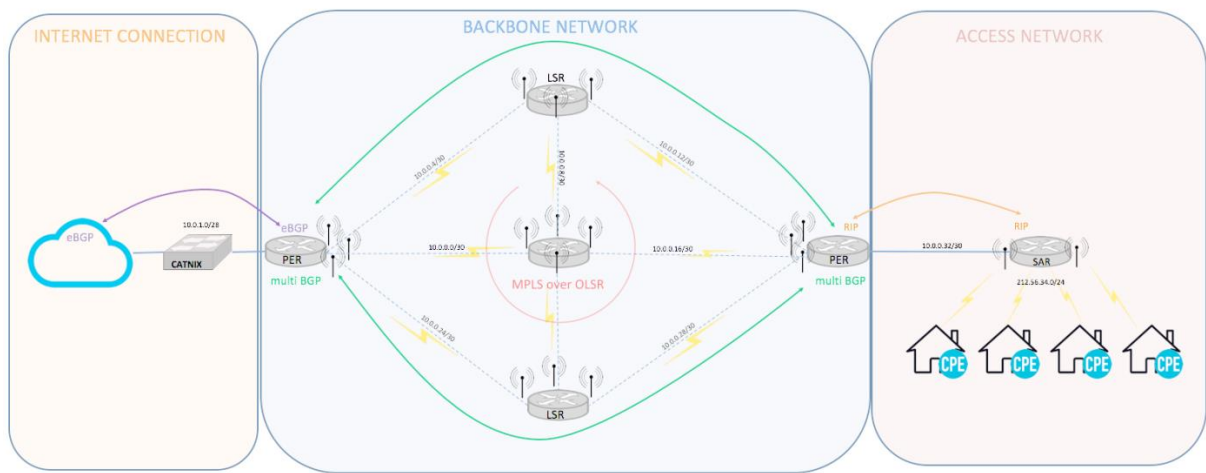


Figure 16 - Final MPLS OLSR schema design

### 4.6.1. Internet connection:

- The connection is provided by the CATNIX and Guifinet must share its own public IPs via eBGP.
- In order to economize the IPv4 address, it's highly recommended to work with a CGNAT (carrier grade NAT).
- Public IPv6 must be provided to all end-customers.

### 4.6.2. Backbone network:

- The backbone network must be deployed forming a mesh topology with N LOS (line of sight) radio-links with a negotiated speed of 1Gigabit Ethernet (ideally).
- To facilitate the routing all the redundant paths should have the same speed. Otherwise, the differences must be places in the internal OLSR metric.

- The RF band should be UHF to the band radio-link band 24GHz, 11GHz.
- The Wi-Fi devices has to work in bridge mode (layer 2), and these must be connected to the router interface.
- The router must be capable to work the protocols MPLS, OLSR and multi BGP (in case this is a PER (provider edge router)).
- The radio-link addressing must be a /30, all the subnetting must be efficiency to save routing table memory (supernetting).
- Deploy the network either IPv4 and IPv6 (focus to IPv6).

#### **4.6.3. Access network:**

- The MPLS border router must be connect via point to point with the access router.
- The access router should split its customer subnet range into different subnets. These subnets may be place in different interfaces or one interface with different VLANs.
- The access wireless technology, instead of Wi-Fi IEEE 802.11, should be LTE (long term evolution) because this technology handles much better the access medium than Wi-Fi (CSMA/CA) offering better end customer experience, and much easier to create traffic prioritization.

## 4.7. Recommended equipment

### 4.7.1. Routers

Unfortunately, OLSR only works in Linux O.S. So, it's impossible to use professional routers such as Cisco, Juniper, Mikrotik, Alcatel or Huawei.

The solution is to create our own routers with reliable hardware such as Intel Xeon architecture, and install the OLSR, MPLS and BGP daemons in a Linux machine.

The interfaces would Gigabit Ethernet PCI-express that would connect to the radio-links.

### 4.7.2. Backbone radio links

Ubiquiti offer its AirFiber® radio-links. These devices work with the RF bands 5GHz and 24GHz and offer up to 2Gbps links.

The mesh network could work perfectly with these devices.

The computer router would connect every AirFiber® to one gigabit Ethernet interface, and this will run the OLSR, MPLS, and the additional protocols.



Figure 17 - Ubiquiti Airfiber link

### **4.7.3. Access network**

Regarding LTE technology, any vendor (Huawei, Cisco, Alcatel) would be suitable. If finally, the LTE option is out the budget, I recommend Ubiquiti as Wi-Fi IEEE 802.11 vendor with its product rocket M with sectorial antenna configuration.

### **4.8. Environment impact**

This project has an environment impact quite low. This is due that all the equipment is low consumption and it's not necessary to do work-roads.

So, we can say that this project has a better impact to the environment than a normal ISP that needs to install fiber and the equipment needs more energy (but, obviously it offers better performance).

### **4.9. Conclusions**

It's quite clear that community networks are networks that when they were born, nobody can predict its success. So, it's quite normal that these projects have coordination issues.

In this memory, we have explained the different issues and we offered a realistic proposal basing that these kind of network work with low budgets.

As we saw, the wireless IP routing protocols are still experimental protocols, and the important vendors such as Cisco, Alcatel or Juniper have not developed their own implementation.

This provides high level of difficult because, currently, the only routers that can support MPLS over OLSR and multi BGP are router based in Linux. This is definitely a disadvantage, because, the hardware reliability will be lower that a full test Cisco router and also this will provide more maintenance.

Guifinet has to evolve to a WISP network becoming a reliable network no using domestic Wi-Fi IEEE 802.11 and moving to LTE (long term evolution).

Regarding my own engineer conclusion, this project has improved my IP routing and MPLS skills. Now I feel capable to deploy an end two end MPLS network either wireless or wireline.



# **ANNEXES**

## **Network schema**

In this chapter, we add the same figures in din A3.

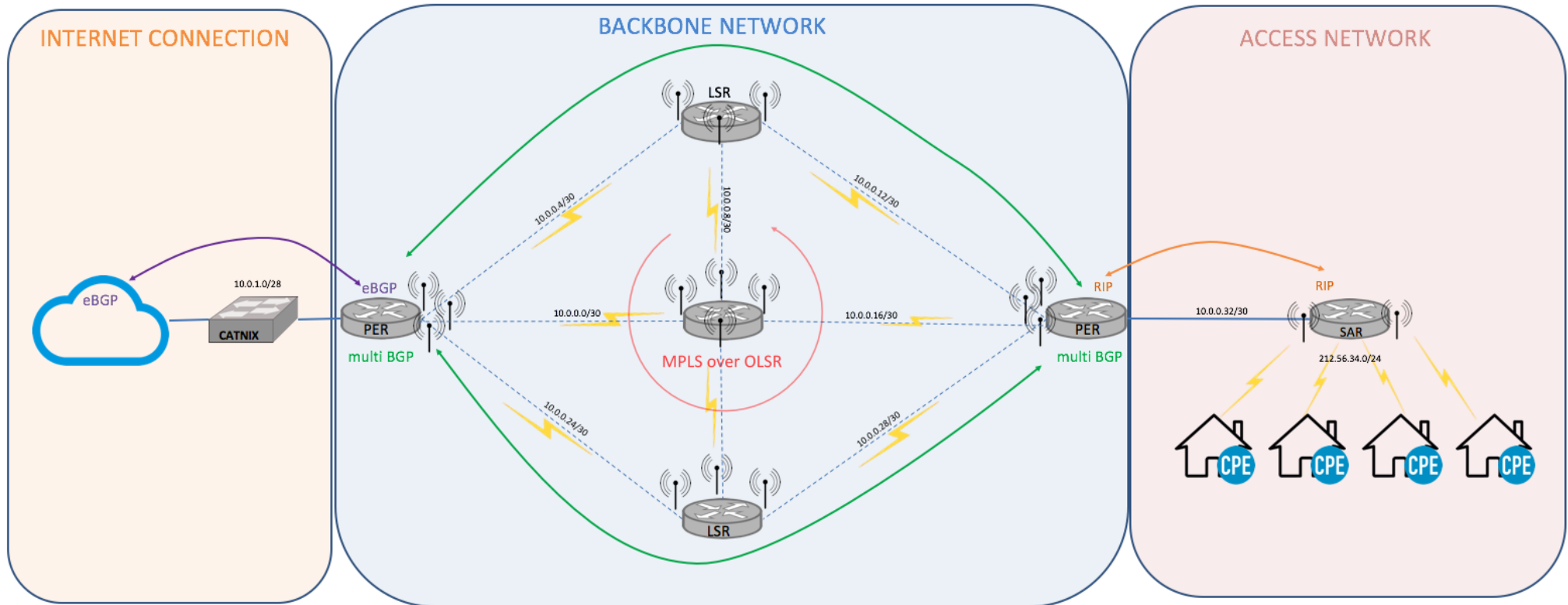


Figure 18 - Final MPLS OLSR schema design (ZOOM picture)

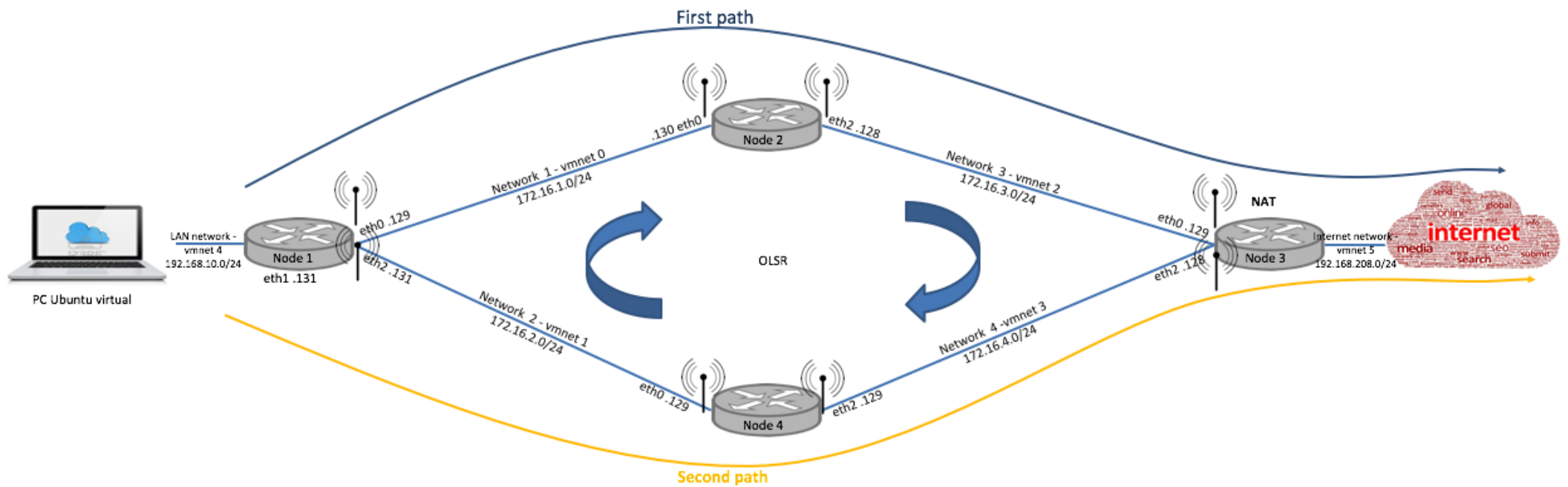


Figure 19 - Simulation network schema (ZOOM)

## **Linux VMware BATMAN and OSLR simulation**

### **Introduction**

In this annex, let's show the schema and configuration that have been necessary to create these two simulations (BATMAN-ADV and OLSR).

### **Scenario**

To test these protocols, we have configured a virtual scenario using the software VMware Workstation. Thanks to this application it was able to create several virtual IP subnets forming a loop without to use any physical network device and connect a virtual final end user device (Ubuntu machine) in order to test the final end customer experience.

To simulate these two protocols, we used four Debian 7 operation systems that worked as routers (nodes).

The router number 1 will have an additional interface in order to connect to the Ubuntu end user machine. So, this router will work as CPE (Customer Premises Equipment) but without NAT capability.

Also, the router number 3, will have an additional interface with NAT outside feature that will connect to Internet. The reason of this NAT is that we don't want to propagate the internal simulation IP subnet prefixes.

Apart of all the necessary interfaces in order to make the loop; it has been necessary to create an additional network that works as controlling layer in order to manage the simulation, handle and create degradation situations in order to see how the protocol acts.

