Universidad Carlos III de Madrid
www.uc3m.es

# TRABAJO FIN DE GRADO

**Título:** Implementación de un sistema SDN para la movilidad en redes OMNIRAN.

**Autor:** Darío Juan Cerón Bergantiños

**Titulación:** Grado en ingeniería telemática

**Profesor:** Antonio de la Oliva Delgado

**Fecha:** 23/02/2014

# SUMMARY

This document details all the information needed to understand and test distributed mobility management using the SDN paradigm.

This project stars by an analysis of the mobility problem in dense networks. Traditionally mobility has been managed with hierarchical approaches extending the current mobility protocols. But thinking in the future evolution of the network into dense environments some scalability problems appear. The traditional centralized elements may not be able to handle all the traffic in the network and bottlenecks appear at the Mobility Anchors. Nowadays, the problems related to scalability are mostly resolved with hardware upgrades, but in dense environments this couldn't be enough and surely it would be quite expensive. To find a solution to this problem the IETF has chartered the Distributed Mobility Management (DMM) Group.This project focus on implementing a DMM-based mobility solution designed within the EU FP7 CROWD project.

Once the analysis of the problem ended and the requirements of the theoretical solution were defined, we developed all the necessary elements to physically build a distributed network using SDN to manage layer 2 and layer 3.The entities of the network are defined by the CROWD projectin its related publications[9][10]. These districts were run using an SDN implementation called OpenFlow. With all the elements developed we proceed to perform the necessary tests in order to evaluate the distributed mobility management as a solution. This document explains the full design, execution and validationprocesses.

Finally all the measurements and statistical data are detailed in order to have an approximation of the services that could achieve the developed network.

# 1. INDEX

## LIST OF FIGURES

## LIST OF ACRONYMS

**AAN:** Anchor Access Nodes

**ADA:** Asymmetric Double Agents

**AP:** Access Point

**API:** Application Programming Interface

**ATM:** Asynchronous Transfer Mode

**BCE:** Binding Cache Entry

**BGP:** Border Gateway Protocol

**CAM:** Content-Addressable Memory

**CLC:** CROWD Local Controller

**CN:** Correspondent Node

**CoA:** Care of Address

**CRC:** CROWD Regional Controller

**CROWD:**Connectivity management for eneRgy Optimized WirelessDense networks

**DHCP:** Dynamic Host Configuration Protocol

**DMM:** Distributed Mobility Management

eNB: Evolved NodeB

**FIB:** Forwarding Information Base

**HA:** Home Agent

**HoA:** Home Address

**ICMPv6:** Internet Control Message Protocol Version 6

**IETF:** Internet Engineering Task Force

**IPv4:** Internet Protocol Version 4

**IPv6:** Internet Protocol Version 6

**LLC:** Logical Link Control

**LMA:** Local Mobility Anchor

**LMD:** Local Mobility Domain

**LTE:** Long Term Evolution

**MAG:** Mobile Access Gateway

**MIPv6:** Mobile IPv6

**MN:** Mobile Node

**ONF:**Open Networking Foundation

**OSI:** Open System Interconnection

**P2P:** Peer to peer

**PBA:** Proxy Binding Acknowledge

**PBU:** Proxy Binding Update

**PMIPv6:** Proxy Mobility Internet Protocol v6

**QoS:** Quality of Service

**RA:** Router Advertisement

**RIB:** Routing Information Base

**RS:** Router Solicitation

**SDN:** Software Defined Networking

**TCAM:** Ternary Content-Addressable Memory

**VAN:** Visited Access Nodes

**Wi-Fi:** Wireless Fidelity

# 2. INTRODUCTION

The main goal of this project is to implement an SDN-based DMM (Distributed Mobility Management [8]) approach within the context of the CROWD EU FP7 project[10]. This document explores different solutions for the issues resulting of mobility in dense network environments. The main idea of the implemented solution is to take an approach to flatten the network, swapping the hierarchical approaches used currently in production networks (such as MIP or PMIP) and distribute the functionality of its centralized entities.

This distributed mobility management will be possible through a Software Defined Networking (SDN) architecture. To have a global understanding of how works a SDN network, the analysis has been separated into two points: The first one is the SDN paradigm, based on the separation of control and data planes. The second one is the practical application of that theory in the OpenFlow protocol. Once the possibilities given by OpenFlow are studied, we will proceed to define the requisites and functionalities for a DMM network in order to study various OpenFlow controllers and choose one fitting those requisites. Following the CROWD designwe will define new concepts such as the Local Controller, the Regional Controller and the network districts for our implementation. The last step is to implement the OpenFlow network on affordable devices. First of all we will develop the necessary software to run over the OpenFlow controller. This software will manage all the connections and traffic through this network (Local Controllers and Regional Controllers). After thiswe will design a small testbed implementing the elements mentioned before. It will include allthe necessary entities of a CROWD wireless networksuch as Access Points, DMM Gateways, CROWD Regional Controllers and CROWD Local Controllers. In this environment a Mobile Node will perform different mobility handovers focused on assessing the viability of the DMM solution.

Finally the full system's performance will be analyzed to validate it as a solution for mobility in dense environments. This analysis will be composedof statistical measures, cost calculations and infrastructure analysis.

# 3. MOTIVATIONS

As the Internet communications are growing exponentially to provide connection to the also growing number of Internet devices, the network has been forced to evolve. This has been more evident with the appearance of smartphones, tablets and other devices needing a wireless access to the network. This raising number of elements has started to generate questions about how the actual network protocols will handle this massive increase of hosts and nodes in the network. These speculations have introduced the concept of "Dense networks".

## 3.1. DENSE NETWORKS

Dense networks[8] are the result of the necessity of providing Access to a growing number of users (network hosts), which is translated into more traffic and wider physical areas to cover and a necessity of faster connections.

Dense networks definition:

The concept of dense network used in this project is associated to the future massive-grown networks of the future. So from now, dense network means networks that will be bigger than the current ones in a scale of one hundred. That means one hundred more nodes, one hundred more access points, one hundred more hosts and one hundred more traffic.

As dense networks have high numbers of connections, routing through them requires more complexity on routing algorithms and since the connection is wireless some extra issues appear.When the number of network nodes increases, it causes more complexity in routing through them. As an example: within an scenario of an IPv6 wireless network managed with PMIPv6, the local mobility anchor has the function of managing all the new attachments and also,handle all the handovers performed in the network. With the huge number of access points and mobile nodes of a dense network, all thatmanagement process can become an excessive overload to be handled by only one device.

When wireless connection appeared, a few protocols were standardized to provide the wireless access to the networks.Those protocols are based on an anchor point, which provides the mobile device with global reachability to the network.  This allows the network to forward packets between the MN and its peers even if the MN has moved from its home domain to another one. These protocols need extensions to improve handovers in terms of delay. The extensions usually involve difficulties on deployment due to the extra-complexity required by the new protocols. All of that has leaded to suboptimal choices.In those centralized modes of operation, various points of failure have been detected such as security issues (with a centralized anchor there is one point where to focus the attacks), routing issues due to the network traffic being redirected through the anchor (which usually is not the shortest path). And finally the main one:these centralized networks are not scalable at all. The problems of handling the traffic of several million users associated to single device managing all their connections will become impossible to solvewith current approaches.All these issues get critical when the networks turn into dense network.

In order to face this traffic explosion, the Internet Engineering Task Force IETF[1] has started its researching of standardizable solutions. One of the groups investigating is the [DMM[2]](Distributed Mobility Management) Group, which has beenstudying the mobility problems in centralized architectures since 2012. Their work areaembraces the idea of defining DMM schemes compatible with the current IP mobility protocols that can be deployed causing a minimum impact. Its definition of a DMM environment is a scenario in which the network-access, routing and mobility management tasks are distributed between various elements substituting the current centralized way.

One of the keys of the DMM concept is to distribute these functions as closer to the Mobile Nodes as possible. There are technologies that already provide mechanisms for distributing the binding information such as Peer-to-Peer (P2P). The DMM Group is studying generic solutions for mobility management based on the MIPv6[4] and PMIPv6[6] concepts. One of the approaches[24] is to split the Anchor points of these protocols into Anchor Access nodes (AAN) and Visited Access Nodes (VAN). The AAN maintain the mobility state of the node. The VANs tunnel the traffic between them when the node is roaming. This structure distributes the Anchor functions between various elements. Another solution is the Asymmetric Double Agents (ADA[23]), an extension of MIPv6 that transforms the Home Agent (HA) into a Local Mobile Proxy (LMP) and a Correspondent Mobile Proxy (CMP), the first one takes the most part of an HA and the second provides route optimization[3].

This solutionhas been developed as an instance of an SDN-driven approach to the network design. Despite SDN already having some implementations such as OpenFlow, it still hasn't been introduced in production networks. However it has been proven to work with mobility protocols as WiMAX(Wi-Fi and Worldwide Interoperability for Microwave Access[22]) was implemented using OpenFlow, and demonstrated its compatibility with handovers.

Following all these efforts, in this project we have decided to develop the CROWD approach for Distributed Mobility Management using SDN.

[1]http://www.ietf.org/February 21 2014
[2]http://datatracker.ietf.org/wg/dmm/charter/February 21 2014
[3] http://www.ict-crowd.eu/ February 21 2014

## 3.2. SDN

SDN[1][3] is an acronym for Software Defined Networking. It is an approach to computer networking that aims at redefining the networkwith data and control plane separation instead of the current architectures.It appeared when the needs of the networks became more and more expensive. The SDN proposers started researching when they noticed that network device vendors were not meeting their needs, particularly regarding the flexibility in the feature development and innovation spaces. Routing devices were getting overpriced due to their complex control plane components. At the same time, they saw that a much more powerful computer was proportionally cheaper. So they decided to create a network with a centralized control plane. That translatedin only one expensive device and much cheaper switches. That resulted in SDN.

This architectural approach optimizes and simplifies network operations by more closely binding the interaction among applications and network services and devices.The main tenet is to separate control and data plane so the second one can be centralized in one device called the "Controller". This element takes all the layer 2 control functionalities so the switches of the network can act just like data forwarding elements. This separation is a concept, so it can be interpreted at will. This wide spectrum covers from the traditional fully distributed control plane to a strictly centralized one.

A fully distributed architecture means one complete instance of control plane and at least one data plane instance per device. It is an architecture approach with high resistance to failures because every element is mostly independent from the others. As a conit has convergence failures due to the fact that there is one control instance to configure per device. As a result, this architecture is non-profitable for horizontal scaling. The current network architectures are defined in a fully distributed way as they were built based on the notion of Autonomous Systems.

Semi-centralized are actually the most used architecture on SDN networks. It consists on one centralized control entity, and an instance of that control plane on each node. The instances and the centralize entity has communication and together conform the control plane of the network apart from the data plane which is fully distributed. This is the architecture with more ease for horizontal scaling, as it only needs to deploy a device with a control plane instance. This is the architecture used by the expanded OpenFlow protocol.

The strictly centralized architecture is mostly experimental. As it is the canonical approach of SDN it has the full control plane centralized on one device. The switches are dumb switching devices remotely controlled by the centralized control plane. That implicates only one point of failure, which makes the architectureprone to errors in the centralize entity.

To have a more accurate understanding of the implications derivate of the separation of the control and data plane, first, the control and data planesfunctions must be defined: Traditionally, routing and forwarding process is performed using two cache tables, the RIB and the FIB. They are not associated to any protocol, they are simply necessary for the routing and forwarding process in traditional devices. The Routing Information Base (RIB) is the table where the IP routing information is stored. When a routing protocol is running on a device, it uses message exchanges to learn new routes and then it adds them to the RIB. It also contains directly connected networks and static routes. The device uses this protocol to guess the output interface for the traffic received. The other table is the Forwarding Information Base (FIB). It is used to make IP switching based on network prefixes. It contains the next hop for each reachable destination network prefix. The process of filling the FIB and distribute the forwarding information can take minutes while the device figures about which is the best path for each prefix. The control plane can be associated to the routing process; this part is the one in charge of deciding the path through a network and is carried out by all the devices forming the network. The data plane can be associated to the forwarding process. Its function is to put packets from an input interface to an output interface; each device is in charge of its own forwarding process.

The concept of control and data planes separation is not new. In fact, most modern network devices have different dedicated resources for their control plane and for their data plane. The innovation here is to remove all (or all the possible) control plane from the switching devices and to group it at one only element call the controller. The main advantage of this is that the controller has full knowledge of the network topology, which simplifies all the routing process and removes overload in the network due to state exchange messages. During large modifications on the network, the controller just interacts with the control entity of the implied devices.But the applications are not limited to this. We can analyze all the areas affected on an SDN network.

**Scalability:**The service card generation's evolution is slow if we compare it with the evolution taken in each processor's generation. So usually the limitation on subscribers, flows or services states on a device is imposed by the service card specifications. Usually the ways of solving these limitations takes considerable time and high costs. In SDN networks these specifications are taken to only one device for the network, so they are much more suitable to adapt and scalable cheaper.

**Evolution:**Usually the network operators had to follow continuous hardware upgrades on its devices to follow the normal evolution of the needs in growing networks. The problem comes when the needs on the control plane doesn't grow in parallel with the needs of the data plane. When both planes are shared in the same device this upgrade must be in both planes even if only one of them needs it. In SDN networks, upgrades on the data plane and control plane can be done separately. In dense networks we have an example of a separate upgrade; while the data plane remains the same, the control plane needs an evolution in order to serve the increasing number of hosts and traffic.

**Complexity:** The modifications in the topology of a network affect only to the control plane, which is in charge of knowing all the elements of the network. When there is one control plane entity in each device the result is a lot of network-state message exchanging. This produces an undesired overload in the network due to control messages. When the control plane is centralized there is only one device that needs to know the network topology and there is no need of having a network-state protocol always running.

As an oppositeofthese advantages, SDN networks obviously have weakness when they are compared to distributed control plane architectures. The issues derived of having only one network control entity come as that entity becoming a single point of failure. The first point to know is that the controller will support a control session with each managed device. As the scale of the network increases the traffic destined to that point can become excessive to handle, so the device chosen as controller must have strong specifications. The second point is that a failure on the controller means a failure on the entire network. The third point is the distance between the controller and its managed elements. As the controller must be directly connected with any device under its management, geography can be a problem if some link is longer than desired and generates delays in the communication with the controller.