## VMWare installation

- The real machine (hypervisor) has the below features.
- Ubuntu 15.04 64bits - 3.19.0-59-generic
- VMware Workstation 11
- 16GB RAM
- Intel i7-4710HQ CPU @ 2.50GHz

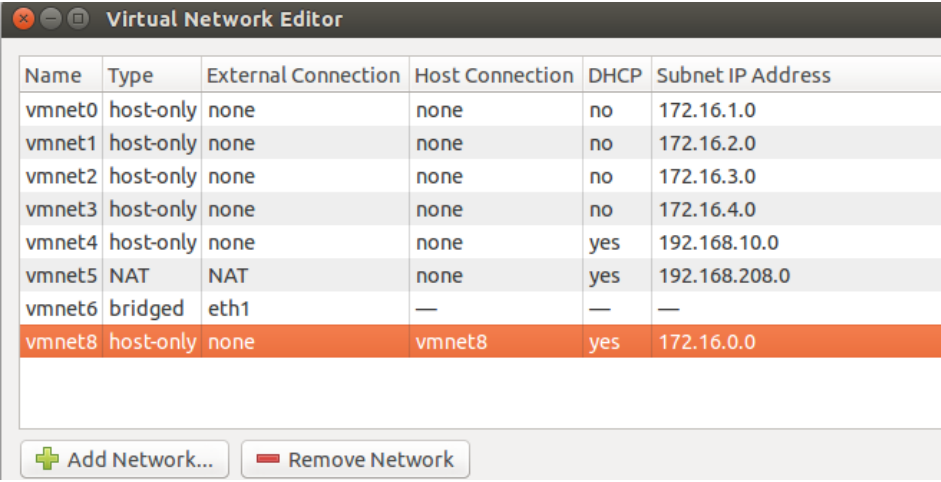
In order to create the simulation, these are the main steps:

- Install the VMware Workstation (Linux) or VMware Fusion (Mac)
- Create the VMware network schema (described below)
- Install the nodes or routers with the below characteristics.
- Debian 7.9 - 3.2.0-4-amd64
- 1 vCPU
- 256MB RAM
- 3 or 4 Ethernet interfaces
- Configure the interface file (described below).
- Install the OLSR and BATMAN ADV daemons (described below)

## Creating the network schema

To do this, we have to open the VMware Virtual network editor to create four networks without connection with the real machine and a network with connection to physical machine and Internet connection with NAT.

This last network was used to manage the nodes by SSH, and download the necessary packet from APT to install the mesh protocols.



Name	Type	External Connection	Host Connection	DHCP	Subnet IP Address
vmnet0	host-only	none	none	no	172.16.1.0
vmnet1	host-only	none	none	no	172.16.2.0
vmnet2	host-only	none	none	no	172.16.3.0
vmnet3	host-only	none	none	no	172.16.4.0
vmnet4	host-only	none	none	yes	192.168.10.0
vmnet5	NAT	NAT	none	yes	192.168.208.0
vmnet6	bridged	eth1	—	—	—
vmnet8	host-only	none	vmnet8	yes	172.16.0.0

Figure 20 - VMware network designer

In this picture, we see the below networks:

- Vmnet0, 1, 2 and 3 are the internal IP subnets that make the network loop.
- Vmnet4 is the network that joins the router 1 with the virtual end user machine (Ubuntu OS).
- Vmnet5 connects the router 3 with Internet.
- Vmnet8 connects all the devices via SSH (simulator managing network).

## Configuring the network

1. Active the IP forwarding adding the below configure line and reboot the device.

```
echo "net.ipv4.ip_forward=1" >> /etc/sysctl.conf
```

2. Load the below interface configuration file to every node.

### Node 1

```
root@Nodo1:~# cat /etc/network/interfaces
auto lo
iface lo inet loopback

#SSH Management network
auto eth1
iface eth1 inet static
address 172.16.0.128
netmask 255.255.255.0

##Ubuntu PC connection
auto eth3
iface eth3 inet static
address 192.168.10.1
netmask 255.255.255.0

##OLSR interface 1
auto eth0
iface eth0 inet static
address 172.16.1.129
netmask 255.255.255.0

##OLSR interface 2
auto eth2
iface eth2 inet static
address 172.16.2.131
netmask 255.255.255.0
```

### Node 2

```
root@Nodo2:~# cat /etc/network/interfaces
```

```
auto lo
iface lo inet loopback

#SSH Management network
allow-hotplug eth1
iface eth1 inet static
address 172.16.0.129
netmask 255.255.255.0

#OLSR interface 1
allow-hotplug eth0
iface eth0 inet static
address 172.16.1.130
netmask 255.255.255.0

#OLSR interface 2
allow-hotplug eth2
iface eth2 inet static
address 172.16.3.128
netmask 255.255.255.0
```

### Node 3

```
root@Nodo3:~# cat /etc/network/interfaces
auto lo
iface lo inet loopback

#SSH Management network
allow-hotplug eth1
iface eth1 inet static
address 172.16.0.131
netmask 255.255.255.0

#Internet gateway
allow-hotplug eth3
iface eth3 inet dhcp

#OLSR interface 1
allow-hotplug eth0
iface eth0 inet static
address 172.16.3.129
netmask 255.255.255.0

#OLSR interface 2
allow-hotplug eth2
iface eth2 inet static
address 172.16.4.128
netmask 255.255.255.0

post-up iptables -t nat -A POSTROUTING -o eth3 -j MASQUERADE
```

### Node 4

```
root@Nodo4:~# cat /etc/network/interfaces
```

```
auto lo
iface lo inet loopback

#SSH Management network
allow-hotplug eth1
iface eth1 inet static
address 172.16.0.130
netmask 255.255.255.0

#OLRS interface 1
allow-hotplug eth0
iface eth0 inet static
address 172.16.2.129
netmask 255.255.255.0

#OLRS interface 2
allow-hotplug eth2
iface eth2 inet static
address 172.16.4.129
netmask 255.255.255.0
```

3. Install a DHCP server in the node 1 in order to provide IP subnet to the Ubuntu client machine and configure it.

```
apt-get install isc-dhcp-server

nano /etc/dhcp/dhcpd.conf

subnet 192.168.10.0 netmask 255.255.255.0 {
    range 192.168.10.2 192.168.10.10;
    option routers 192.168.10.1;
    option subnet-mask 255.255.255.0;
    option domain-name-servers 8.8.8.8;
}
```

4. Finally, install, tcpdump and bwm-ng.

```
aptitude install bwm-ng tcpdump
```



# OLSR simulation

## Installation

1. Install the protocol typing this below command.

```
aptitude install olsrd
```

2. Activate the OLSR daemon.

```
root@Nodol1:~# cat /etc/default/olsrd
START_OLSRD="YES"
DEBUGLEVEL=0
DAEMON_OPTS="-f /etc/olsrd/olsrd.conf -d $DEBUGLEVEL"
```

3. Load the proper configuration to every node.

## Node 1:

```
root@Nodol1:~# cat /etc/olsrd/olsrd.conf |grep -v "#"

DebugLevel 0
Interface "eth0" "eth2"
{
    HelloInterval          6.0
    HelloValidityTime      600.0
    TcInterval              0.5
    TcValidityTime         300.0
    MidInterval            10.0
    MidValidityTime        300.0
    HnaInterval            10.0
    HnaValidityTime        300.0
}
LinkQualityFishEye 1
IpVersion 4
ClearScreen yes

Hna4
{
    192.168.10.0 255.255.255.0
}
Hna6
{
}

AllowNoInt yes
Willingness 7
IpcConnect
{
    MaxConnections 0
    Host 127.0.0.1
}
UseHysteresis no
LinkQualityLevel 2
Pollrate 0.1
```

```
TcRedundancy2
MprCoverage 5
```

## Node 2:

```
root@Nodo2:~# cat /etc/olsrd/olsrd.conf |grep -v "#"
DebugLevel 0
Interface "eth0" "eth2"
{
    HelloInterval      6.0
    HelloValidityTime  600.0
    TcInterval         0.5
    TcValidityTime     300.0
    MidInterval        10.0
    MidValidityTime    300.0
    HnaInterval        10.0
    HnaValidityTime    300.0
}
LinkQualityFishEye 1
IpVersion 4
ClearScreen yes
Hna4
{
}
Hna6
{
}
AllowNoInt yes

Willingness 7
IpcConnect
{
    MaxConnections 0
    Host 127.0.0.1
}
UseHysteresis no
LinkQualityLevel 2
Pollrate 0.1
TcRedundancy2
MprCoverage 5
```

### Node 3:

```
root@Nodo3:~# cat /etc/olsrd/olsrd.conf |grep -v "#"
DebugLevel 0
Interface "eth0" "eth2"
{
    HelloInterval      6.0
    HelloValidityTime  600.0
    TcInterval         0.5
    TcValidityTime     300.0
    MidInterval        10.0
    MidValidityTime    300.0
    HnaInterval        10.0
    HnaValidityTime    300.0
}
LinkQualityFishEye 1
IpVersion 4
ClearScreen yes

Hna4
{
    0.0.0.0      0.0.0.0
}

Hna6
{
}
AllowNoInt yes
Willingness 7
IpcConnect
{
    MaxConnections 0
    Host           127.0.0.1
}
UseHysteresis no
LinkQualityLevel 2
Pollrate 0.1
TcRedundancy2
MprCoverage 5
```

## Node 4:

```
root@Nodo4:~# cat /etc/olsrd/olsrd.conf |grep -v "#"
DebugLevel 0
Interface "eth0" "eth2"
{
    HelloInterval      6.0
    HelloValidityTime  600.0
    TcInterval         0.5
    TcValidityTime     300.0
    MidInterval        10.0
    MidValidityTime    300.0
    HnaInterval        10.0
    HnaValidityTime    300.0
}
LinkQualityFishEye 1
IpVersion 4
ClearScreen yes
Hna4
{
}
Hna6
{
}

AllowNoInt yes
Willingness 7
IpcConnect
{
    MaxConnections 0
    Host           127.0.0.1
}
UseHysteresis no
LinkQualityLevel 2
Pollrate 0.1
TcRedundancy2
MprCoverage 5
```

## Troubleshooting

In order to check if the daemon is running, you can type the command:

```
netstat -lunp
```

This command shows all the UDP sockets opened. If the OLSR daemon is running, you should observe one socket per interface.

```
root@Nodol1:~# netstat -lunp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
udp        0      0 172.16.2.131:698       0.0.0.0:*                2084/olsrd
udp        0      0 0.0.0.0:698           0.0.0.0:*                2084/olsrd
udp        0      0 172.16.1.129:698      0.0.0.0:*                2084/olsrd
udp        0      0 0.0.0.0:698           0.0.0.0:*                2084/olsrd
```

Also, in order to check if the node is sharing data with the rest of device, you can check the routing table.

```
root@Nodol1:~# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref
Use Iface
0.0.0.0 172.16.0.2 0.0.0.0 UG 0 0
0 eth1
172.16.0.0 0.0.0.0 255.255.255.0 U 0 0
0 eth1
172.16.1.0 0.0.0.0 255.255.255.0 U 0 0
0 eth0
172.16.1.130 172.16.1.130 255.255.255.255 UGH 2 0
0 eth0
172.16.2.0 0.0.0.0 255.255.255.0 U 0 0
0 eth2
172.16.2.129 172.16.2.129 255.255.255.255 UGH 2 0
0 eth2
172.16.3.128 172.16.1.130 255.255.255.255 UGH 2 0
0 eth0
172.16.3.129 172.16.2.129 255.255.255.255 UGH 2 0
0 eth2
172.16.4.128 172.16.2.129 255.255.255.255 UGH 2 0
0 eth2
172.16.4.129 172.16.2.129 255.255.255.255 UGH 2 0
0 eth2
```

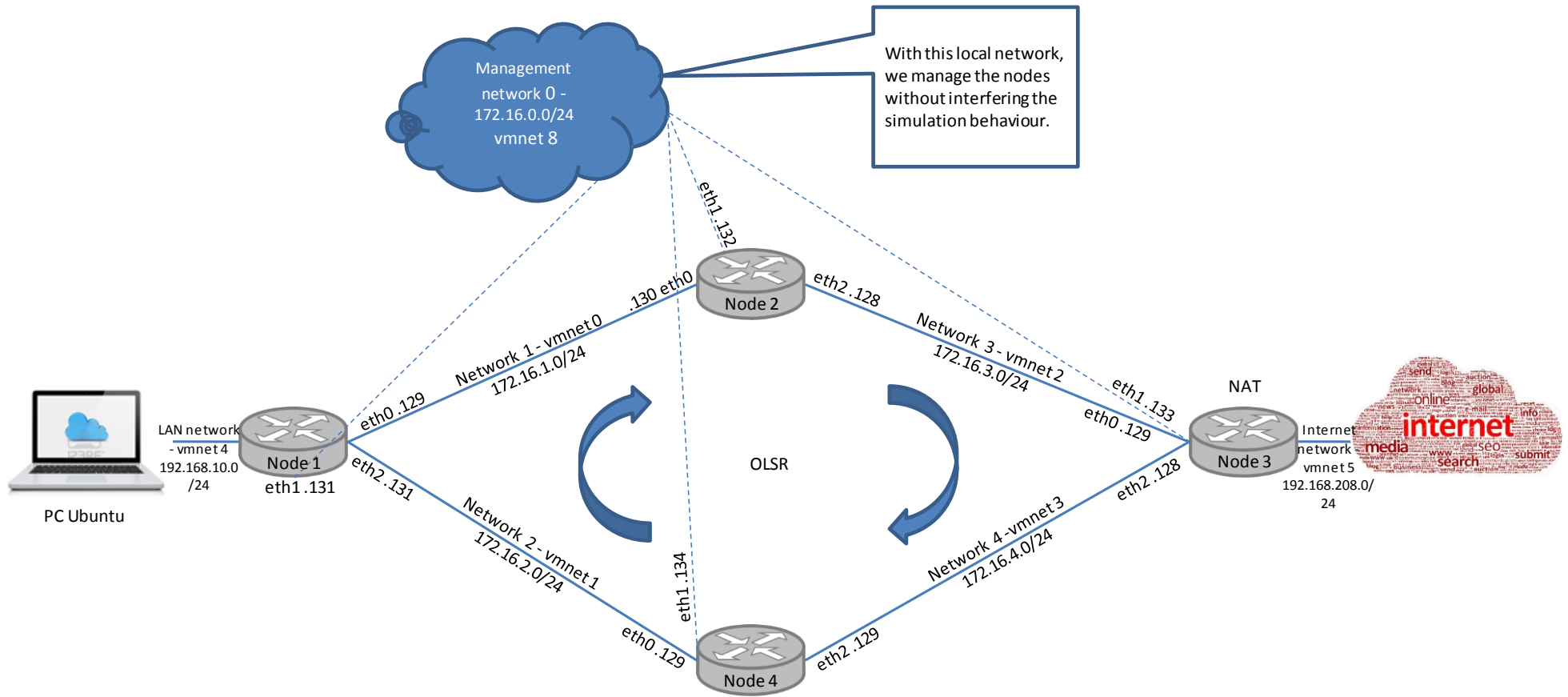


Figure 21 - OLSR simulation

## Batman-adv simulation

### Installation

1. Install the below packages.

```
aptitude install batmand batctl
```

2. Check that the module is loaded.

```
root@Nodo1:~/batman# lsmod |grep batman

batman_adv                81025  0

crc16                     12343  2 ext4,batman_adv
```

3. If the module is loaded automatically, do manually.

```
root@Nodo2:~/batman# cat /etc/modules
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be
loaded
# at boot time, one per line. Lines beginning with "#" are
ignored.
# Parameters can be specified after the module name.

loop
batman_adv
```

4. Configure the proper configuration per Node.

### Node 1:

```
root@Nodo1:~/batman# cat start_batman
#!/bin/bash

#configure the mtu max size (its a mandatory require to ensure
the BATMAN is going to work fine)
ifconfig eth0 mtu 1500
ifconfig eth2 mtu 1500

#Add the node interfaces that you want to add in the protocol.
batctl if add eth0
batctl if add eth2
```

## Node 2:

```
root@Nodo2:~/batman# cat start_batman
#!/bin/bash

#configure the mtu max size (it's a mandatory require to ensure
the BATMAN is going to work fine)
ifconfig eth0 mtu 1500
ifconfig eth2 mtu 1500

#Add the node interfaces that you want to add in the protocol.
batctl if add eth0
batctl if add eth2

#Activate the interfaces without IP stack.
ifconfig eth0 up
ifconfig eth2 up

#Create a IP and active the interface.
ifconfig bat0 192.168.0.2 netmask 255.255.255.0
ifconfig bat0 up
#Activate the interfaces without IP stack.
ifconfig eth0 up
ifconfig eth2 up

#Create a IP and active the interface.
ifconfig bat0 192.168.0.1 netmask 255.255.255.0
ifconfig bat0 up

#Active the IP forwarding to get access to the PC client
echo 1 > /proc/sys/net/ipv4/ip_forward
ip route add default via 192.168.0.3
```

## Node 3:

```
root@Nodo3:~/batman# cat start_batman
#!/bin/bash
root@Nodo3#configure the mtu max size (it's a mandatory require
to ensure the BATMAN is going to work fine)
ifconfig eth0 mtu 1500ifconfig eth2 mtu 1500

#Add the node interfaces that you want to add in the protocol.
batctl if add eth0
batctl if add eth2

#Activate the interfaces without IP stack.
ifconfig eth0 up
ifconfig eth2 up

#Create a IP and active the interface.
ifconfig bat0 192.168.0.3 netmask 255.255.255.0
ifconfig bat0 up

#Activate the IP forwarding to get access to Internet and NAT
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -o eth3 -j MASQUERADE
ip route add 192.168.10.0/24 via 192.168.0.1
```



## Node 4:

```
root@Nodo4:~/batman# cat start_batman
#!/bin/bash

#configure the mtu max size (it's a mandatory require to ensure
the BATMAN is going to work fine)
ifconfig eth0 mtu 1500
ifconfig eth2 mtu 1500

#Add the node interfaces that you want to add in the protocol.
batctl if add eth0
batctl if add eth2

#Activate the interfaces without IP stack.
ifconfig eth0 up
ifconfig eth2 up

#Create a IP and active the interface.
ifconfig bat0 192.168.0.4 netmask 255.255.255.0
ifconfig bat0 up
```

## Configuration

When it starts, the system creates a new interface named bat0.

This new interface has a relationship with all the nodes (Ethernet and Wi-Fi interfaces that have been joined to BATMAN protocol before). (In the schema design, is able to see this behavior).

To initialize the bat0 interface, it needs to execute the following bash script.

```
#!/bin/bash
#configure the mtu max size (it's a mandatory require to ensure
the BATMAN is going to work fine)
ifconfig eth0 mtu 1528
ifconfig eth2 mtu 1528
#Add the node interfaces that you want to add in the protocol.
batctl if add eth0
batctl if add eth2

#Activate the interfaces without IP stack.
ifconfig eth0 up
ifconfig eth2 up

#Create a IP and active the interface.
ifconfig bat0 192.168.0.1 netmask 255.255.255.0
ifconfig bat0 up
```

This script executes the node BATMAN behaviour. Then, the protocol is starting to send packets for all its real interfaces controlled by bat0 interface to create the mesh topology.

To start the simulation, it has to execute the batman-config script on every node.

## Node 1

```
root@Nodo1:~/batman# cat batman-config
#!/bin/bash
ifconfig eth0 mtu 1528
ifconfig eth2 mtu 1528

batctl if add eth0
batctl if add eth2

ifconfig eth0 up
ifconfig eth2 up

ifconfig bat0 192.168.0.1 netmask 255.255.255.0
ifconfig bat0 up

root@Nodo1:~# bash batman/batman-config
```

## Node 2

```
root@Nodo2:~/batman# cat batman-config
#!/bin/bash
ifconfig eth0 mtu 1528
ifconfig eth2 mtu 1528

batctl if add eth0
batctl if add eth2

ifconfig eth0 up
ifconfig eth2 up

ifconfig bat0 192.168.0.2 netmask 255.255.255.0
ifconfig bat0 up

root@Nodo2:~# bash batman/batman-config
```

## Node 3

```
root@Nodo3:~/batman# cat batman-config
#!/bin/bash
ifconfig eth0 mtu 1528
ifconfig eth2 mtu 1528
```

```
batctl if add eth0
batctl if add eth2

ifconfig eth0 up
ifconfig eth2 up

ifconfig bat0 192.168.0.3 netmask 255.255.255.0
ifconfig bat0 up

root@Nodo3:~# bash batman/batman-config
```

## Node 4

```
root@Nodo4:~/batman# cat batman-config
#!/bin/bash
ifconfig eth0 mtu 1528
ifconfig eth2 mtu 1528

batctl if add eth0
batctl if add eth2

ifconfig eth0 up
ifconfig eth2 up

ifconfig bat0 192.168.0.4 netmask 255.255.255.0
ifconfig bat0 up

root@Nodo4:~# bash batman/batman-config
```

## Tweaking and protocol behaviour

After starting the simulation, it has to check if the protocol is working as well as expect. To do that, Batman-adv offers the “batctl” command, which provide a set of debug and information tools to make troubleshooting and measure its performance.

There are two important commands, which show the actual local and global translation table.

- This provides the local announcements. In this case, we have a batman interface, therefore, there’s only one.

```
root@Nodol1:~/batman# batctl tl
Locally retrieved addresses (from bat0) announced via TT (TTVN:
1):
* 8e:8b:e4:b0:06:a4 [.P...
```

- Here, it can see the three announcements, which receive from the rest of nodes.

```
root@Nodol1:~/batman# batctl tg
Globally announced TT entries received via the mesh bat0
      Client          (TTVN)          Originator          (Curr TTVN)
Flags
* be:7b:e0:15:b6:1f  ( 1) via 00:0c:29:7c:e4:33  ( 1)
[...]
* 46:b7:88:2e:76:70  ( 1) via 00:0c:29:bf:60:46  ( 1)
[...]
* 6a:d5:3a:04:73:a7  ( 1) via 00:0c:29:34:46:f8  ( 1)
[...]
```

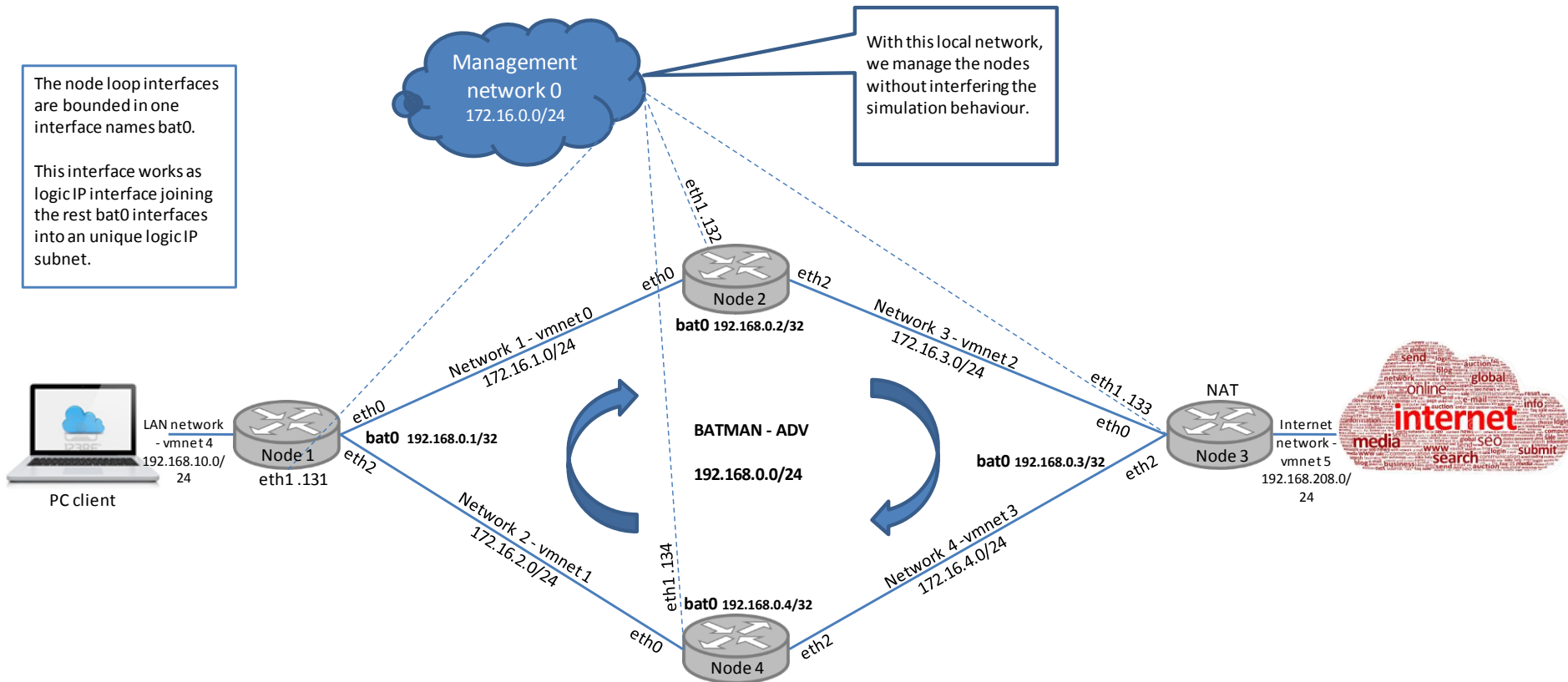


Figure 22 - BATMAN simulation

# Netdata routers data

## Traffic idle consumption

In this test, we want to check how the BATMAN and OLSR traffic consumption is.

### OLSR

Average 3.5kbps.



Figure 23 - OLSR traffic IDLE mode

### BATMAN

Average 1.88kbps.