Considering this, the SDN proponents argue that SDN is an alternative to the increasing costs of the devices derived from the evolution of the network. The reality is that a fully centralized control plane hasn't been proven yet as a true possibility. Instead, a partial centralization of the network has demonstrated to be as efficient as distributed networks. Some protocols like OpenFlow actually work with partial centralization of the network and havealready been successfully deployed.

# 4. STATE OF THE ART

This section describes the current state of mobility and SDN. It starts with a description of MIPv6 (Mobile IPv6) as the first mobility oriented protocol developed, and PMIPv6 (Proxy Mobile IPv6) as the MIPv6 evolution that is being used actually. As these protocols have shown some issues when applying them to the future networks, the DMM group has been researching and proposing solutions to replace them. One of those solutions is the development of distributed mobility networks using SDN. One of the existing protocols developed under the umbrella of SDN technologies is OpenFlow, which is referred at the end.

## 4.1. MIPv6

Mobile IP (MIPv6 - RFC 6275[4][5]) provides the mobile nodes with global reachability and session continuitywhile moving among IPv6 networks. That is done by introducing the Home Agent (HA), an entity which anchors the permanent IP address used by the MN. That address is called Home Address (HoA).When the MN moves to another network and changes its IP, that IP is stored in the HA's Binding Cache Entry (BCE) as the MN's Care-off Address (CoA). The HA now can tunnel to the CoA all the traffic destined to the HoA. From the look of the Correspondent Node (CN) which is the MN's communication peer, the MN is alwaysreachable on its HoA and remains outside of all the mobility process.

The operation of MIPv6 starts when the mobile node attaches to the MIP network (the home network). Then the Home Address is assigned to it based on a sub-network prefix on its home link. When the node moves to a foreign network, it receives its care-of address (CoA). This address is assigned in the foreign network via standard processes such as Stateless Auto-configuration or DHCP. While the node is in the foreign network, the packets destined to it (to its HoA) will be forwarded to the foreign network and addressed to the CoA. This association between HA and CoA is called binding. The traffic re-direction is explained in the [Figure 1: MIP Mobility]. First of all (Purple arrows) the traffic destined to the MN's HoA is sent to the Home Agent. Then the Home Agent creates an IP over IP tunnel between him and the Foreign Agent (Green arrows). Then the Home Agent forwards the traffic destined to the MN through the tunnel. Finally (Blue arrow) the Foreign Agent forwards the traffic to the MN where is received at the CoA.

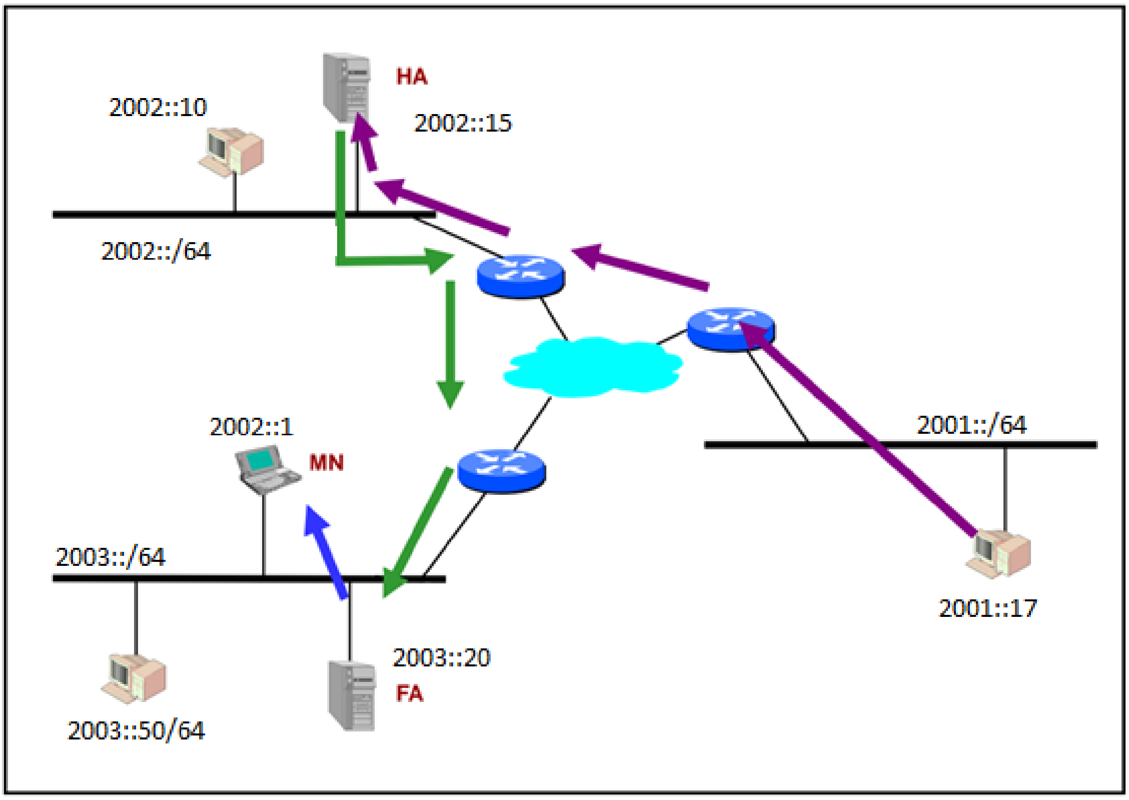


Figure 1: MIPv6 Mobility[4]

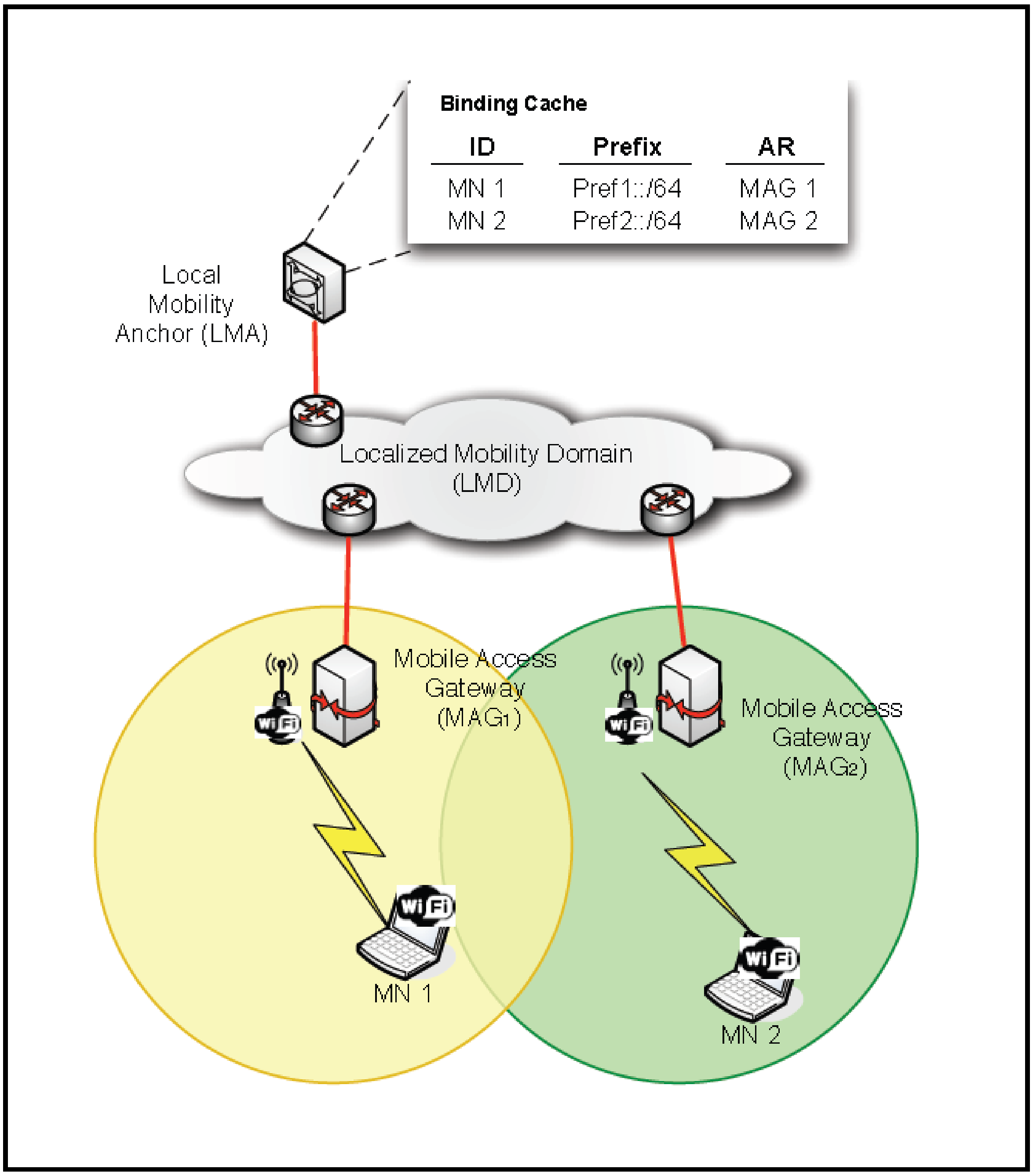
## 4.2. PMIPv6

Proxy Mobile IPv6 (PMIPv6 - RFC 5213[6][7]) is a mobility protocol based on MIPv6.It's an extension of IPv6 whose main feature is that the mobility is now network-based. A PMIPv6 Local Mobility Domain (LMD) like the one shown in [Figure 2:PMIPv6 scenario]. The support of host mobility is possible without the mobile node involvement by the use of signaling messages between the network nodes and the anchor point, called Local Mobility Anchor (LMA).The nodes in charge of performing all mobility related signaling in behalf of the MN are called Mobile Access Gateway (MAG) and are collocated with the default gateway of the node. The signaling is performed between the MAG and the LMA. They establish a tunnel between them through which the traffic is redirected to the MN so the MN can be always reachable.

The operation of PMIPv6 is the following: When a MN attaches at first to one of the LMD's AP; it sends a Router Solicitation (RS) message requesting an IPv6 prefix to the MAG (Proxy Binding Update (PBU) message). The MAG forwards the message to the LMA which answers with a network prefix (Proxy Binding Acknowledge (PBA) message) and stores the MN's mapping on its Binding Cache (BC). The MAG sends the prefix to the MN via a Router Advertisement (RA) message. Finally the LMA forwards all the traffic between the MN and the network to an IPv6 over IPv6 tunnel between the MAG and itself. When the MN moves to an AP under the coverage of another MAG and sends the RS, the LMA will find the MN on its BC and will answer

[4] www.uc3m.es February 21 2014

with the already stored prefix and change the traffic forwarding to another tunnel between the new MAG and itself. This process means no IPv6 changes for the MN in the entire LMD.



Binding Cache
ID
Prefix
AR
MN 1
Pref1::/64
MAG 1
MN 2
Pref2::/64
MAG 2
Local Mobility Anchor (LMA)
Localized Mobility Domain (LMD)
Mobile Access Gateway (MAG1)
Mobile Access Gateway (MAG2)
MN 1
MN 2

Figure 2:PMIPv6 scenario[5]

[5] www.uc3m.es February 21 2014

## 4.3. DMM

As the current networks are deployed in a hierarchical centralized manner, all the data traffic passes through the Mobility Anchor (HA or LMA), which presents security performance, and scalability issues like bottlenecks and a centralized point of failure. Also, the current networks are always activated. That is a non-optimal solution due to the fact that the mobile node is not performing handovers all the time during the connection.

To solve these problems, the DMM working group has been working in alternatives to MIPv6 and PMIPv6. A feature always present in the DMM schemes is the idea of distributing the mobility anchoring at the AR level. When the MN performs mobility, its session at the old anchor is terminated and the data traffic is routed optimally without tunneling. The approaches to the distributed mobility management function can be divided in four categories:

1- Routing-based approaches: The main idea is to use routing updates between routers to manage the mobility within the mobility domain.The process is to detect the node's assigned prefix by the use of DNS lookups and then the exchange of Border Gateway Protocol (BGP) update messages to actualize the routing information. This proposal still lacks of deepness performance calculations.

2- Architecture-dependent solutions: The idea of this proposal is to find solutions focused on providing enhanced mechanisms for local breakout and offloading to improve mobility reducing the traffic through the operator core network. These proposals detect non-optimal paths.

3- Peer-to-peer approaches: The concept is to distribute mobility management functionality in various entities and to do it as much closer to the mobile node as possible.

4- Solutions based on extending existing IETF protocols: There are solutions designed based on the mobility behavior of MIPv6 and PMIPv6. The ones adapting MIPv6encompass the use of Anchor Access Nodes (AAN) and Visited Access Nodes (VAN). The first keep global reachability while the Mobile Node is roaming from the first to the second. This has been extended to support prefix relocation so the MN can maintain its prefix when it is moving. That makes mobility absolutely transparent to the MN. There are another solutions based on PMIPv6 that includes a new element between the LMA and the MN. This element adopts certain LMA functions such as route optimization and establishing tunnels between MAGs.

The proposed SDN-based solution falls within the fourth category since it is a way of implementing a PMIP-based DMM approach. The background of SDN is to separate control and data planes and manage them separately without changing any hardware. As it is been explained at the SDN (section 3.2 - Complexity), one of the advantages of a centralized control plane is that it can be adapted much easier to network changes. Studying the concept of dense networks, it can be seen that it mainly impacts on the control plane as the problems come only on routing and traffic management and not on forwarding at all. In an SDN environment, the dense network problem can be removed from the forwarding devices and isolated in a unique element: the controller. In that scenario a solution could be developed with a minimum impact on the previously developed elements. That means minimum costs and turns any possible solution in a suitable one. Another advantage of SDN is the flexibility of software. Software can be adapted much faster and cheaper than hardware. The network controller while keeping its managing functions can be easily adapted to be any customized network manager device. With these points in mind, the DMM Group is working to develop a DMM network over the concepts developed by the CROWD Group using SDN to manage the traffic. The SDN protocol chosen to perform in that scenario is OpenFlow.