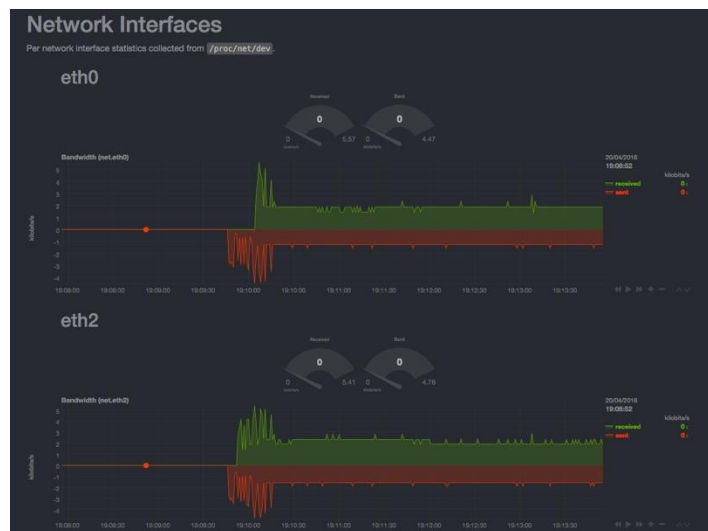


Figure 24 - BATMAN traffic IDLE mode

## PER (packet error rate) first path 0% and second path 0%

### OLSR

All the traffic goes only to one path. This is chosen by the node 1 randomly. I observed that sometimes takes the first and sometimes the second one. Both paths have the same PER, jitter and delay. It works very well.

TEST → **Passed.**

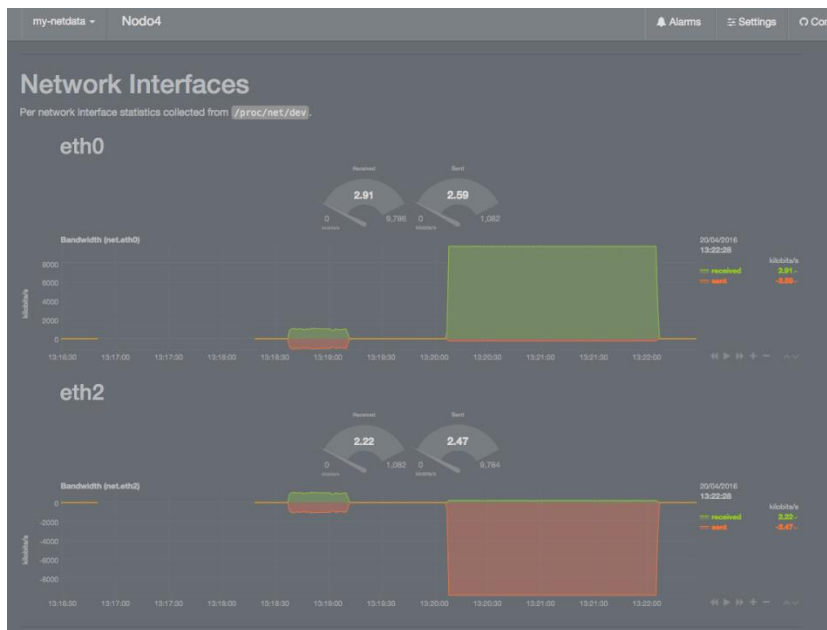


Figure 25 - OLSR - packet error rate 0% - path 1



Figure 26 - OLSR - packet error rate 0% - path 2

## BATMAN

In this case, node 1 balances the traffic to both paths (I don't know which parameters it takes to select one of them, but the final throughput does not change).

I think that this is not the correct way. For TCP/RTT measure and end to end performance (packet per second) is better to send the traffic always for the same path.

Test → **No passed.**



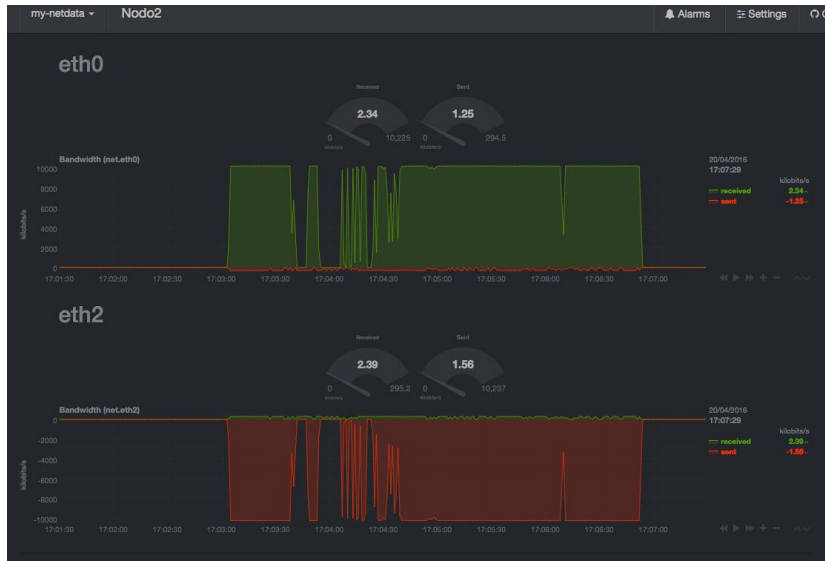


Figure 27 - BATMAN - packet error rate 0% - path 1



Figure 28 - BATMAN - packet error rate 0% - path 2

## PER (packet error rate) first path (node 2) 4% and second path (node 4) 0%:

This test starts in the interface with PER (I shut down the second path).

The expected result is that the protocol switches to the good path.

### TC command:

```
root@Nodo2:~# tc qdisc add dev eth0 root netem loss 2%
root@Nodo2:~# tc qdisc add dev eth2 root netem loss 2%
```

### OLSR

It works perfect. The TCP connection started in the path with errors, and after I active the second path, after 57s OLSR moved the traffic to the second path.

Test → **Passed**



Figure 29 - OLSR - packet error rate 4% - path 1



Figure 30 - OLSR - packet error rate 0% - path 2

## BATMAN

This protocol switches to the second path after 55s since I activated the second path.

Test → **Passed**

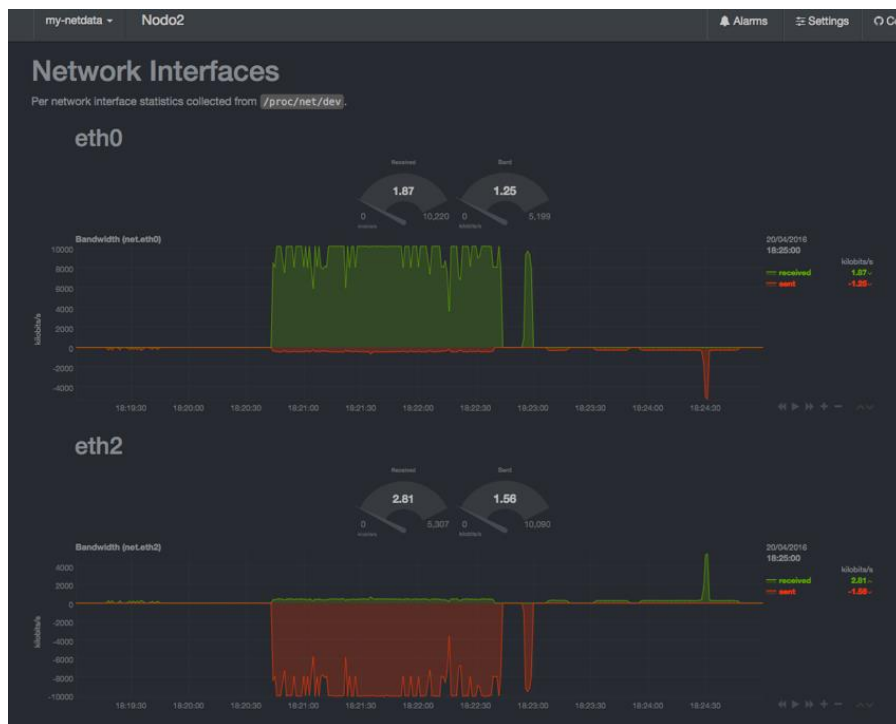


Figure 31 - BATMAN packet error rate 4% - path 1



Figure 32 - BATMAN packet error rate 0% - path 2

## PER (packet error rate) first path (node 2) 20% and second path (node 4) 0%

```
root@Nodo2:~# tc qdisc add dev eth0 root netem loss 10%
root@Nodo2:~# tc qdisc add dev eth2 root netem loss 10%
```

### OLSR:

Convergence time was 42s.

Test → **Passed.**



Figure 33 - OLSR packet error rate 20% - path 1



Figure 34 - OLSR packet error rate 0% - path 2

# BATMAN

Convergence time was 46s.

Test → **Passed.**

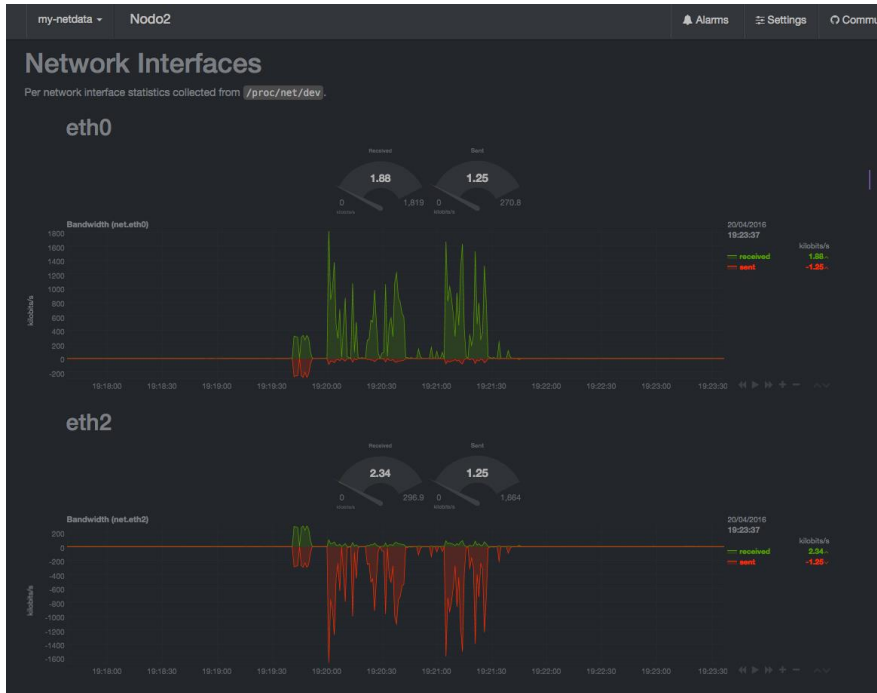


Figure 35 - BATMAN packet error rate 20% - path 1

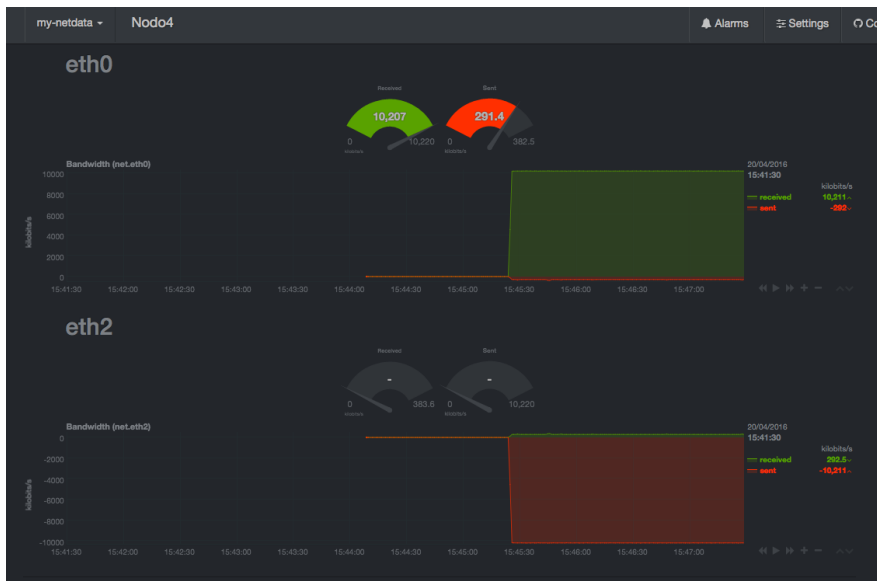


Figure 36 - BATMAN packet error rate 0% - path 2

# Measuring the protocol behavior with RTT

This test starts in the interface with PER (I shut down the second path).

```
tc qdisc add dev eth0 root netem delay 200ms
tc qdisc add dev eth2 root netem delay 200ms
```

**RTT (round trip time) first path (node 2) 20ms and second path (node 4) 0.5ms:**

**OLSR**

Path change after 45s.