## 4.4. OPENFLOW

OpenFlow[12] protocol was designed by the Stanford University. It came from the idea of creating a network for experimental researching, and that ended in an array of protocols whose functionality could replace link and network layers (OSI layers 2 and 3) in commercial switches and were grouped under the name of OpenFlow [Figure 3: OSI stack Vs. OpenFlow stack].The goal was to create a protocol for devices containing only the data plane to respond to commands received from a centralized controller that gathered all control plane. Later in 2011 the Open Networking Foundation (ONF) was founded with the goal of standardize, commercialize and promote OpenFlow[2].

| Traditional OSI stack | OpenFlow Stack |
|---|---|
| Application | Application |
| Presentation | Presentation |
| Session | Sesion |
| Transport | Transport |
| Network | OpenFlow |
| Link | |
| Phisical | Phisical |

Figure 3: OSI stack Vs. OpenFlow stack

OpenFlow, as an SDN protocol,separates data and control planes. While the first remains distributed as it is done traditionally, the control plane gets centralized. That introduces a new element called the "Controller". The controller is the brain core of the network, so it relies all the layer 2 and layer 3 of the network on itself. The controller and the switches communicate using a wire protocol so the controller can take all the routing and forwarding decisions of each switch of the network[19]. So summarizing: OpenFlow consists in a Controller managing the network switches behavior, the switches with their flow tables and its control interface, and a communication protocol between the controller and the switches.

### 4.4.1. The OpenFlow protocol

The protocol is a bidirectional language between the Network Controller and the switches.The OpenFlow protocol works by installing matches and actions on forwarding tables. When a message is received at a switch and a match to handle it is not installed, the messages are destined to the Controllerby encapsulating them into an OpenFlow message and sending them to the Controller via its control interface. In the other direction (From the Controller to one or various switches) the messages contains orders to add to the Flow Tables. These orders can be divided in CLASSIFIER and ACTION. The classifier is the matching that the switch will perform on its incoming packets. The different types of matching depend on the OpenFlow version, on 1.0 the match can be done on MAC address, IP address, TCP/UDP/SCTP port, VLAN id, input/output port, any combination of these or a wildcard. The action applied to the matching can be to forward, modify, enqueue or drop. One of the features of OpenFlow is the allowing of pipeline forwarding, which consists on have an associated action in the Flow Table consisting on forward the packet to another Flow Table and continue the matching in that table.

### 4.4.2. The switches

The OpenFlow switches are built on the idea of re-using the flow-tables of a commercial switch. These switches used to have this flow tables from TCAMs,and OpenFlow protocol turns these tables into OpenFlow flow tables. TCAMs are a kind of Content-addressable memory that allows third state matching. That means a "don't care value" or "X" in the comparison. For example, a 1X1 matching can be 101 or 111. This is very useful in routing to route using wildcards. An example of this in OpenFlow is a match of a header with a determinate input MAC and any value (a wildcard) of output MAC. OpenFlow takes those TCAMs and turn them into Flow Tables. The datapath of an OpenFlow switch is a Flow Table and an action associated with each flow entry.

A basic OpenFlow switch is a dumb datapath whose functionality is just to forward input packets at an input port to an output port basing its decisions on orders provided by a remote process (the controller). It consists of at least these three elements:

1. The Flow Table
   The Flow Tables are a modification from the original use of the switch’s tables. The entries now are conformed by a match header entry, and an action associated. When an incoming data flow matches with an entry of the table [Figure 4: OpenFlow message matching values], the switch performs the action associated to it. This forwarding process equals to the traditional layer 1 of the OSI stack.

| Ingress Port | Ethernet | | | VLAN | | IP | | | | TCP/UDP | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | SA | DA | Type | ID | Priority | SA | DA | Proto | TOS | src | dst |

Figure 4: OpenFlow message matching values

2. The secure channel
   This channel is physically located at one of the switch’s interface, which is designed as “Controller port”. That interface is linked to the network controller (who has functionality of layers 2 and 3) and allows a bidirectional communication between the two elements. This communication is made using the OpenFlow Protocol, which is the third basic element of an OpenFlow switch.

3. The OpenFlow Protocol
   This protocol defines the structures of the messages exchanged between the OpenFlow switches and the controller. These messages are for establishing control session, exchange the orders of Flow Table modifications and collecting statistics. In versions higher than 1.0 pipeline processing is supported.

Despite of a switch can be used partially as an OpenFlow switch, in this project switches are used as fully dedicated OpenFlow switches. Their functionality is the following:

When there is an incoming data flow the switch takes the first packet and catchesits header to perform matching with it on its first Flow Table. If there is no match(no forwarding instructions settled) the switch sends that first packetto the controller via its control port (the secure channel) so the controller can manage routing and forwarding for that flow. That packet will be forwarded encapsulated as another OpenFlow message. A basic OpenFlow switch only forwards the first packet of any incoming flow to the controller, but this can be settled to forward all the unmatched packets what results in the forwarding of the full flow to the controller. In the case that the packet matches with any entry of the flow table, the switch will perform the action associated to that entry. Those actions can be forward, modify, enqueue, drop or forward to another Flow Table [Figure 5: OpenFlow Switch Flow operation].

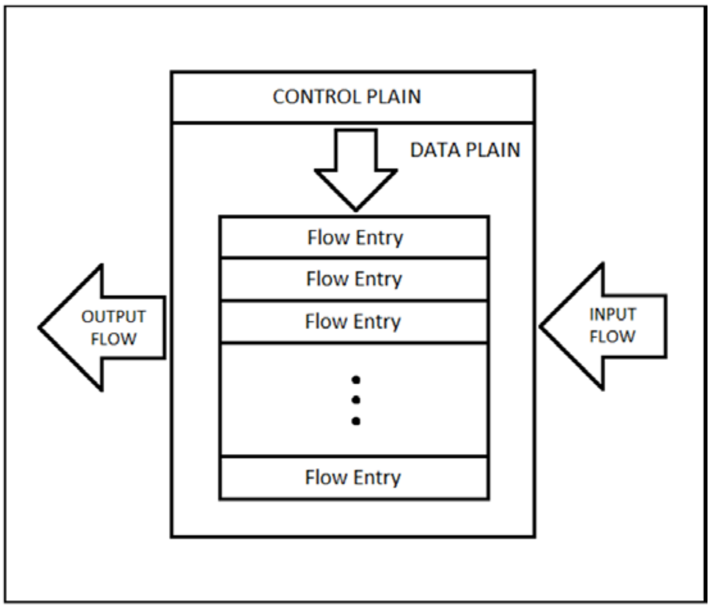


**Figure 5: OpenFlow Switch Flow operation**

For this operation, the Flow Table must be populated with information. That happens when the OpenFlow switch receives a message at the controller port.That message contains one or more modifications for the Flow Tables that will be added by the switch. Then when here is an incoming data flow that matches to one of the flow entries of the entry table, the switch forwards the flow on the output port settled on the flow table.

### 4.4.3. Controller

The Controller is the device that centralizes the entire control plane as SDN defines. It is a device that communicates with each switch of the network to manage all the traffic through it. That is done by designing one of the physical ports of the switch as a controller port where this communication happens. The full process of packet routing is performed in the controller. Once the path through the network is decided, the controller communicates with the implicated switches to configure them in order to forward the traffic through the network [Figure 6: OpenFlow routing and forwarding].

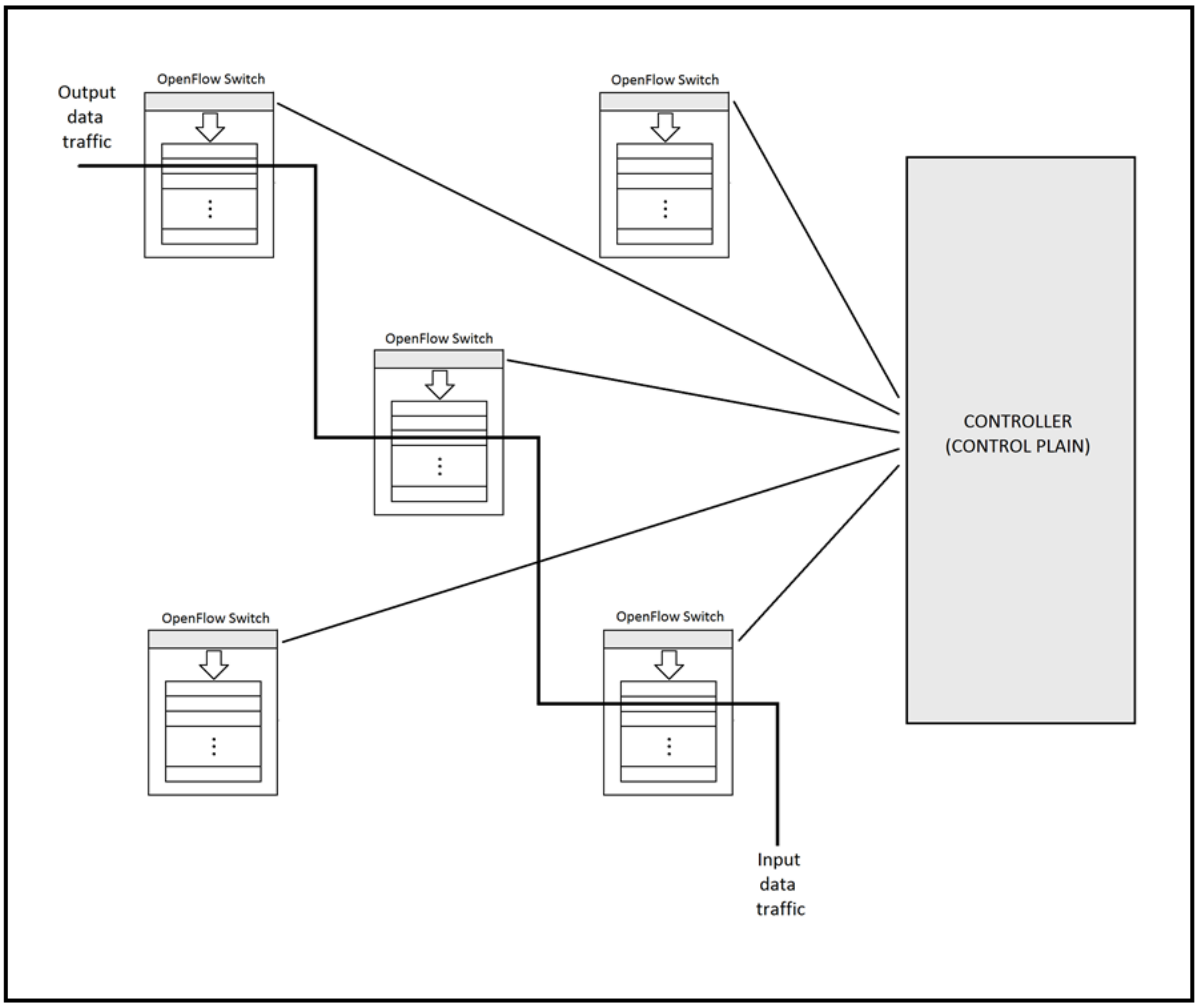


Figure 6: OpenFlow routing and forwarding

# 5.EXPERIMENTAL DEVELOPMENTS

The Experimental development of the project include the necessary software development and the physical network implementation where test the software.

## 5.1. SOFTWARE

As OpenFlow is an open standard, there are various open source controllers, each one developed with its own goals and characteristics. In the following brief an analysis of the most relevant controllers is presented.

**NOX:** Nox is a C++ controller developed by Nicira Networks specifically as a platform for building network control applications. It was created side by side with the OpenFlow foundation so it provides full OpenFlow support. It provides a well-defined OpenFlow 1.0 API. This API includes host tracking, routing and topology-discover, learning switches, network wide switches and fast asynchronous IO. Some popular applications developed over NOX are SANE and Ethane, with proven SDN efficiency. The first one is an approach to representing the network as a filesystem. The second one is a research application for centralized network security at the level of an access control list. With both of them, researches have demonstrated MPLS-like applications over a NOX core.

**POX:** It is a Python (Python 2.7) version of Nicira´s NOX controller. Despite its graphic interface looks the same as NOX, it claims some advantages over it: It is specially targeted for Linux, Mac OS and Windows. It also brings an install-free PyPy runtime for easy deployment. PyPy applications perform better over POX than over NOX.

**TREMA:** It is a Ruby/C framework for OpenFlow developed by NEC with later contributions of GPLv2. It provides a unique interface to handle all OpenFlow messages, something important in the pursuit of simplicity. It also provides a network-simulation framework, very useful to test developed controllers. A controller running over Trema can interoperate with any OpenFlow element without needing a specific agent.

**FLOODLIGHT:** It is a Java controller created by Big Switch Networks based on Stanford's Beacon. Some of itsfunctionalities are topology discover, device management (MAC and IP tracking) and web management. Its operation is to translateOpenFlow messages from switches into Floodlight events.

**OpenDaylight Project:** "The OpenDaylight Project[16][17] is a community-led, open, industry-supported framework, for accelerating adoption, fostering new innovation, reducing risk and creating a more transparent approach to Software-Defined Networking.

As a collaborative project under The Linux Foundation, OpenDaylight is structured using open source development best practices, and is comprised of the leading organizations in the technology industry." [Quoted from[6]]

It is a project which aim is to deliver a common Open Source SDN framework across the industry for customers, partners and developers. Its first version called Hydrogen was released on February 4, 2014 and can be downloaded from its website[7]. This version allows vendors and customers alike the ability to utilize standard-based SDN-solutions.

**RYU:** Ryu is a controller developed by the OSRG. As it is the controller chosen for the software developments of this project, it will be fully described at the next topic.

### 5.1.1. Controller: Ryu

Ryu[13] is the key name for a component-based SDN framework. In Japanese Ryu means "flow" and it’s the name given to a water god dragon. It is developed by the OSRG group in the NTT laboratories. It provides an API for developers to create network control applications over protocols like Netconf, OF-config and OpenFlow (1.0 + Nicira Extensions, 1.2 and 1.3).Ryu has been chosen as the software to develop the DMM elements CROWD Local Controller (CLC) and CROWD Regional Controller (CRC). The reasons that make Ryu be the fittest are described next:

**Supports several versions of OpenFlow:**
Ryu gives support as an OpenFlow controller. It can run various versions of the protocol from 1.0 to 1.3.Despite the CLC and CRC have been implemented over version 1.0, the 1.3 allows better performance using ipv6 matching in the flow tables. In this project, the OpenFlow network has been deployed using the 1.0 version and Ethernet matching, but in future changes this version could be switched to 1.3 and the Ethernet matching to IPv6 matching in order to improve performance in the network and code efficiency in the controller.