Test → **Passed.**

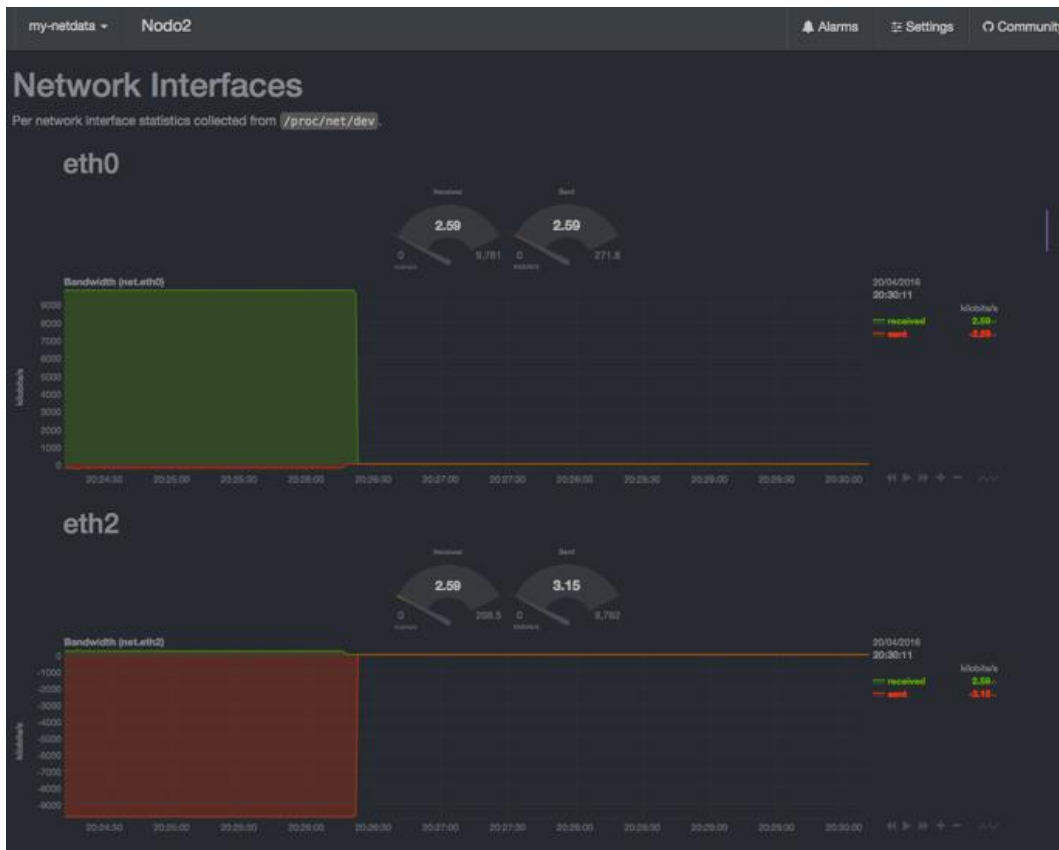


Figure 37 - OLSR RTT 20ms - path 1





Figure 38 - OLSR RTT 0,5ms - path 2

## BATMAN

It does not switch to second path (node 4) (it keeps in the highest RTT path (node 2)) and only send some traffic to the second path (node 4).

Test → **No passed.**



Figure 39 - BATMAN RTT 20ms - path 1



Figure 40 - BATMAN RTT 0,5ms - path 2

**RTT (round trip time) first path (node 2) 40ms and second path (node 4) 0.5ms**

## OLSR

When I turn the second path on, suddenly the OLSR moved the traffic to the second path, but after 1s, it went back to the first one, and after 44s moved all the traffic to the second one.

Test → **Passed.**

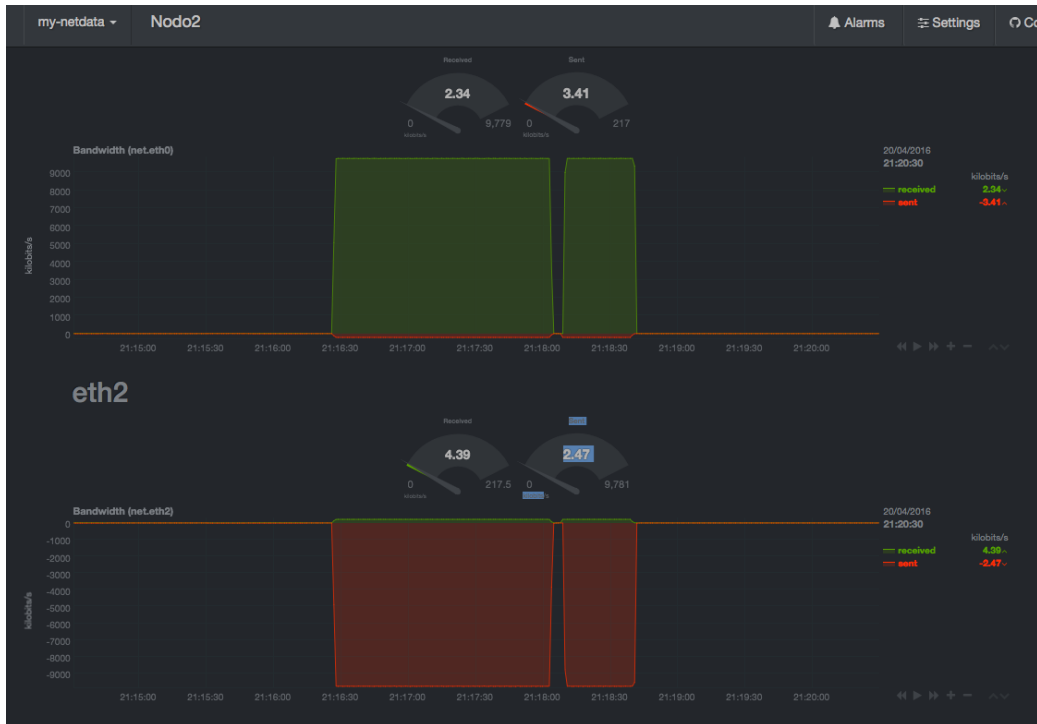


Figure 41 - OLSR RTT 40ms - path 1

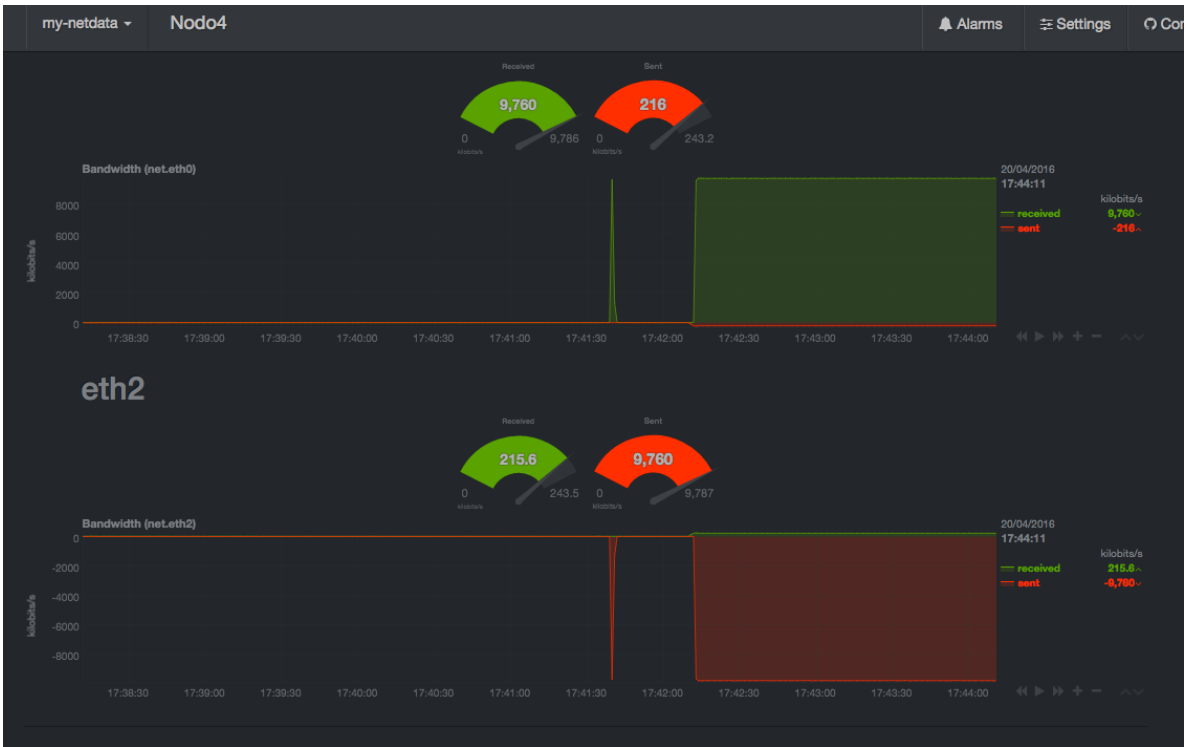


Figure 42 - OLSR RTT 0,5ms - path 2

# BATMAN

In this case, after I turned the second path on, the protocol started to send some traffic to the second one, and after 90s, it started to balance the traffic to both paths, but giving more priority the first one.