**Open:**
As Ryu is open source, it can be freely downloaded, built on, or be modified without any restriction or license needed. That is a very important point to consider if we think that the applications derived from this research could be implemented over current networks by a small price.

**Built in python:**
As today there are many expanded protocols (mainly Java or C++) and the most part of the network controllers have been developed in those languages, we have thought

[6] www.opendaylight.org/project February 21 2014
[7] www.opendaylight.org/software/downloads February 21 2014

that Python presents some advantages over them. It is a powerful programming language whose main advantage is to be an interpreted one. That means it is independent from any platform, which gives flexibility to choose the hardware support to the controller. Thinking again in future applications of this research, this platform independence becomes an important point to take care of.

Another thing that turned the balance to the python side is the libraries already developed for python. Mainly [Scapy[8]], which fitted at perfection with some of the project's needs in order to ease the development of the controller elements. Scapy is a Python package for network packet manipulation. It can be used to create, encapsulate, send or capture customized packets from a wide array of protocols. It allows the programmer to work with any element of the packet in an almost unlimited way. It runs natively in Linux using Python interpreter. As its library is very well defined, it can be easily adapted to the use of the OpenFlow controller. In the CROWD Local Controllers some packet manipulation is needed to analyze and respond the incoming packets. Scapy can build exactly the packet desired. Even if it has no sense, like an inexistent stack order. Any value can be putted on any field, and that provides us as developers with almost unlimited possibilities in the controllers.

**Easy to develop:**

We found the Ryu API [14]very well defined and the tutorials on the Ryu website very easy to follow with the step-to-step guide of how to create a Ryu application.

### 5.1.2. Pantou OpenFlow implementation for OpenWRT

Pantou[15] is the key name for the OpenFlow software available for the OpenWRT operating system. It turns a commercial wireless router (running OpenWRT) to an OpenFlow-enabled switch running version 1.0[9].

## 5.2. PROCESS DESIGN

The theoretical design of this project is to develop a network following the CROWD[10] architecture [Figure 6: CROWD network]. This architecture defines the concepts of district and region. A CROWD district is a tier with a limited but fine grain scope for short time scales composed by base stations (LTE, Wi-Fi, APs or eNBs). A CROWD Region is another tier with a broader but coarser grain scope for long time scales. So with the goal of testing the performance of a distributed mobility managed network system, this project has designed a network environment where the DMM mobility proposals can be studied. The main idea here

[8] http://www.secdev.org/projects/scapy/ February 21 2014
[9] https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf February 21 2014
[10] http://www.ict-crowd.eu/downloads/ali-ahmad13crowd.pdf February 21 2014

is to take the theoretical concepts to design a physical network and apply the solution in order to check its viability. The first requirement for the network is to be formed by at least two CROWD districts to test mobility between districts. Each district will have a different gateway to access Internet and one Local Controller under the hierarchical domain of a Regional controller to manage all the districts of its region. Following the CROWD definitions these controllers are the following: A CROWD Local Controller (CLC) can take fast, short time scale decisions on a limited but fine grain scope. A CROWD Regional Controller (CRC) can take slower, long time scale decision with a broader but coarser grain scope.

The controller will be developed in Python[11], and its functions defined are oriented to routing the traffic through the network, even though the competences of a CLC include resources management and QoS, Network Discovery, Topology detection, etc. Those are out of the scope. So the controller will implement all the control plane of the network. A device running OpenFlow controller software will fill the Flow Tables of the network's node in order to provide it with IPv6 internet connection.

The switches of the OpenFlow network will be Linksys WRT54GL [25], a Linux-based router on which we will run Pantou [14].

[11]http://www.python.org/ February 21 2014

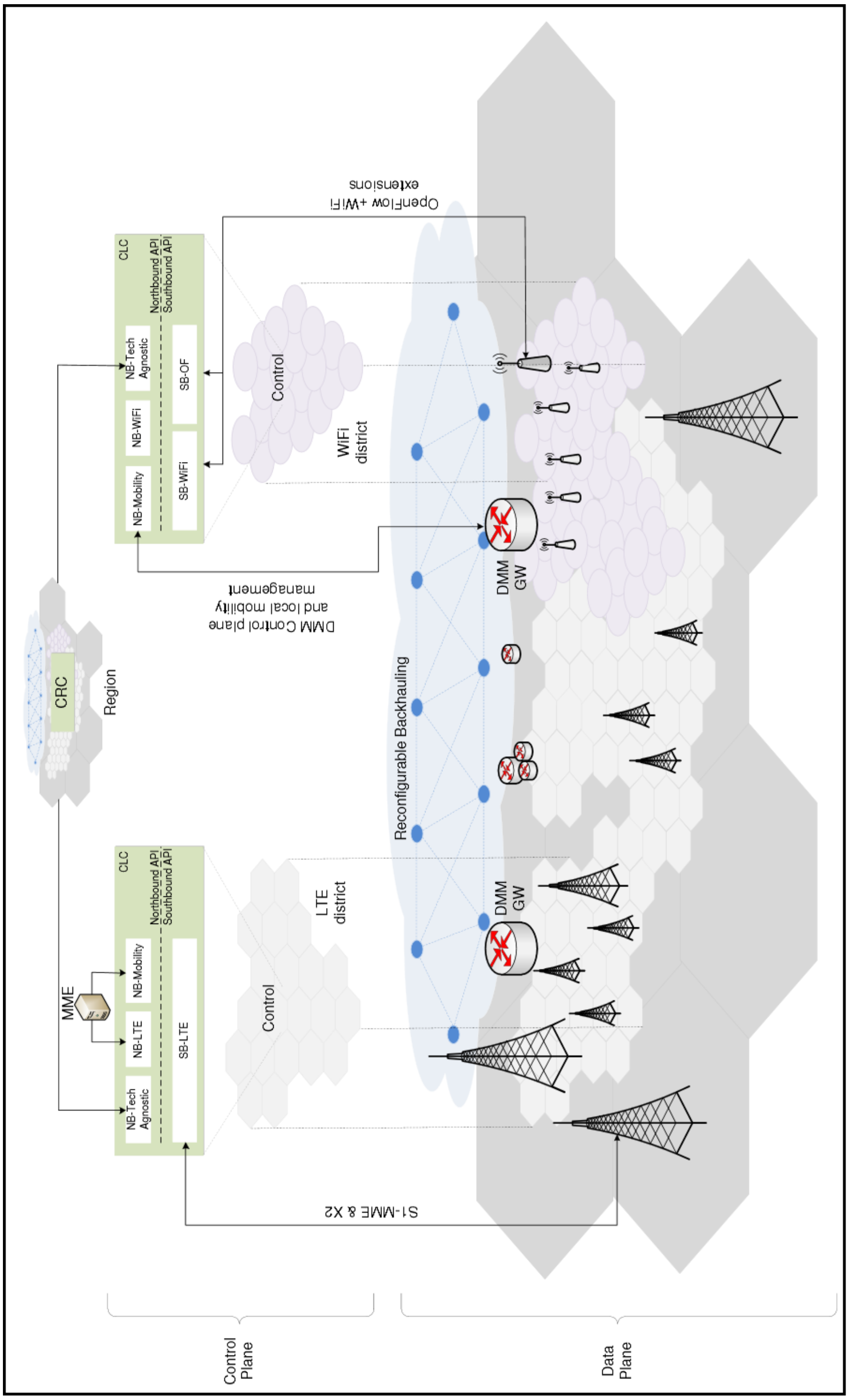


Figure 7: CROWD network[12]

[12]http://www.ict-crowd.eu/ February 21 2014

### 5.2.1. The mobility process

#### *District attachment*

Once the different elements of the DMM network[10] are defined, it's time to do the same with all the processes of all the message exchange between these elements when a Mobile Node is connected through the network. The first attachment to the network will happen in the following way [Figure 7: District attachment]:

1. The Mobile Node attaches to one of the Access Points sending an LLC message to it.

2. The AP performs matching on its Flow Tables with the header fields. As there is no entry with a successful matching, it notifies (forwards the message) to its Local Controller. The Local Controller checks if the host is allowed and if it is stored in the BCE. As it is the first attachment the mobile node will not be registered in the BCE so the controller adds it and configures a network prefix on the DMM Gateway.

3. The Mobile Node once attached starts the IPv6 neighbor discovery sending a Router Advertisement to the Access Point. The Access Point still doesn't have any entry in its flow table referring to the MN, so when it performs matching again it will forward the Neighbor discovery message to the controller.

4. The Access Point forwards the Router Advertisement to the Local Controller who answers with it with the appropriate network prefix configured at the DMM Gateway and performs two more tasks: calculate the shortest path from the access point to the Gateway and send the necessary OpenFlow messages to all the nodes involved in the path with the entries to their Flow Tables.

5. The neighbor discovery continues as the Mobile Node sends a Neighbor Solicitation, but as the path through the network is established the communication will happen directly with the DMM Gateway that answers with a Neighbor Advertisement.At the end of the ICMPv6 process the Mobile Node has connection with the Internet through the DMM Gateway.

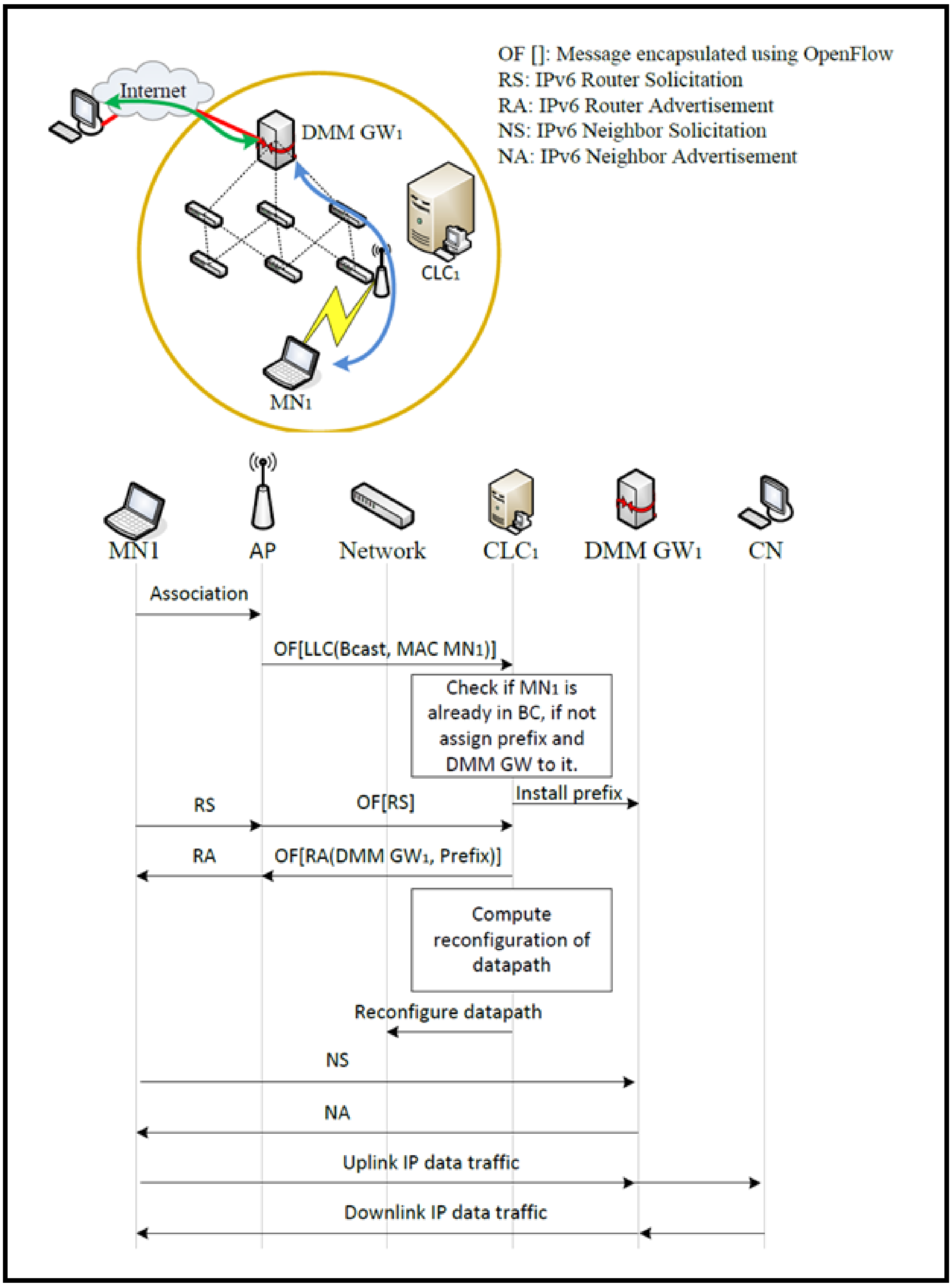

OF []: Message encapsulated using OpenFlow
RS: IPv6 Router Solicitation
RA: IPv6 Router Advertisement
NS: IPv6 Neighbor Solicitation
NA: IPv6 Neighbor Advertisement
Internet
DMM GW1
CLC1
MN1
MN1
AP
Network
CLC1
DMM GW1
CN
Association
OF[LLC(Bcast, MAC MN1)]
Check if MN1 is already in BC, if not assign prefix and DMM GW to it.
RS
OF[RS]
Install prefix
RA
OF[RA(DMM GW1, Prefix)]
Compute reconfiguration of datapath
Reconfigure datapath
NS
NA
Uplink IP data traffic
Downlink IP data traffic

Figure 8: District attachment[13]

[13]http://www.ict-crowd.eu/ February 21 2014

### *District handover*

With the connection established the next test is intra-districtmobility. That happens when an MN once attached to the network changes its attachment to another Access Point of the same district. The handover develops like this [Figure 8: District handover]:

1. The Mobile Node sends an LLC message to the new attachment Access Point.

2. The AP performs matching on its Flow Tables with the header fields. As there is no entry with a successful matching, it notifies (forwards the message) to its Local Controller. The Local Controller checks if the host is allowed and if it is stored in the BCE. This time the Mobile Node is already at the BCE and that is where the controller detects the change of attachment. Then it obtains the prefix assigned previously to the MN and recalculates the path through the OpenFlow network. After that it sends messages to create the new path and delete the old one. When this is done the Mobile Node has connection with the Internet through the DMM Gateway following the new path.