Test → **No passed.**

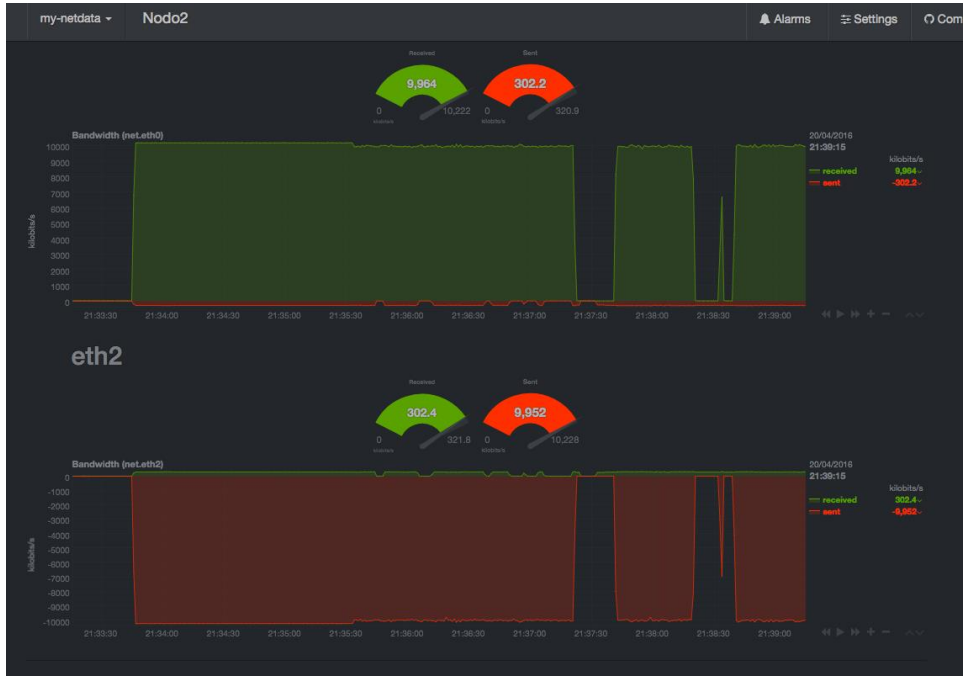


Figure 43 - BATMAN RTT 40ms - path 1



Figure 44 - BATMAN RTT 0,5ms - path

RTT (round trip time) first path (node 2) 60ms and second path (node 4) 0.5ms

OLSR

Test → **passed.**

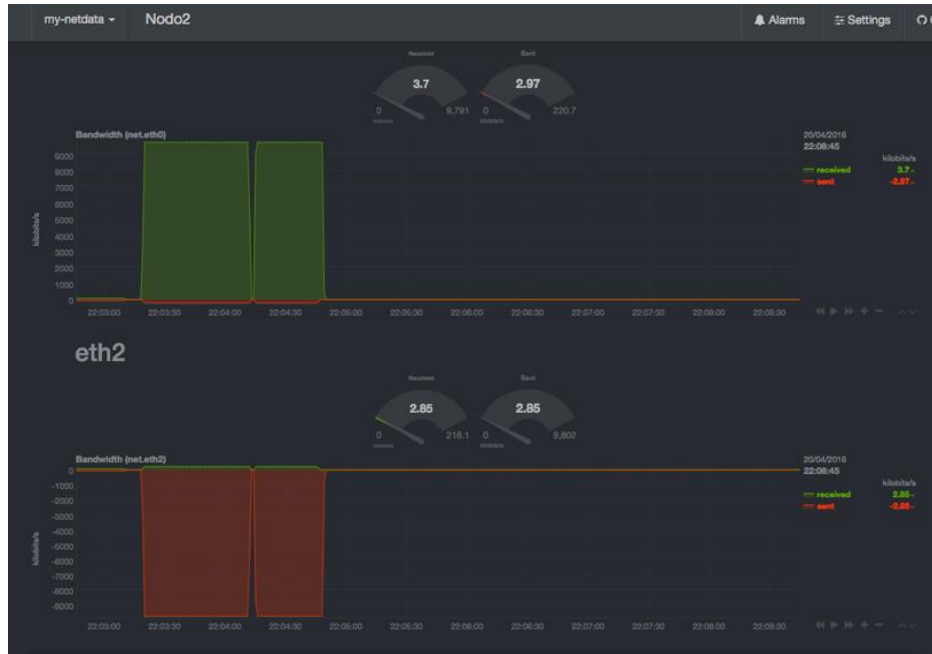


Figure 45 - OLSR RTT 60ms - path 1

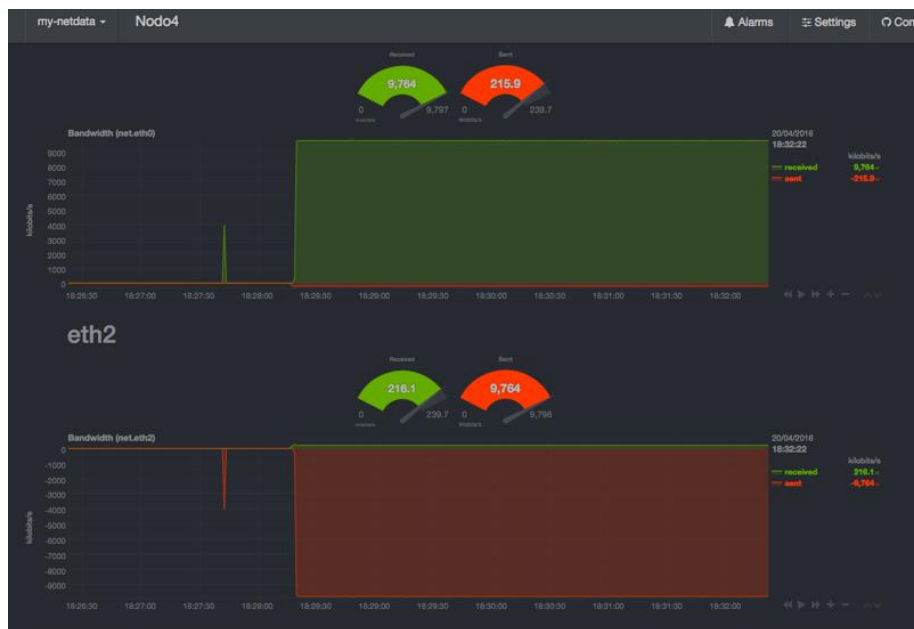


Figure 46 - OLSR RTT 0,5ms - path 2

## BATMAN

In this case, after I turned the second path on, the protocol started to send some traffic to the second one, and after 90s, it started to balance the traffic to both paths, but giving more priority the first one.

Test → **No passed.**



Figure 47 - BATMAN RTT 60ms - path 1



Figure 48 - BATMAN RTT 0,5ms - path 2



## Jitter measure

In this test, we are going to add 10ms of RTT to both paths, and the one of them (Node 2) will add some jitter.

The expected result is that the protocol has to choose the path with less jitter.

Jitter is a high problem for TCP high speed and due to the wireless medium, these networks have high levels of jitter.

### **RTT 10ms – Node 2 Jitter 3.5ms – Node 4 jitter 0.5ms**

#### **Node 2**

Adding RTT 10ms and jitter 3.5ms

```
tc qdisc add dev eth0 root netem delay 5ms 2.5ms distribution
normal
tc qdisc add dev eth2 root netem delay 5ms 2.5ms distribution
normal
```

#### **Node 4**

Adding RTT 10ms

```
tc qdisc add dev eth0 root netem delay 5ms
tc qdisc add dev eth2 root netem delay 5ms
```

## OLSR

It works perfect.

The path change time was 40s.