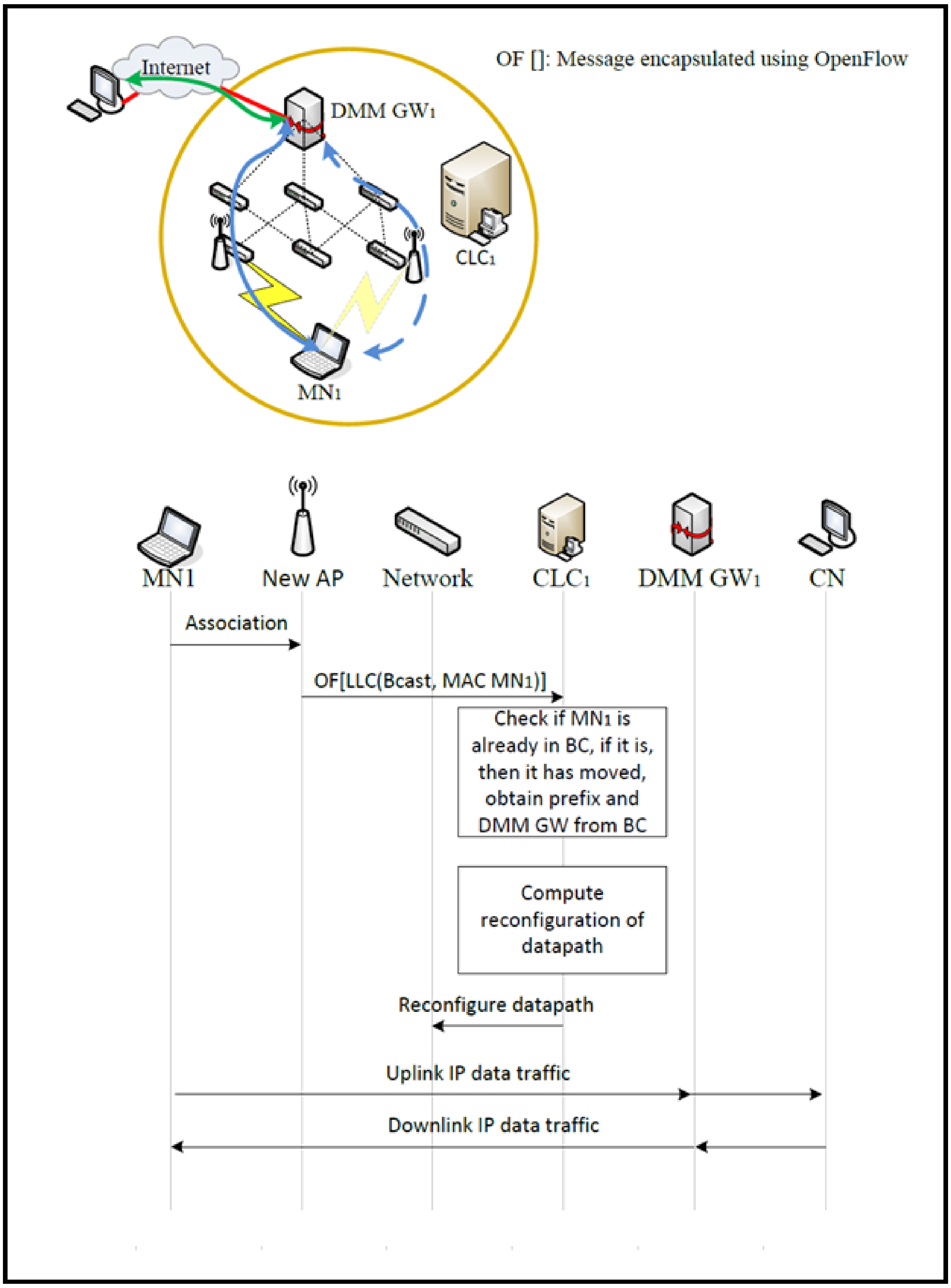

OF []: Message encapsulated using OpenFlow
Internet
DMM GW1
CLC1
MN1
MN1
New AP
Network
CLC1
DMM GW1
CN
Association
OF[LLC(Bcast, MAC MN1)]
Check if MN1 is already in BC, if it is, then it has moved, obtain prefix and DMM GW from BC
Compute reconfiguration of datapath
Reconfigure datapath
Uplink IP data traffic
Downlink IP data traffic

Figure 9: District handover[14]

[14]http://www.ict-crowd.eu/ February 21 2014

### *Region attachment*

The next step in the study is the mobility between different districts. The process is defined as handover inter-district and happens when a Mobile Node once attached to a DMM network changes its attachment to another Access Point of a different district. For this event the environment must be expanded with another district to include a CROWD Regional Controller. The first attachment in the CROWD network develops like this [Figure 9: Region attachment]:

1. The Mobile Node changes its attachment sending an LLC message to theAccess Point.

2. The AP performs matching on its Flow Tables with the header fields. As there is no entry with a successful matching, it notifies (forwards the message) to its Local Controller. The Local Controller checks if the host is allowed and if it is stored in the BCE. As it is the first attachment the mobile node will not be registered in the BCE so the Local Controller asks to the Regional Controller about the Mobile node.

3. The Regional Controller checks if the Mobile Node is in its own BCE. As it is not, it answers to the Local Controller with a negative answer.

4. The Local Controller adds the Mobile Node to its BCE and configures a network prefix on the DMM Gateway. After that it notifies to the Regional Controller about the new attachment to the network. The Regional Controller will add the MN to its BCE with the prefix and Gateway assigned.

5. The Mobile Node once attached starts the IPv6 neighbor discovery sending a Router Advertisement to the Access Point. The Access Point still doesn't have any entry in its flow table referring to the MN, so when it performs matching again it will forward the Neighbor discovery to the controller.

6. The Access Point forwards the Router Advertisement to the Local Controller who answers with it with the appropriate network prefix configured at the DMM Gateway and performs two more tasks: calculate the shortest path from the access point to the Gateway and send the necessary OpenFlow messages to all the nodes involved in the path with the entries to their Flow Tables.

7. The neighbor discovery continues as the Mobile Node sends a Neighbor Solicitation, but as the path through the network is established the communication will happen directly with the DMM Gateway that answers with a Neighbor Advertisement. At the end of this process the Mobile Node has connection with the Internet through the DMM Gateway.

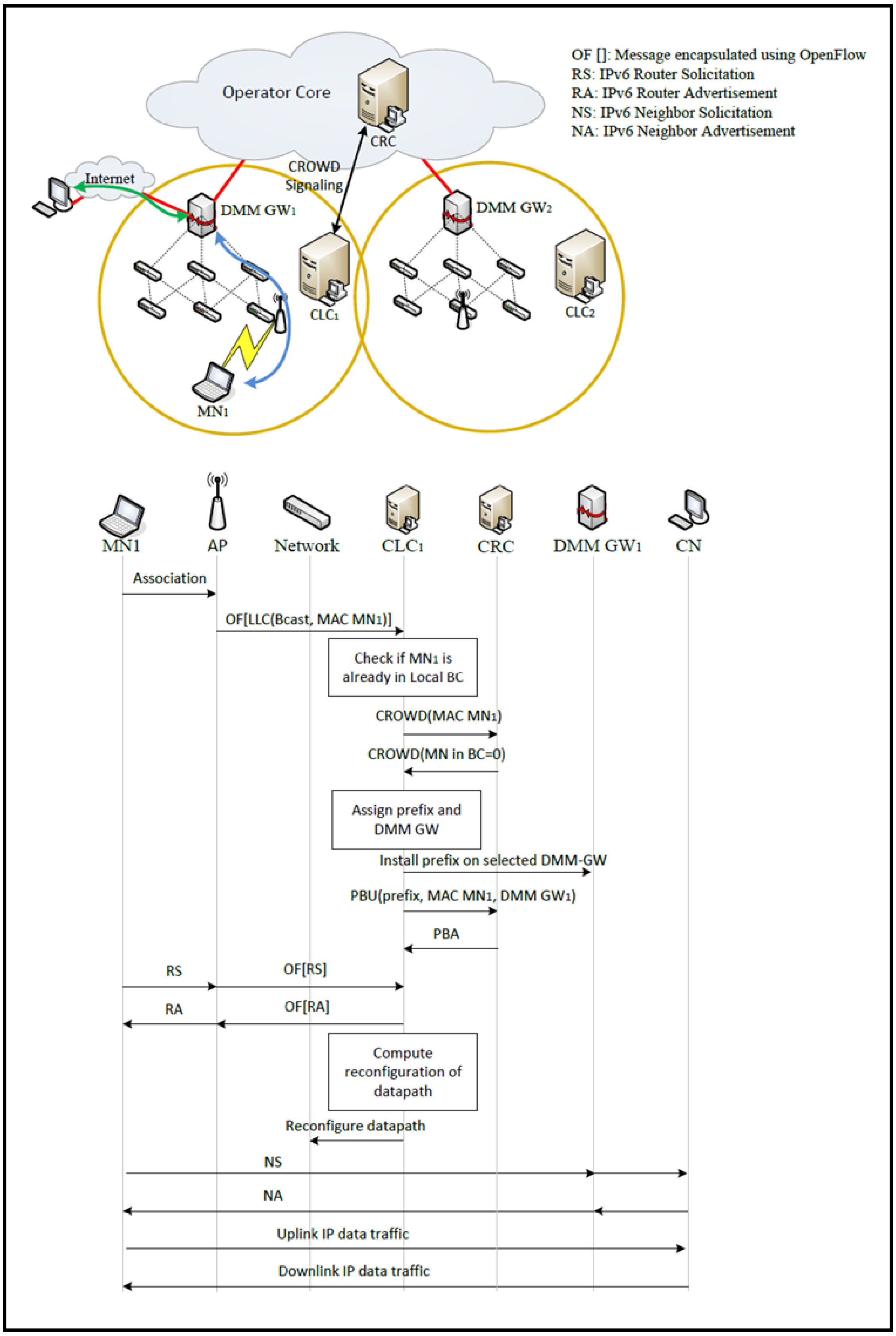

OF []: Message encapsulated using OpenFlow
RS: IPv6 Router Solicitation
RA: IPv6 Router Advertisement
NS: IPv6 Neighbor Solicitation
NA: IPv6 Neighbor Advertisement
Operator Core
CRC
Internet
CROWD Signaling
DMM GW1
DMM GW2
CLC1
CLC2
MN1
MN1
AP
Network
CLC1
CRC
DMM GW1
CN
Association
OF[LLC(Bcast, MAC MN1)]
Check if MN1 is already in Local BC
CROWD(MAC MN1)
CROWD(MN in BC=0)
Assign prefix and DMM GW
Install prefix on selected DMM-GW
PBU(prefix, MAC MN1, DMM GW1)
PBA
RS
OF[RS]
RA
OF[RA]
Compute reconfiguration of datapath
Reconfigure datapath
NS
NA
Uplink IP data traffic
Downlink IP data traffic

Figure 10: Region attachment[15]

[15]http://www.ict-crowd.eu/ February 21 2014

### *Region handover*

With the connection established it is time to perform inter-district mobility. That happens when an MN once attached to the network changes its attachment to another Access Point of another district. This process also involves the Regional controller. The handover develops like this [Figure 10: Region handover]:

1. The Mobile Node sends an LLC message to an Access Point of a different district of its attachment.

2. The AP performs matching on its Flow Tables with the header fields. As there is no entry with a successful matching, it notifies (forwards the message) to its Local Controller. The Local Controller checks if the host is allowed and if it is stored in the BCE. As it is the first attachment in this district the mobile node will not be registered in the BCE so the Local Controller asks to the Regional Controller about the Mobile node.

3. The Regional Controller checks if the Mobile Node is in its own BCE. As it is since its previous attachment in the other district, it will answer to the Local Controller with the information of assigned prefix and previous DMM Gateway assigned in an affirmative answer.

4. The Local Controller adds the Mobile Node to its BCE and configures a network prefix on the DMM Gateway. After that, it notifies to the Regional Controller about the new attachment to the network. The Regional Controller will add the MN to its BCE with the prefix and Gateway assigned.

5. The Local Controller configures its assigned DMM Gateway with the prefix received from the Regional Controller and sends an update to the Regional Controller notifying about the same prefix and the new DMM Gateway.

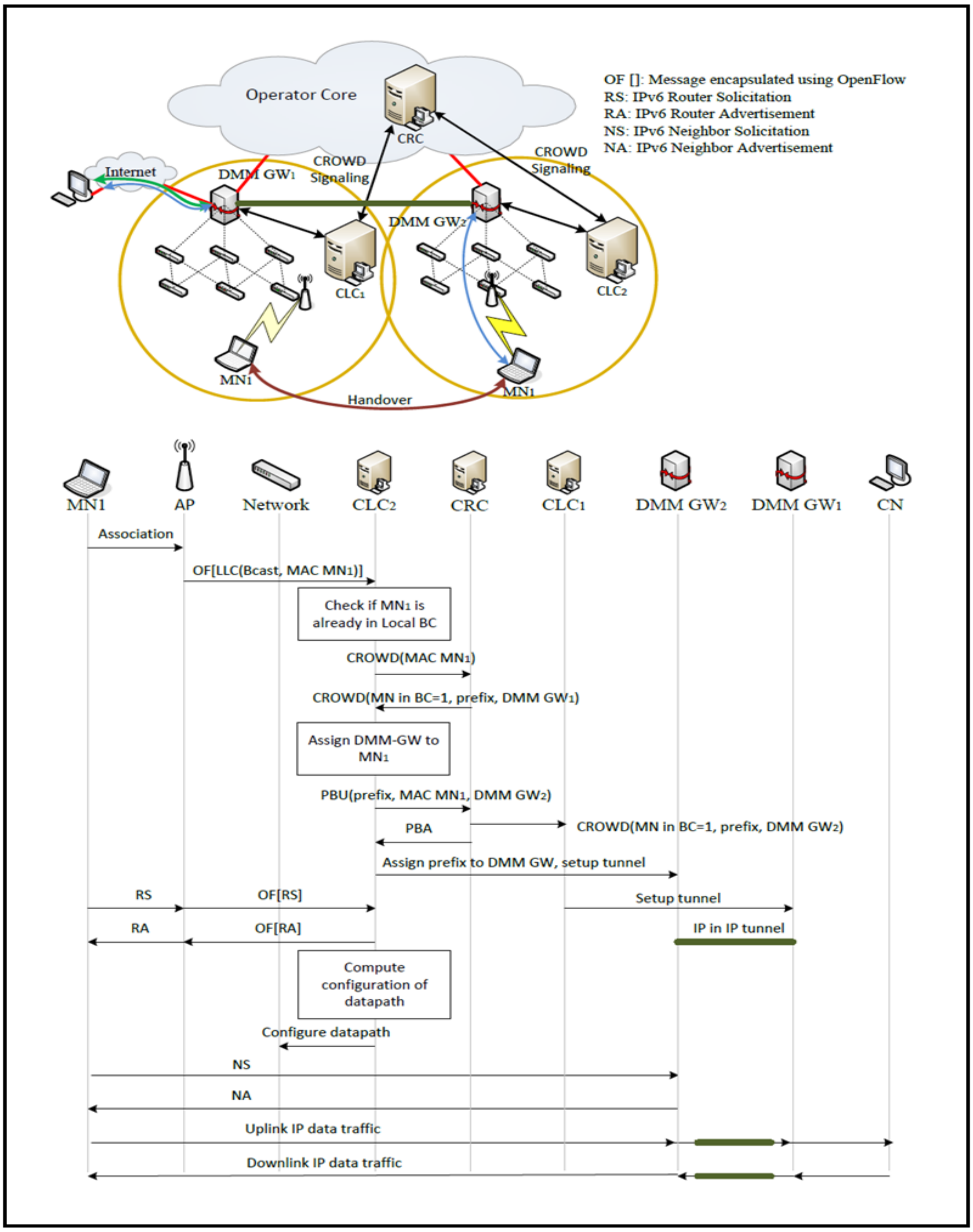
6. The Local Controller will establish an IPv6 tunnel between its Gateway and the one of the old district (information retrieved from the CRC) to use it for redirecting all the traffic destined to the mobile node to its. As the tunnel must be established in both Gateways, the old Local Controller must configure its own one too.

7. The Mobile Node once attached starts the IPv6 neighbor discovery sending a Router Advertisement to the Access Point. The Access Point still doesn't have any entry in its flow table referring to the MN, so when it performs matching again it will forward the Neighbor discovery to the new controller.

8. The Access Point forwards the Router Advertisement to the Local Controller who answers with it with the appropriate network prefix configured at the DMM Gateway and performs two more tasks: calculate the shortest path from the access point to the Gateway and send the necessary OpenFlow messages to all the nodes involved in the path with the entries to their Flow Tables.

9. The neighbor discovery continues as the Mobile Node sends a Neighbor Solicitation, but as the path through the network is established the communication

will happen directly with the DMM Gateway that answers with a Neighbor Advertisement. At the end of the ICMPv6 process the Mobile Node has connection with the Internet through the OpenFlow network and the IPv6 tunnel between the two DMM Gateways.



Operator Core
CRC
OF []: Message encapsulated using OpenFlow
RS: IPv6 Router Solicitation
RA: IPv6 Router Advertisement
NS: IPv6 Neighbor Solicitation
NA: IPv6 Neighbor Advertisement
CROWD Signaling
CROWD Signaling
Internet
DMM GW1
DMM GW2
CLC1
CLC2
MN1
MN1
Handover
MN1
AP
Network
CLC2
CRC
CLC1
DMM GW2
DMM GW1
CN
Association
OF[LLC(Bcast, MAC MN1)]
Check if MN1 is already in Local BC
CROWD(MAC MN1)
CROWD(MN in BC=1, prefix, DMM GW1)
Assign DMM-GW to MN1
PBU(prefix, MAC MN1, DMM GW2)
PBA
CROWD(MN in BC=1, prefix, DMM GW2)
Assign prefix to DMM GW, setup tunnel
RS
OF[RS]
Setup tunnel
RA
OF[RA]
IP in IP tunnel
Compute configuration of datapath
Configure datapath
NS
NA
Uplink IP data traffic
Downlink IP data traffic

Figure 11: Region handover[16]

[16]http://www.ict-crowd.eu/ February 21 2014

## 5.3. IMPLEMENTATION

The environment designed to test the distributed mobility management is a small sample of what could be a real DMM network. The network will have two districts, each one with its own CLC and DMM Gateway. Both are part of a region with a CRC managing them. With this scenario mobility can be tested in intra-district and inter-districts handovers. The full test process has been divided into three stages:

**Stage 1: Intra-district mobility on IPv4**

A small network has been built in order to test the controller performance on mobility. The network consists of two access points, one node and a gateway [Figure 7: Project design - stage 1]. The goal of this stage is to check how the controller manages OpenFlow signaling. For this we developed the CLC with ARP and IPv4 management functionalities. For the tests we only performed intra-district handovers due to the fact that we only want to check how OpenFlow operates with Ryu, and to figure how to adapt it to our further developments.

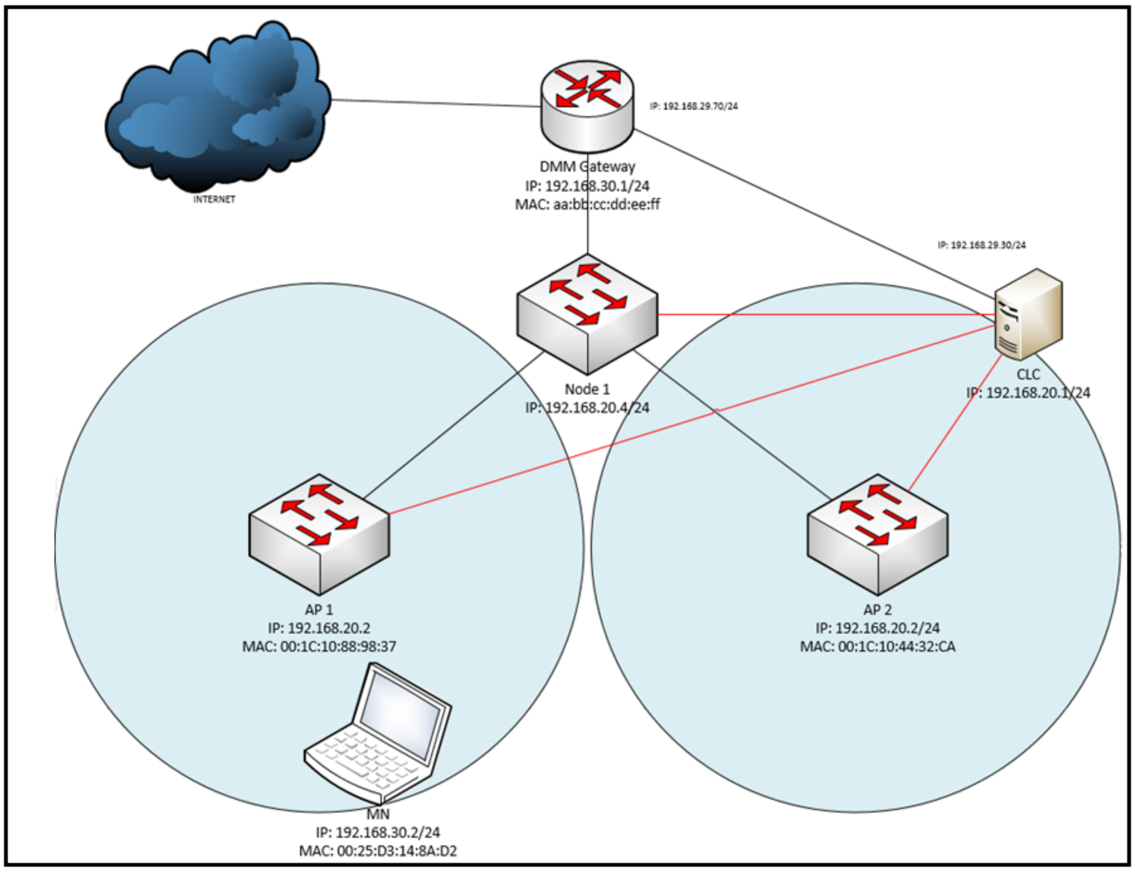


Figure 12: Project design - stage 1

**Stage 2: Intra-district mobility on IPv6**

The second is an extension of the controller adding IPv6 packet handling. The OpenFlow network is the same designed in the first stage, the difference is only at the controller, which now has been extended with IPv6 handling functionality.The ARP and IPv4 code has been replaced by ICMPv6 and IPv6. The ICMPv6 handler listens from packets received at the AP only, and it will use the BCE to guess if the MN is performing an attachment or a handover. The CLC will answer the RS of the MN. The IPv6 handler will accept packets from the MN or from the DMM Gateway. [Figure 8: Project design - stage 2].

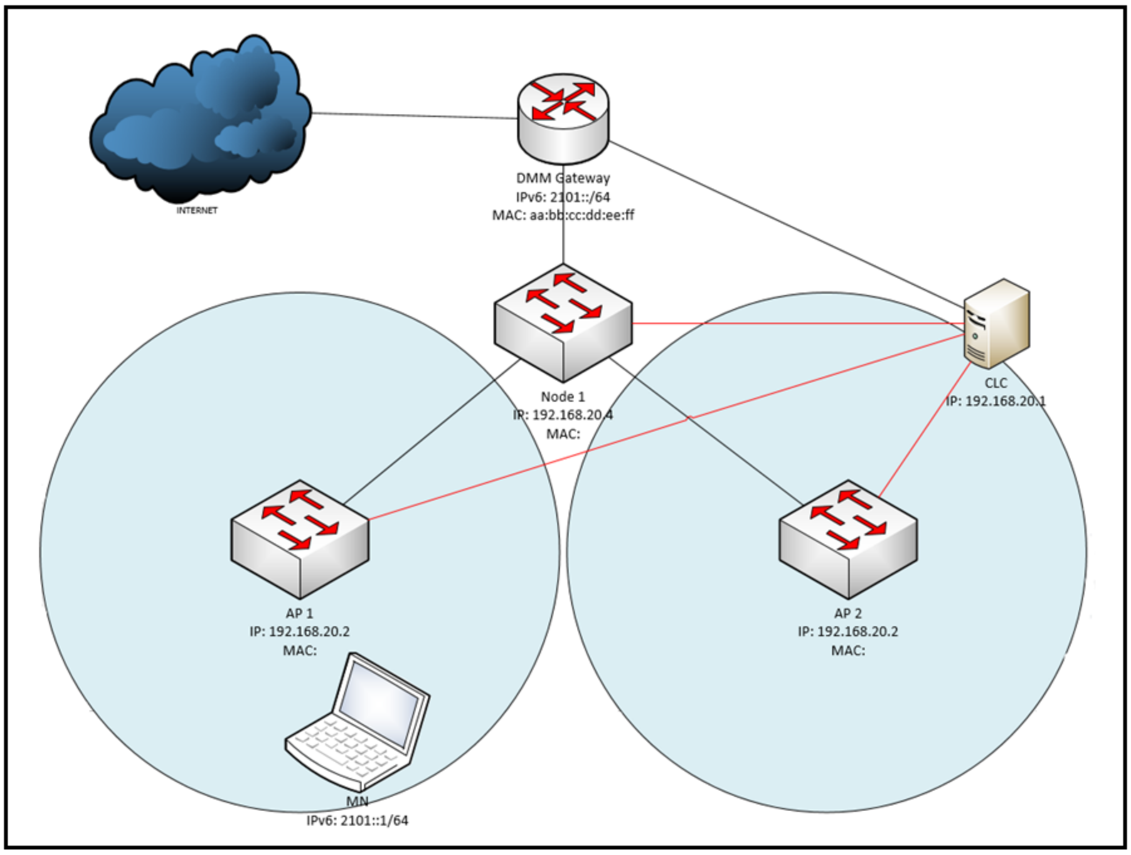


Figure 13: Project design - stage 2

**Stage 3: Inter-district mobility on IPv6**

The final stage is a complete DMM network. The OpenFlow network built for the first two stages has been now replicated into a second one and the gateways are considered now as DMM gateways [Figure 9: Project design - stage 3]. One Regional controller has been added in order to handle the intra-district handovers (handovers between the two OpenFlow networks). This CRC is just a python program, it doesn't run over Ryu. The CLCs has been quite