Test → **Passed.**

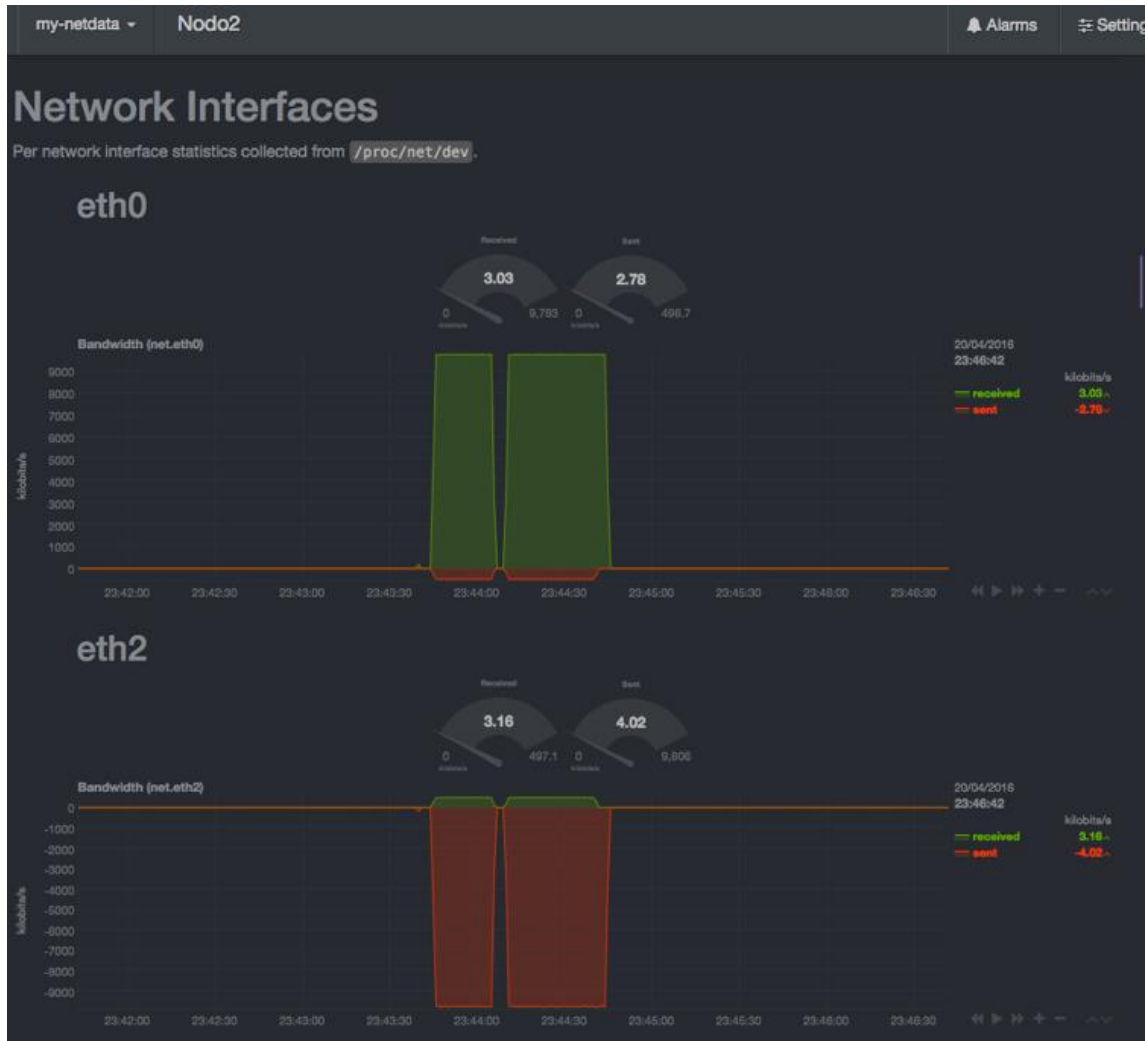


Figure 49 - OLSR RTT 10ms and jitter 3,5ms - path 1

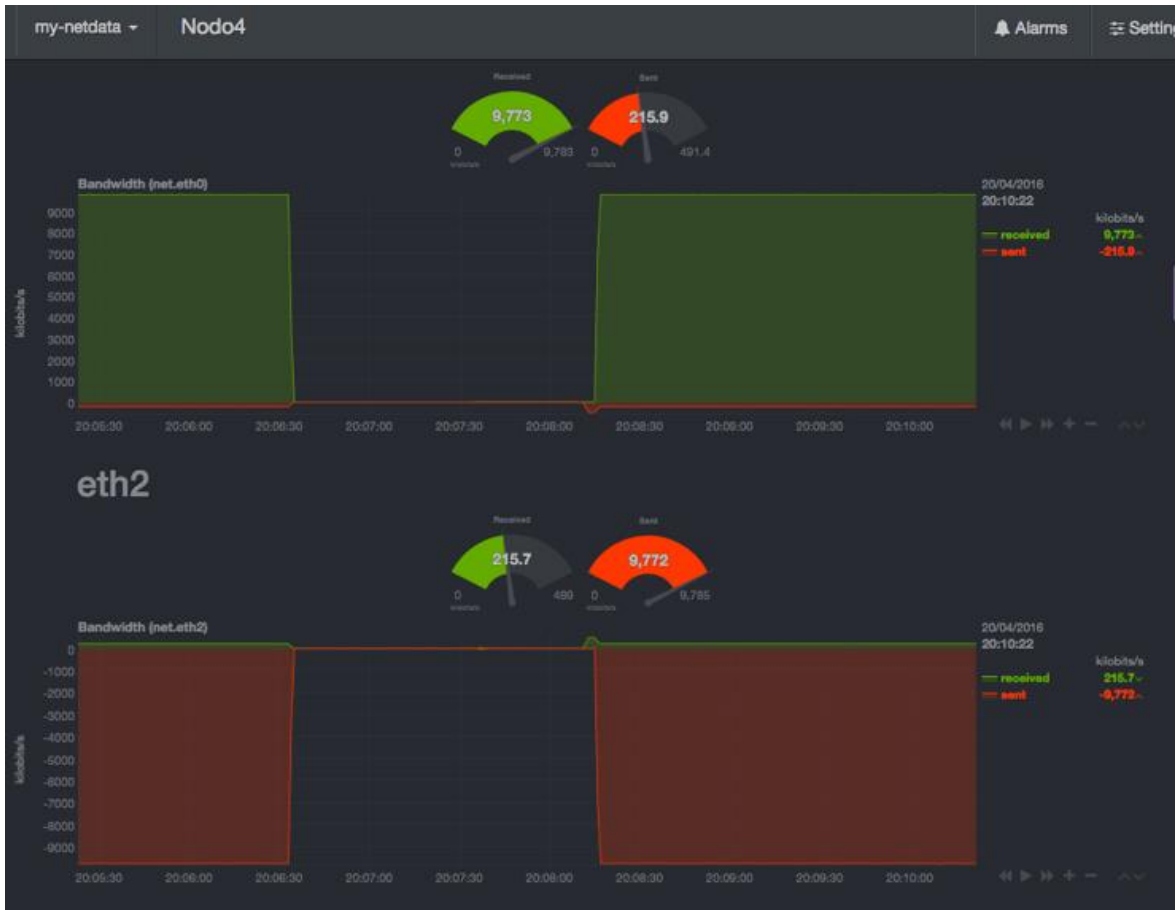


Figure 50 - OLSR RTT 10ms - path 2

# BATMAN

Test → **Not passed.**

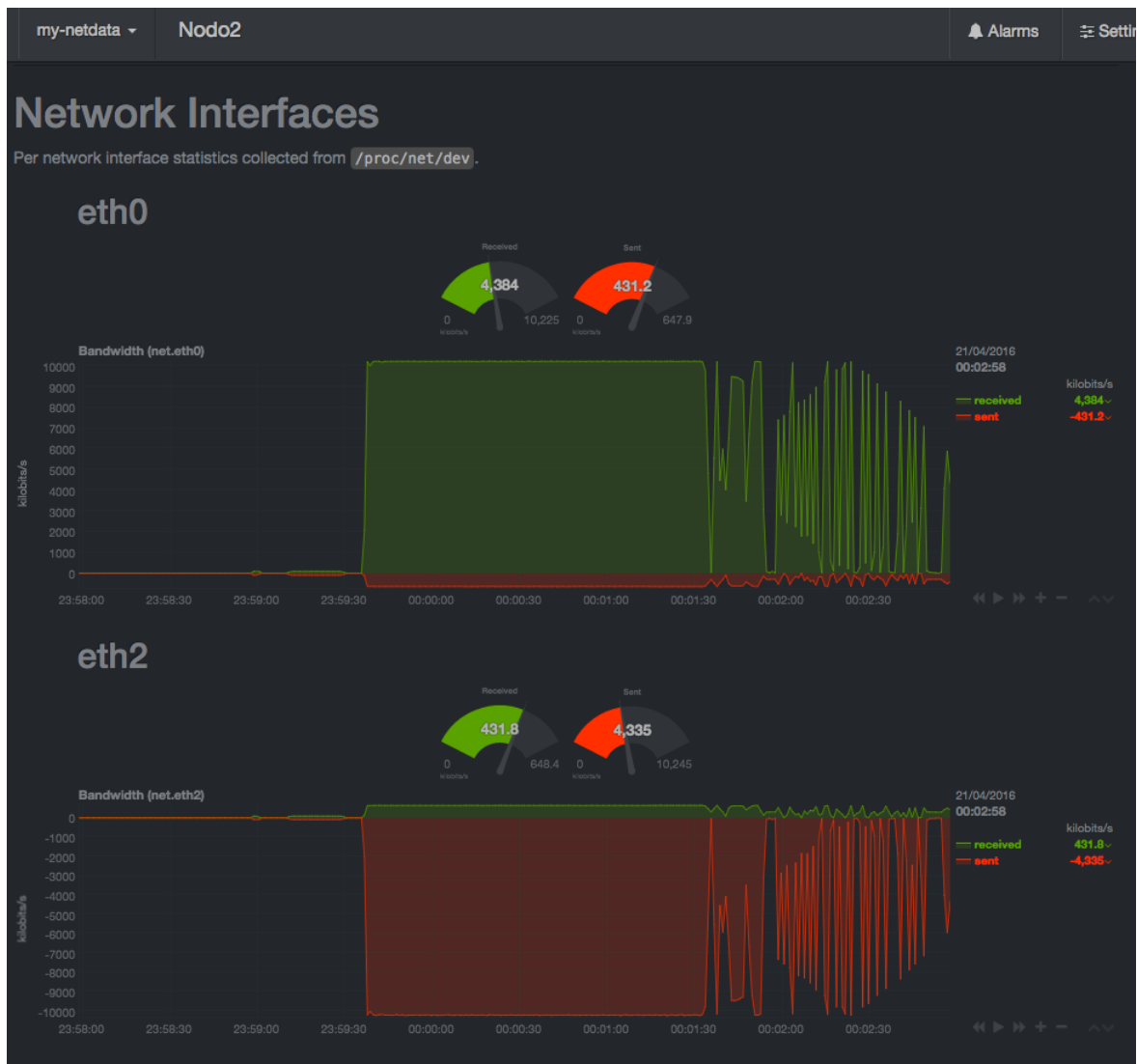


Figure 51 - BATMAN RTT 10ms and jitter 3,5ms - path 1



Figure 52 - BATMAN RTT 10ms - path 2

## RTT 10ms – Node 2 Jitter 6ms – Node 4 jitter 0.5ms

### Node 2

Adding RTT 10ms and jitter 5ms

```
tc qdisc add dev eth0 root netem delay 5ms 5ms distribution normal
tc qdisc add dev eth2 root netem delay 5ms 5ms distribution normal
```

### Node 4

Adding RTT 10ms

```
tc qdisc add dev eth0 root netem delay 5ms
tc qdisc add dev eth2 root netem delay 5ms
```

### OSLR

Test → **Passed**

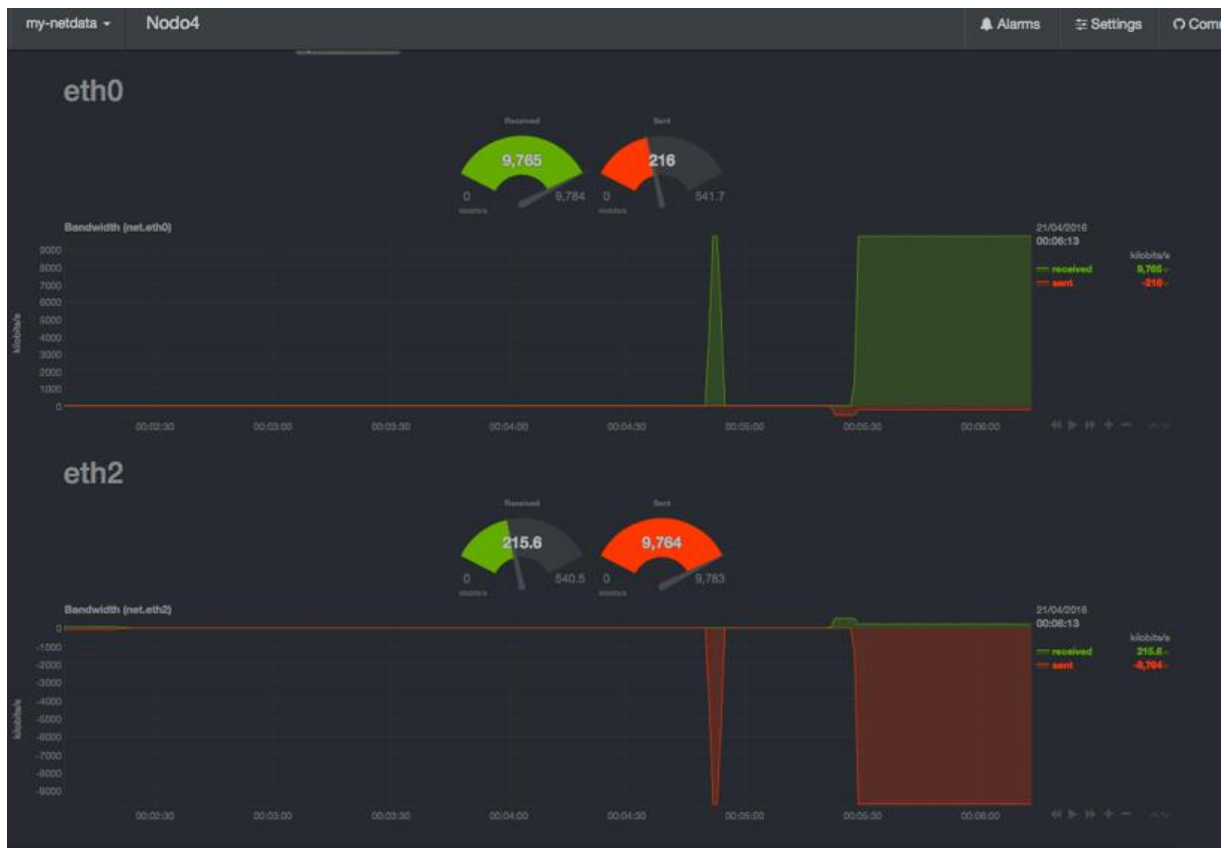


Figure 53 - OLSR RTT 10ms and jitter 5ms - path 1

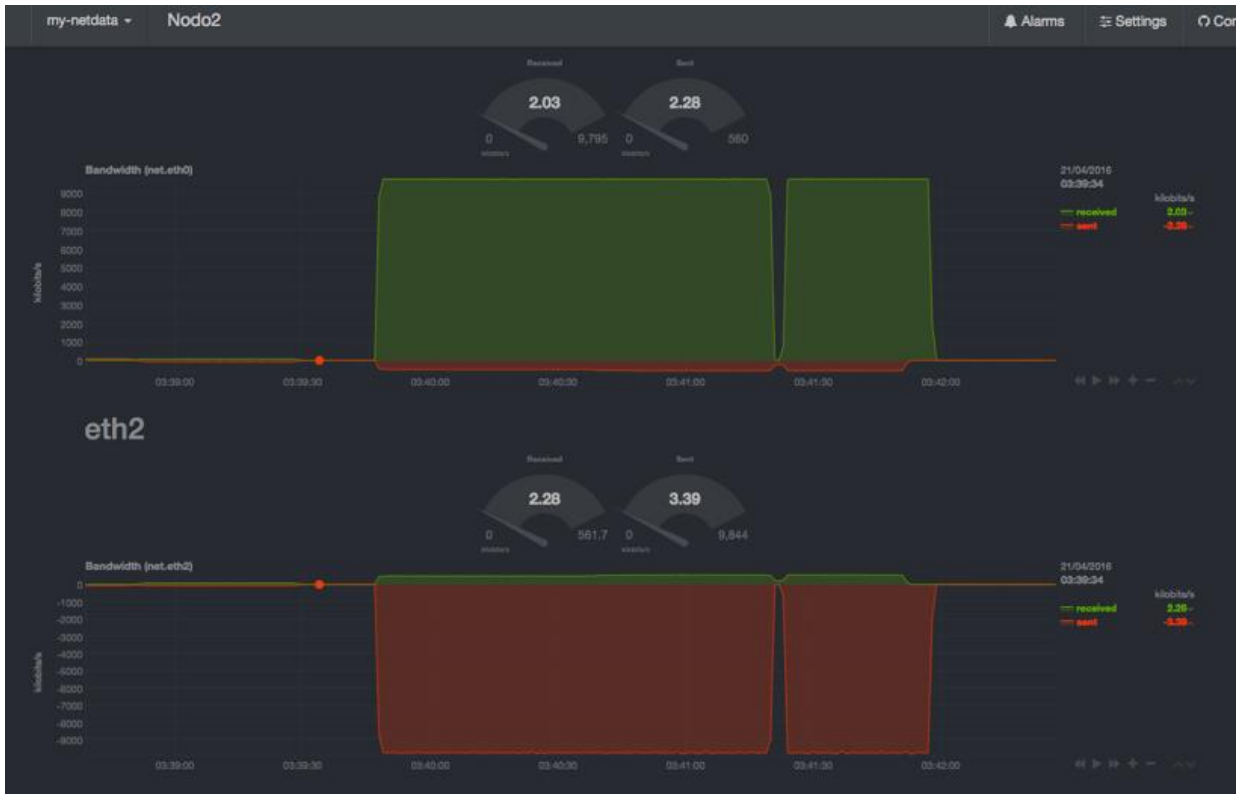


Figure 54 - OLSR RTT 10ms - path 2

## BATMAN

Test → **Not passed**

For an unknown reason (I checked the PER and this was 0%), the Iperf TCP connection suffered a severity performance degradation.

In TCP, the degradations may be caused due to high PER and high jitter. In this case, we have jitter, but no more than the test with OLSR.

In OLSR the TCP worked properly.

BATMAN always tries to balance the traffic (less there is PER). This is not good, because this balancing causes jitter (TCP is oriented to connection, so its desirable that all the packet should be send using the same end to end path).



Figure 55 - BATMAN RTT 10ms and jitter 5ms - path 1





Figure 56 - BATMAN RTT 10ms - path 2

## **GNS3 – MPLS – OSPF – multi BGP simulation**

In this section, we are going to add the whole GN3 configuration.

The general schema

All the Cisco router configuration

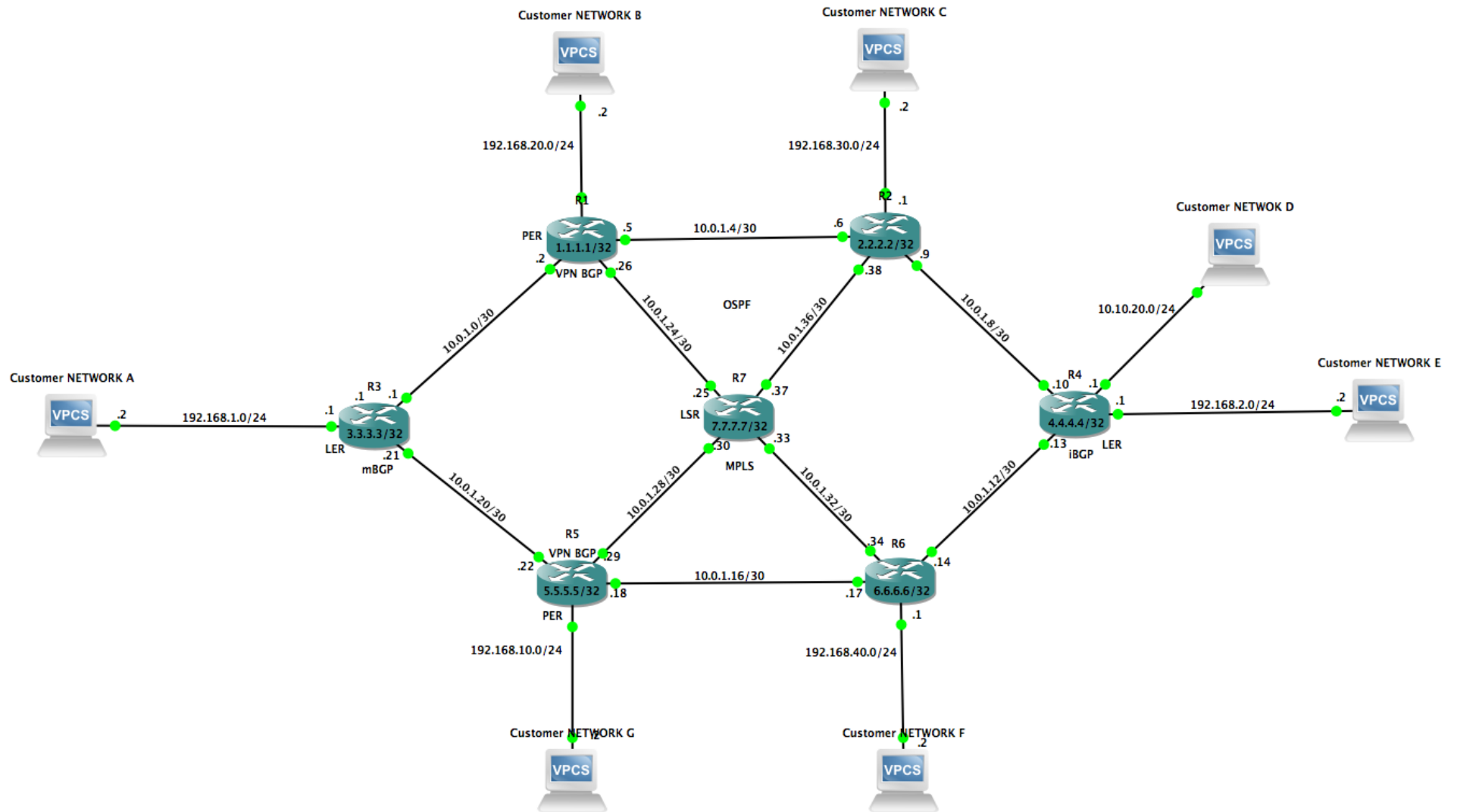


Figure 57 - GNS3 MPLS simulation (ZOOM)

## R1:

```
R1#sh run
Building configuration...

Current configuration : 1667 bytes
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname R1
!
boot-start-marker
boot-end-marker
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
ip vrf VPN1
rd 100:1
route-target export 100:1
route-target import 100:1
!
no ip domain lookup
no ipv6 cef
!
!
multilink bundle-name authenticated
!
ip tcp synwait-time 5
!
interface Loopback0
ip address 1.1.1.1 255.255.255.255
!
interface FastEthernet0/0
ip address 10.0.1.2 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet0/1
ip address 10.0.1.5 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/0
ip address 10.0.1.26 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/1
ip vrf forwarding VPN1
ip address 192.168.20.1 255.255.255.0
speed auto
duplex auto
!
router ospf 1
router-id 1.1.1.1
network 1.1.1.1 0.0.0.0 area 0
network 10.0.1.0 0.0.0.3 area 0
network 10.0.1.4 0.0.0.3 area 0
network 10.0.1.24 0.0.0.3 area 0
!
router bgp 10
bgp log-neighbor-changes
neighbor 5.5.5.5 remote-as 10
neighbor 5.5.5.5 update-source Loopback0
!
address-family vpnv4
neighbor 5.5.5.5 activate
neighbor 5.5.5.5 send-community both
exit-address-family
!
address-family ipv4 vrf VPN1
redistribute connected
exit-address-family
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
control-plane
!
```

```
line con 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line aux 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line vty 0 4
login
!
end
```

## R2:

```
R2#sh run
Building configuration...

Current configuration : 1302 bytes
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname R2
!
boot-start-marker
boot-end-marker
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
no ip domain lookup
no ipv6 cef
!
multilink bundle-name authenticated
!
ip tcp synwait-time 5
!
interface Loopback0
ip address 2.2.2.2 255.255.255.255
!
interface FastEthernet0/0
ip address 10.0.1.6 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet0/1
ip address 10.0.1.9 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/0
ip address 10.0.1.38 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/1
ip address 192.168.30.1 255.255.255.0
speed auto
duplex auto
!
router ospf 1
router-id 2.2.2.2
network 2.2.2.2 0.0.0.0 area 0
network 10.0.1.4 0.0.0.3 area 0
network 10.0.1.8 0.0.0.3 area 0
network 10.0.1.36 0.0.0.3 area 0
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
ip route 192.168.40.0 255.255.255.0 6.6.6.6
!
control-plane
!
```

```
line con 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line aux 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line vty 0 4
login
!
!
end
```

## R3

```
R3#sh run
Building configuration...

Current configuration : 1891 bytes
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname R3
!
boot-start-marker
boot-end-marker
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
ip vrf Internal
rd 100:1
route-target export 100:1
route-target import 100:1
!
no ip domain lookup
no ipv6 cef
!
multiink bundle-name authenticated
!
ip tcp synwait-time 5
!
interface Loopback0
ip address 3.3.3.3 255.255.255.255
!
interface FastEthernet0/0
ip address 192.168.1.1 255.255.255.0
speed auto
duplex auto
!
interface FastEthernet0/1
ip address 10.0.1.1 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/0
ip address 10.0.1.21 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/1
ip vrf forwarding Internal
ip address 10.10.10.1 255.255.255.0
speed auto
duplex auto
!
interface FastEthernet2/0
no ip address
shutdown
speed auto
duplex auto
!
interface FastEthernet2/1
```

```

no ip address
shutdown
speed auto
duplex auto
!
router ospf 1
router-id 3.3.3.3
network 3.3.3.3 0.0.0.0 area 0
network 10.0.1.0 0.0.0.3 area 0
network 10.0.1.20 0.0.0.3 area 0
!
router bgp 10
bgp log-neighbor-changes
neighbor 4.4.4.4 remote-as 10
neighbor 4.4.4.4 update-source Loopback0
!
address-family ipv4
network 192.168.1.0
neighbor 4.4.4.4 activate
exit-address-family
!
address-family vpnv4
neighbor 4.4.4.4 activate
neighbor 4.4.4.4 send-community extended
exit-address-family
!
address-family ipv4 vrf Internal
redistribute connected
exit-address-family
!
ip forward-protocol nd
!
!
no ip http server
no ip http secure-server
!
control-plane
!
line con 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line aux 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line vty 0 4
login
!
end

```

## R4

```

R4#sh run
Building configuration...

Current configuration : 1794 bytes
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname R4
!
boot-start-marker
boot-end-marker
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
ip vrf customerA
rd 100:1
route-target export 100:1
route-target import 100:1
!
no ip domain lookup
no ipv6 cef
!
multilink bundle-name authenticated
!

```

```

ip tcp synwait-time 5
!
interface Loopback0
ip address 4.4.4.4 255.255.255.255
!
interface FastEthernet0/0
ip address 10.0.1.10 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet0/1
ip address 192.168.2.1 255.255.255.0
speed auto
duplex auto
!
interface FastEthernet1/0
ip address 10.0.1.13 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/1
ip vrf forwarding customerA
ip address 10.10.20.1 255.255.255.0
speed auto
duplex auto
!
router ospf 1
router-id 4.4.4.4
network 4.4.4.4 0.0.0.0 area 0
network 10.0.1.8 0.0.0.3 area 0
network 10.0.1.12 0.0.0.3 area 0
!
router bgp 10
bgp log-neighbor-changes
neighbor 3.3.3.3 remote-as 10
neighbor 3.3.3.3 update-source Loopback0
!
address-family ipv4
network 0.0.0.0
network 192.168.2.0
neighbor 3.3.3.3 activate
exit-address-family
!
address-family vpnv4
neighbor 3.3.3.3 activate
neighbor 3.3.3.3 send-community extended
exit-address-family
!
address-family ipv4 vrf customerA
redistribute connected
exit-address-family
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
ip route 0.0.0.0 0.0.0.0 192.168.2.2
!
control-plane
!
line con 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line aux 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line vty 0 4
login
!
!
end

```



## R5:

```
R5#sh run
Building configuration...

Current configuration : 1671 bytes
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname R5
!
boot-start-marker
boot-end-marker
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
ip vrf VPN1
rd 100:1
route-target export 100:1
route-target import 100:1
!
no ip domain lookup
no ipv6 cef
!
!
multilink bundle-name authenticated
!
ip tcp synwait-time 5
!
interface Loopback0
ip address 5.5.5.5 255.255.255.255
!
interface FastEthernet0/0
ip address 10.0.1.18 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet0/1
ip address 10.0.1.22 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/0
ip address 10.0.1.29 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/1
ip vrf forwarding VPN1
ip address 192.168.10.1 255.255.255.0
speed auto
duplex auto
!
router ospf 1
router-id 5.5.5.5
network 5.5.5.5 0.0.0.0 area 0
network 10.0.1.16 0.0.0.3 area 0
network 10.0.1.20 0.0.0.3 area 0
network 10.0.1.28 0.0.0.3 area 0
!
router bgp 10
bgp log-neighbor-changes
neighbor 1.1.1.1 remote-as 10
neighbor 1.1.1.1 update-source Loopback0
!
address-family vpnv4
neighbor 1.1.1.1 activate
neighbor 1.1.1.1 send-community both
exit-address-family
!
```

```

address-family ipv4 vrf VPN1
 redistribute connected
 exit-address-family
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
control-plane
!
line con 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 login
!
end

```

## R6:

```

R6#sh run
Building configuration...

Current configuration : 1306 bytes
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname R6
!
boot-start-marker
boot-end-marker
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
no ip domain lookup
no ipv6 cef
!
multilink bundle-name authenticated
!
ip tcp synwait-time 5
!
interface Loopback0
 ip address 6.6.6.6 255.255.255.255
!
interface FastEthernet0/0
 ip address 10.0.1.14 255.255.255.252
 speed auto
 duplex auto
 mpls ip
!
interface FastEthernet0/1
 ip address 10.0.1.17 255.255.255.252
 speed auto
 duplex auto
 mpls ip
!
interface FastEthernet1/0
 ip address 10.0.1.34 255.255.255.252
 speed auto
 duplex auto
 mpls ip
!
interface FastEthernet1/1
 ip address 192.168.40.1 255.255.255.0
 speed auto
 duplex auto
!
router ospf 1
 router-id 6.6.6.6
 network 6.6.6.6 0.0.0.0 area 0
 network 10.0.1.12 0.0.0.3 area 0
 network 10.0.1.16 0.0.0.3 area 0

```

```

network 10.0.1.32 0.0.0.3 area 0
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
ip route 192.168.30.0 255.255.255.0 2.2.2.2
!
control-plane
!
line con 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line aux 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line vty 0 4
login
!
end

```

## R7:

```

R7#sh run
Building configuration...

Current configuration : 1304 bytes
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
!
hostname R7
!
boot-start-marker
boot-end-marker
!
no aaa new-model
no ip icmp rate-limit unreachable
ip cef
!
no ip domain lookup
no ipv6 cef
!
!
multilink bundle-name authenticated
!
ip tcp synwait-time 5
!
interface Loopback0
ip address 7.7.7.7 255.255.255.255
!
interface FastEthernet0/0
ip address 10.0.1.25 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet0/1
ip address 10.0.1.37 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/0
ip address 10.0.1.30 255.255.255.252
speed auto
duplex auto
mpls ip
!
interface FastEthernet1/1
ip address 10.0.1.33 255.255.255.252
speed auto
duplex auto
mpls ip
!
router ospf 1
router-id 7.7.7.7
network 7.7.7.7 0.0.0.0 area 0
network 10.0.1.24 0.0.0.3 area 0
network 10.0.1.28 0.0.0.3 area 0

```

```

network 10.0.1.32 0.0.0.3 area 0
network 10.0.1.36 0.0.0.3 area 0
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
control-plane
!
line con 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line aux 0
exec-timeout 0 0
privilege level 15
logging synchronous
stopbits 1
line vty 0 4
login
!
end

```

## BIBLIOGRAFY

1. **Wikipedia.** <http://es.wikipedia.org/>. *http://es.wikipedia.org/*. [Online] Junio 24, 2012. [Cited: 07 10, 2012.] <http://es.wikipedia.org/wiki/Guifi.net>.
2. ¿Qué es guifi? <https://guifi.net/>. [Online] 01 28, 2011. [Cited: 07 12, 2012.] [https://guifi.net/es/que\\_es\\_2](https://guifi.net/es/que_es_2).
3. Cómo se vertebrata la red: Troncal, puntos de acceso y conexiones simples. <http://guifi.net>. [Online] 06 29, 2009. [Cited: 07 12, 2012.] <http://guifi.net/es/node/22571>.
4. **Ubiquiti.** Ubiquiti networks. <http://www.ubnt.com>. [Online] [Cited: 07 11, 2012.] [http://dl.ubnt.com/datasheets/airfiber/airFiber\\_DS.pdf](http://dl.ubnt.com/datasheets/airfiber/airFiber_DS.pdf).
5. **Guifi.net.** The Wireless Commons License. <http://guifi.net/>. [Online] 11 25, 2012. [Cited: 7 10, 2012.] [http://guifi.net/WCL\\_EN](http://guifi.net/WCL_EN). About the Wireless Common License.