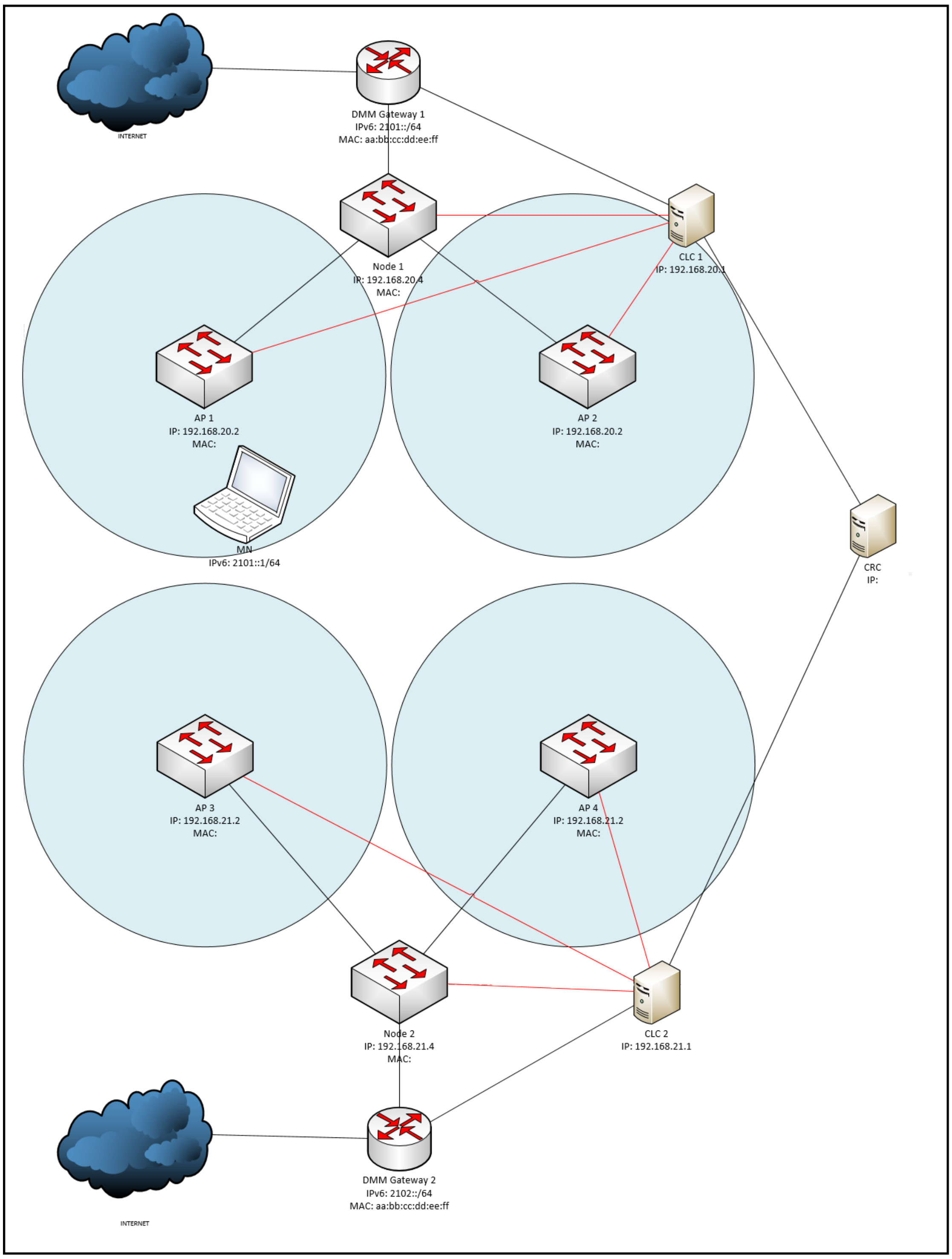


Figure 14: Project design - stage 3

modified to be able now to handle a TCP connection with the CRC and an UDP connection with the DMM Gateway to configure them. During the inter-district mobility, the CLCs uses that connection to set up an IPv6 over IPv6 tunnel between the DMM Gateways to forward the traffic through it.

### 5.1.2. Actors

The following elements will take part in the experiments:

**CROWD Regional Controller:** Manages mobility intra-district on its regional network. For that, it has a cache containing tuples of: Mobile Nodes attached to the network –DMM Gateway of the district they are attached to. For this element we have chosen to use a computer running python.

**CROWD Local Controller:** Manages its local district. For this element we have chosen to use another computer running Ubuntu for each one of the two CLC of the testbed and the Ryu OpenFlow controller.

**Mobile Node:** It is the host of the OpenFlow network. It doesn’t have any kind of OpenFlow knowledge; its only requirement is to have wireless connection and an IPv6 neighbor discovery (ICMPv6) protocol running. For this element we have chosen a laptop computer.

**Access Point:** An OpenFlow switch with the traditional behavior of a wireless access point. Provides connection to the Mobile Node. We will use two switches as access point per each district and they will be Linksys WRT54GL running OpenFlow Pantou.

**Node:** An OpenFlow switch with its wireless connection disabled. It has the same behavior of the Access points, with the only difference of the wireless connection. When a Node is directly connected to a Gateway/DMM Gateway is considered as an OpenFlow Edge node. There is one per district, and it will also be a Linksys WRT54GL running OpenFlow Pantou.

**DMM Gateway:** It’s a non OpenFlow device. It is a normal router providingthe OpenFlow network of internet connection. It is linked to the Local Controller to be remotely configured. There is one per district, and it will be a Linksys OpenWRT54GL device running the WRT Backfire version.

**IPv6 tunnel:** An IPv6 over IPv6 tunnel established between the districts’s DMM Gateways when an inter-district mobility happens. It is used to forward all the traffic destined to the MN during the roaming.

## 5.4. HOW TO INSTALL

The following pointcontains a step-to-step guide of how to download, install, configure and set up all the necessary elements of the OpenFlow Network test bed.

### 5.4.1. RYU

**INSTALLATION:**

Ryu can be installed in any device running Linux. This installation has been done in a Linux machine running Ubuntu. It is very easy just a python interpreter is needed. To download it just execute the following commands on the terminal:

```
% sudo apt-get update

% sudo apt-get install python
```

Now the interpreter is downloaded and installed. To get the python packages needed, it is recommended to download de python package installer PIP (Python Package Index).

```
% apt-get install python-pip
```

Now, Ryu can be downloaded with pip using the following command:

```
% pip install ryu
```

Or it can be downloaded from the source code using the version controller GIT using the following commands:

```
% git clone git://github.com/osrg/ryu.git
% cd ryu; python ./setup.py install
```

The Ryu controller built in this project also needs some extra python packages installed. To get them just execute these commands:

**TCP:**If this package is not included in your python version by default just download it with python pip:

```
% pip install python-handler-socket
```

**Subprocess:** This package is already included in the python main installation as a python standard library.

**Billiard/multiprocessing/multithreading:** This library changes its name between python versions, but is always included as a python standard library.

**Scapy:** For the Scapy package download just follow the next steps:

```
% cd /tmp

% wget scapy.net

% unzip scapy-latest.zip

% cd scapy-2-.*

% sudo python setup.py install
```

Now the device has all the required software installed, and it is ready to configure and run.

**CONFIGURATION:**

The device designed as OpenFlow controller just uses one Ethernet interface to communicate with the OpenFlow network. This interface must be configured with the appropriate IPv4 address.

```
% sudo ip addr add 192.168.20.1/24 dev eth1

% sudo ifconfig eth1 up
```

The resulting configuration in the controller for that interface can be checked using

```
user@controller:~$ sudo ifconfig
```

Andmust look like:

```
eth1     Link encap:Ethernet  direcciónHW 00:13:f7:ff:08:ef
         Direc. inet:192.168.20.1  Difus.:0.0.0.0  Másc:255.255.255.0
         Dirección inet6: fe80::213:f7ff:feff:8ef/64 Alcance:Enlace
         ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
         Paquetes RX:30814 errores:0 perdidos:0 overruns:0 frame:0
         Paquetes TX:33095 errores:0 perdidos:0 overruns:0 carrier:0
         colisiones:0 long.colaTX:1000
         Bytes RX:3055990 (3.0 MB)  TX bytes:2251107 (2.2 MB)
         Interrupción:19 Dirección base: 0xd000
```

The Ryu controller device is now configured.

### 5.4.2. PANTOU

**INSTALLATION:**

Pantou[15] will be the software installed in the routers to turn them into OpenFlow switches following the Stanford reference. It is based on OpenWRT Backfire version (Linux 2.6.32).

Pantou works for these devices:

- Linksys WRT54GL
- TP-LINKTL-WR1043ND
- Generic routers with chipset Broadcom BCM47XX

The chosen device for the testbed is the Linksys WRT54GL. To turn it into an OpenFlow switch the following steps must to be followed:

- Download the appropriate Pantou image
- Load the image on the router
- Check the functionality

1. Download the Pantou image.

The image can be downloaded from the Pantou web in the following URL:

http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT

The images are in the "supported devices" point. Clicking on the selected device (Linksys WRT54GL) will start the download.

2. Uploading the image to the router

Once the download has ended there are various ways of uploading the image to the router. The chosen here is to set the router on failsafe mode and then transfer the image using the tftp command (Trivial File Transfer Protocol). Setting the router on failsafe mode is not mandatory, but prevents future incompatibility problems installing Pantou. The generic way to do this requires the following steps starting with the router being power fed:

1. Press and hold the Reset button located on the back of the WRT54GL.
2. While holding the reset button, unplug the power cord and plug it again. The POWER and DMZ LEDs on the front will start blinking.
3. Wait until DMZ led stops blinking, when this happen release the Reset button, after a few seconds the device will reboot into failsafe mode.

Now that the router is on failsafe mode, it can be accessed by an Ethernet connection to one of its interfaces, it doesn't matter which one is chosen because the router on failsafe mode is configured as a bridge by default.

To communicate with the router, the computer must have its interface configured with an IPv4 of the same subnet. As the router has the default IPv4 192.168.1.1/24 by default on all of its interfaces, the computer must be configured for example with the 192.168.1.2/24. Using the following command:

```
% sudo ip addr add 192.168.1.2/24 dev interface_name
```

Once this is done, the router can be accessed if it has ended with its reboot on failsafe mode. One way of checking that it is ready is sending ping messages to it.

```
% ping 192.168.1.1
```

If the correct echoes are received the router is ready to receive de image. To use tftp just execute the following commands (located in the directory of the previously downloaded image):

```
% tftp 192.168.1.1

tftp> Binary

tftp> trace

tftp> rexmt1

tftp> timeout 90

tftp> put nombre_imagen_pantou.bin
```

In this moment, the image will start to be sent unsuccessfully. While this is happening the failsafe mode steps must be repeated in order to make the router accept the image sent.

1. Press and hold the Reset button located on the back of the WRT54GL.
2. While holding the reset button, unplug the power cord and plug it again. The POWER and DMZ LEDs on the front will start blinking.
3. Wait until DMZ led stops blinking, when this happen release the Reset button, after a few seconds the device will accept the image.

Now the connection with the router may be lost because Pantou is not configured as a bridge by default, and the interface configuration is the following:

Interface Internet: 192.168.1.1/24

Interface 0: No IP

Interface 1: No IP

Interface 2: No IP

Interface 3: No IP

To recover the connection, the Ethernet wire must be connected to the port labeled as "internet". Then the router can be accessed using the IPv4 address configured on the computer. (To check the rooter has ended its reboot process ping 192.168.1.1).

To check the Pantou installation on the router and the right working of OpenFlow, just access to the router via Telnet

```
% telnet 192.168.X.1
```

And execute these commands:

```
% ps aux | grep ofprotocol
% ps aux | grep ofdatapath
```

**CONFIGURATION:**

To configure the OpenFlow switch with the designed network configuration, three configuration files must be edited (using vim command for example).

1. Wireless file:

Location: /etc/config/wireless

By default this file has the wireless connection disabled. On the switches designed as Access Points this must be enabled. To do that, just comment the line that disables wireless tipping an "#" at the beginning.

The file must look like this:

```
root@OpenWrt:/# cat etc/config/wireless

config wifi-device  radio0
        option type     mac80211
        option channel  5
        option macaddr  00:1c:10:88:98:37
        option hwmode   11g

        # REMOVE THIS LINE TO ENABLE WIFI:
        #option disabled 1

config wifi-iface
        option device   radio0
        option network  lan
        option mode     ap
        option ssid     OpenWrt
        option encryption none
```

2. Network file:

Location: /etc/config/network

This file will set the OpenFlow interface's IPv4 address to establish connection between the Ryu Controller and each switch. In this project, the interface labeled as "internet" is the chosen as control interface. That interface will be the only one configured with an IPv4 address, and it must belong to the same subnet of the Controller's one. The following text shows the final configuration of a switch with a control port with the IPv4 settled to 192.168.20.2.

```
root@OpenWrt:/# cat etc/config/network
#### VLAN configuration
config switch eth0
        option enable   1

config switch_vlan eth0_0
        option device   "eth0"
        option vlan     0
        option ports    "0 5"

config switch_vlan eth0_1
        option device   "eth0"
        option vlan     1
        option ports    "1 5"

config switch_vlan eth0_2
        option device   "eth0"
        option vlan     2
        option ports    "2 5"

config switch_vlan eth0_3
        option device   "eth0"
        option vlan     3
        option ports    "3 5"

config switch_vlan eth0_4
        option device   "eth0"
        option vlan     4
        option ports    "4 5"


#### Loopback configuration
config interface loopback
        option ifname   "lo"
        option proto    static
        option ipaddr   127.0.0.1
        option netmask  255.0.0.0


#### LAN configuration
config interface
        option ifname   "eth0.0"
        option proto    static

config interface
        option ifname   "eth0.1"
        option proto    static
```

```
config interface
        option ifname   "eth0.2"
        option proto    static

config interface
        option ifname   "eth0.3"
        option proto    static

config interface
        option ifname   "eth0.4"
        option proto    static
        option ipaddr   192.168.20.2
```

As showed, the ports have been splitted up, but they have none address assigned. Only the controller interface has one.

3. OpenFlow configuration file:

Location /etc/config/openflow

This file configures the OpenFlow functionality of the switch. It has just 4 options described like this:

- dp: names the OpenFlow datapath
- ofports: sets which ports will act as OpenFlow ports
- ofctl: sets the controller's ipv4 and port
- mode: defines the switch functionality as inband or outofband (the second one separates the control and data planes)

This file once configured for an AP must look like this:

```
root@OpenWrt:/# cat etc/config/openflow

config 'ofswitch'
        option 'dp' 'dp0'
        option 'ofports' 'wlan0 eth0.0 eth0.1 eth0.2 eth0.3'
        option 'ofctl' 'tcp: 192.168.20.1:6633’
        option 'mode'  'outofband'
```

If the switch is a normal node instead of an Access Point, the “wlan0” port must be removed from the ofports.

Now that the switch is fully configured, it must be network rebooted in order to commit the changes. That can be done executing the following process:

```
root@OpenWrt:./etc/init.d/network restart
```

Now the connection with the switch is lost again because of the new configuration of the router is taking effect erasing the IPv4 that was being used (The 192.168.1.1 is now a 192.168.20.X). The telnet process will hang out, so it should be closed and open a new one.

To connect now with the switch, a new IPv4 must be assigned to the Controller, one belonging to the 192.168.20.0/24 network an access the switch via its “internet” port.

### 5.4.3. ADDITIONAL CONFIGURATIONS:

**DMM Gateways:**

As the MN always has the same interface address as default gateway, All the DMM Gateway must have the same MAC address (so they have the same link local ipv6 address). This is done using the following command:

```
root@OpenWrt:/# ifconfig eth0.3 down

root@OpenWrt:/# ifconfig eth0.3 hw ether aa:bb:cc:dd:ee:ff

root@OpenWrt:/# ifconfig eth0.3 up
```

By default, the Backfire[17] version installed in the DMM Gateway has not the IPv6 forwarding enabled. To enable it we just have to modify the sysctl file of the switch. That is done executing the following commands:

```
% sysctl –a | grep forwarding
```

So we can check if in fact these options are deactivated.

```
net.ipv6.conf.all.forwarding = 0
net.ipv6.conf.all.mc_forwarding = 0
net.ipv6.conf.default.forwarding = 0
net.ipv6.conf.default.mc_forwarding = 0
net.ipv6.conf.lo.forwarding = 2
net.ipv6.conf.lo.mc_forwarding = 0
net.ipv6.conf.eth0.forwarding = 0
net.ipv6.conf.eth0.mc_forwarding = 0
net.ipv6.conf.eth0.0.forwarding = 2
net.ipv6.conf.eth0.0.mc_forwarding = 0
net.ipv6.conf.eth0.1.forwarding = 0
net.ipv6.conf.eth0.1.mc_forwarding = 0
net.ipv6.conf.eth0.2.forwarding = 0
net.ipv6.conf.eth0.2.mc_forwarding = 0
net.ipv6.conf.eth0.3.forwarding = 2
net.ipv6.conf.eth0.3.mc_forwarding = 0
net.ipv6.conf.eth0.4.forwarding = 0
sysctl: error reading key 'net.ipv6.route.flush': Permission denied
net.ipv6.conf.eth0.4.mc_forwarding = 0
net.ipv6.conf.ip6tnl0.forwarding = 0
net.ipv6.conf.ip6tnl0.mc_forwarding = 0
```

[17] http://downloads.openwrt.org/backfire/10.03.1/brcm47xx/ February 21 2014

To change this we have to tip the following:

```
% sysctl –w net.ipv6.conf.all.forwarding=1
```

And now we can check that forwarding is now enabled in our device.

```
net.ipv6.conf.all.forwarding = 1
net.ipv6.conf.all.mc_forwarding = 0
net.ipv6.conf.default.forwarding = 1
net.ipv6.conf.default.mc_forwarding = 0
net.ipv6.conf.lo.forwarding = 1
net.ipv6.conf.lo.mc_forwarding = 0
net.ipv6.conf.eth0.forwarding = 1
net.ipv6.conf.eth0.mc_forwarding = 0
net.ipv6.conf.eth0.0.forwarding = 1
net.ipv6.conf.eth0.0.mc_forwarding = 0
net.ipv6.conf.eth0.1.forwarding = 1
net.ipv6.conf.eth0.1.mc_forwarding = 0
net.ipv6.conf.eth0.2.forwarding = 1
net.ipv6.conf.eth0.2.mc_forwarding = 0
net.ipv6.conf.eth0.3.forwarding = 1
net.ipv6.conf.eth0.3.mc_forwarding = 0
net.ipv6.conf.eth0.4.forwarding = 1
sysctl: error reading key 'net.ipv6.route.flush': Permission denied
net.ipv6.conf.eth0.4.mc_forwarding = 0
net.ipv6.conf.ip6tnl0.forwarding = 1
net.ipv6.conf.ip6tnl0.mc_forwarding = 0
```

### 5.4.4. LAUNCH THE SYSTEM:

1. Launching the Controller:

The controller must be launched over the python path and it requires two arguments. The first one is the `ryu_manager` of the Ryu controller. The second one is the python controller, Ryu has some controllers included by default on its download that can be used for testing, overwrite or simply as a controller with default performance. The following line shows an example of the controller launching:

```
% PYTHONPATH=. ./ryu’s_path/bin/ryu_manager
./controller’s_path/controlador.py
```

Or if we want the verbose detailed output we can add "–verbose" at the end.

```
% PYTHONPATH=. ./ryu’s_path/bin/ryu_manager
./controller’s_path/controlador.py –verbose
```

Once this is done, the controller is running and its output will be displayed on the window.

2. Launching the nodes:

To run OpenFlow on the switches, just access to the switch using Telnet or ssh commands on its Control IPv4 and execute the following command:

```
% ./etc/init.d/openflow start
```

To reset Openflow (erase the FlowTables and reset the system):

```
% ./etc/init.d/openflow restart
```

To stop OpenFlow (erase the FlowTables and stop the system):

```
% ./etc/init.d/openflow stop
```

3. Launching the DMM Gateways:

The DMM Gateways must launch its listener:

```
% ./../usr/udp_server
```

## 5.5. SOFTWARE DEVELOPED

### 5.5.1. THE LOCAL CONTROLLER

The OpenFlow controller (CLC) developed has to handle all the traffic through the district under its management and also communicate with the CRC to notify the attachment updates performed by it.To fulfill this, the execution has been divided in two threads. The main thread's function is to be the OpenFlow message listener and also has the active part of the communication with the CRC. That is when the communication is started by the CLC. The second thread only has the passive communication function. It waits listening to the CRC on an UDP port and acts only when it receives some orders. With those orders, it configures the Gateways to redirect the traffic during inter-district handovers. In [Figure 15: CLC OF listener's flow diagram] and [Figure 16: CLC UDP listener's flow diagram] the “Start” and “Define variables” are common, the two threads split in the “Start thread” box.

**The OpenFlow listener:**

The listener process is launched when the Controller is started. It will handle four network protocols: IPv6 and ICMPv6. Any packet received of a different protocol will be discarded. The first task done on a packet is to check its origin (if it is an incoming packet from the Internet or from an access point).If it comes from a host it checks if it is authorized or not. If it is not, the user will be blocked. Then it distinguishes between the two protocols admitted. When the message is ICMPv6 it means a new attachment, so it checks if that attachment comes from a handover (if that is the case, it removes the old path) and then installs the new path and answers the ICMPv6 message with the information necessary to continue the communication directly with the DMM Gateway. If it is an IPv6 message, it checks if the message comes from the gateway, if that is the case it installs the new path through the network to the Mobile Node.

The operation described in [Figure 15: CLC OF listener's flow diagram]:

1. The global variables of the CLC are defined such as the ACL, CLC, Gateway parameters or dictionaries.

2. The second thread is launched.

3. The CLC gets into listening mode waiting for incoming OpenFlow messages from the switches until one arrives.

4. When a message arrives the CLC checks if the MAC is authorized in the network.

5. If the MN's MAC is not authorized, that user is blocked and the packet is discarded.

6. The CLC checks if the message is an IPv6 packet or ICMPv6 (RS).

7. If it is an RS first checks if the MN’s MAC address is registered in the BCE in order to determine if this is a new attachment to the network or an intra-district handover.

8. If the MN is not in the BCE, the CLC asks the CRC if the MN is in its cache in order to determine if it is a first attachment to the region o an inter-district handover. For that it creates a TCP message containing an “ask” identifier, the MN’s MAC address and the IPv6 of the district’s DMM Gateway.

9. The MN waits to the answer of the CRC, when it receives it checks if it is a “known MN” or “unknowns MN”.

10. If the answer is “unknowns MN” means that it is a first attachment to the region so the CLC adds the MN to its BCE. In order to make the memory access atomicity, the CLC listener will lock a thread before modifying the BCE and release it after doing it.

11. The CLC answers with RA filled using the prefix given by the CRC if the message received was a “known MN” or generate a new one if the message was “unknown MN”.

12. If the answer is “known MN” means that the MN is roaming from another district so the CLC creates an IPv6 over IPv6 tunnel end. The traffic redirection is done by source routing, that implies 3 steps: create the tunnel, add a default rule for all the traffic coming from the MN on a customized routing table, and an order in that table of forwarding all the MN’s traffic through the tunnel.

13. The CLC calculates the shortest path through its district using Djikstra algorithm.

14. The CLC notifies the CRC with the prefix assigned to the MN and the IPv6 address of the district’s DMM Gateway.

15. If the MN is already stored in the BCE it means it is moving (repeated RS in different AP). So the CLC calculates the path to delete and sends the appropriate OpenFlow messages to remove Flow Entries on those switches.

16. The CLC calculates the new path and sends the appropriate OpenFlow messages to add Flow Entries on those switches.

17. If the message is IPv6 the CLC checks if it comes from the DMM Gateway or from the MN.

18. If it comes from the DMM Gateway the CLC obtains from its BCE the AP where the MN is attached.

19. The CLC calculates the path from the DMM Gateway to the AP and installs the appropriate Flow Entries on those switches.

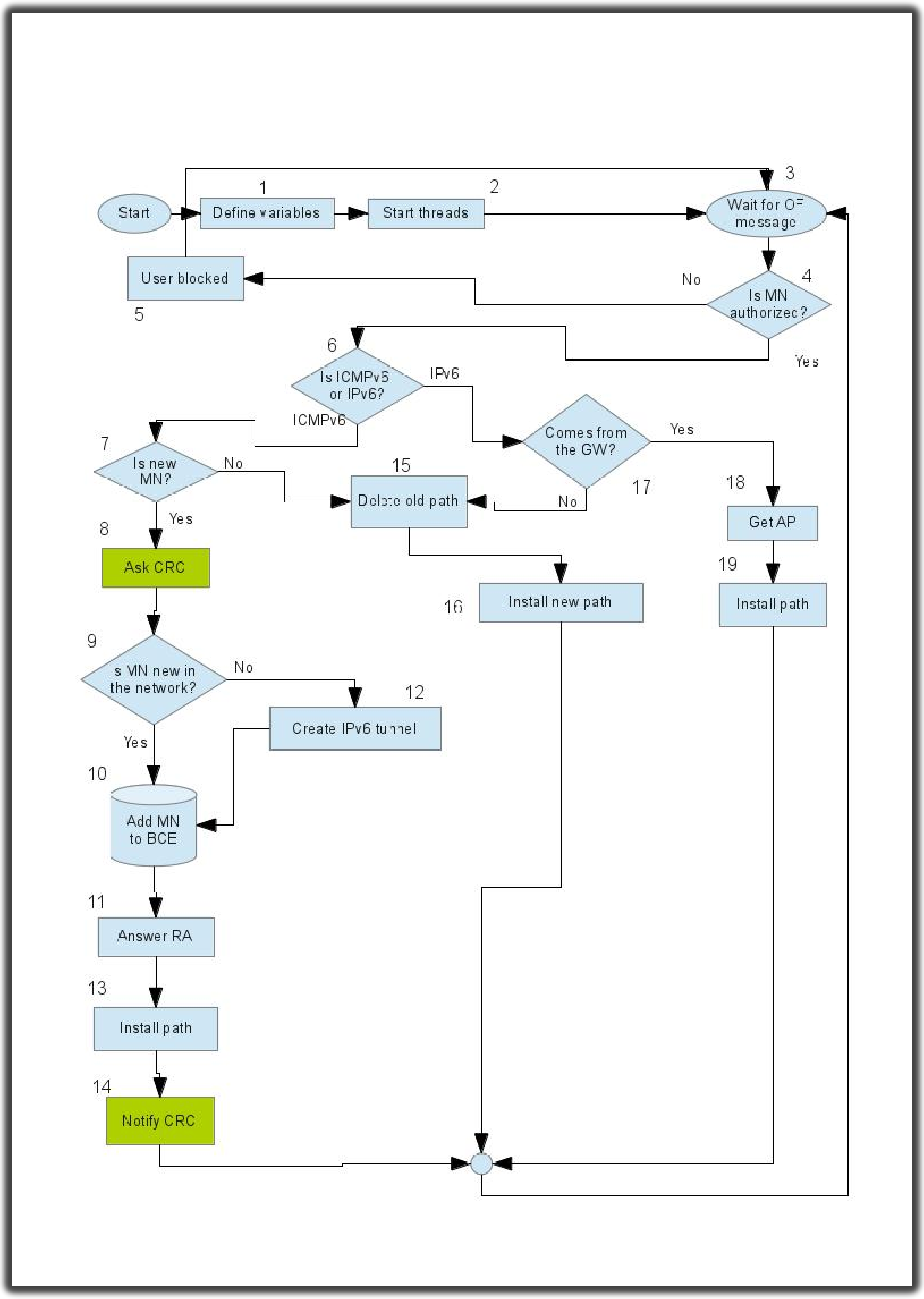


Figure 15: CLC OF listener's flow diagram

**The UDP listener:**

This thread is launched at the beginning of the OpenFlow listener execution. Its function is to create and destroy IPv6 tunnels between different districts when an inter-district handover happens. Its operation is tobe in a continuous wait state for an incoming message from the CRC. These messages contain parameters to modify the Gateways configuration. It can be an “Add” message containing the values of a remote Gateway so the CLC can create an IPv6 tunnel between the remote Gateway and its own one, or a delete message with the values to delete a tunnel.

The operation described in [Figure 16: CLC UDP listener's flow diagram]:

1. The global variables of the CLC are defined such as the ACL, CLC, Gateway parameters or dictionaries.

2. The second thread is launched.

3. The CLC binds a TCP connection with the CRC.

4.The CLC gets into listening mode waiting for incoming TCP messages from the CRC until one arrives.

5. When a TCP message arrives the CLC unpacks it and checks if the order is to "add" or "delete" an IPv6 tunnel.

6. If the order is to "delete" the CLC checks now if it has to delete it as the origin or the destiny of the tunnel. The difference lies in the source routing process, in where the route can be “from” or “to”.

7. If the order is to "delete origin" the CLC generates an UDP message with the orders to delete the IPv6 over IPv6 tunnel from its origin.

8. If the order is to "delete destiny" the CLC generates an UDP message with the orders to delete the IPv6 over IPv6 tunnel from its destiny.

9. As the MN is not attached now to the network the CLC deletes it from the BCE. In order to make the memory access atomicity, the CLC listener will lock a thread before modifying the BCE and release it after doing it.

10. The CLC sends the UDP message to the DMM Gateway.

11. If the message is to create a tunnel the CLC removes the current MN's path through its district. For that it sends a message to those switches with the order “Delete Flow” associated to the MN’s MAC address and a wildcard.

12. The CLC creates an UDP message with a string containing the necessary orders to create an IPv6 over IPv6 origin tunnel at the DMM gateway. The traffic redirection is done by source routing, that implies 3 steps: create the tunnel, add a default rule for all the traffic coming from the MN on a customized routing table, and an order in that table of forwarding all the MN’s traffic through the tunnel.

13.As the MN is not attached now to the network the CLC deletes it from the BCE. In order to make the memory access atomicity, the CLC listener will lock a thread before modifying the BCE and release it after doing it.

14. The CLC sends the UDP message created in boxes 7 or 8 to the DMM Gateway.

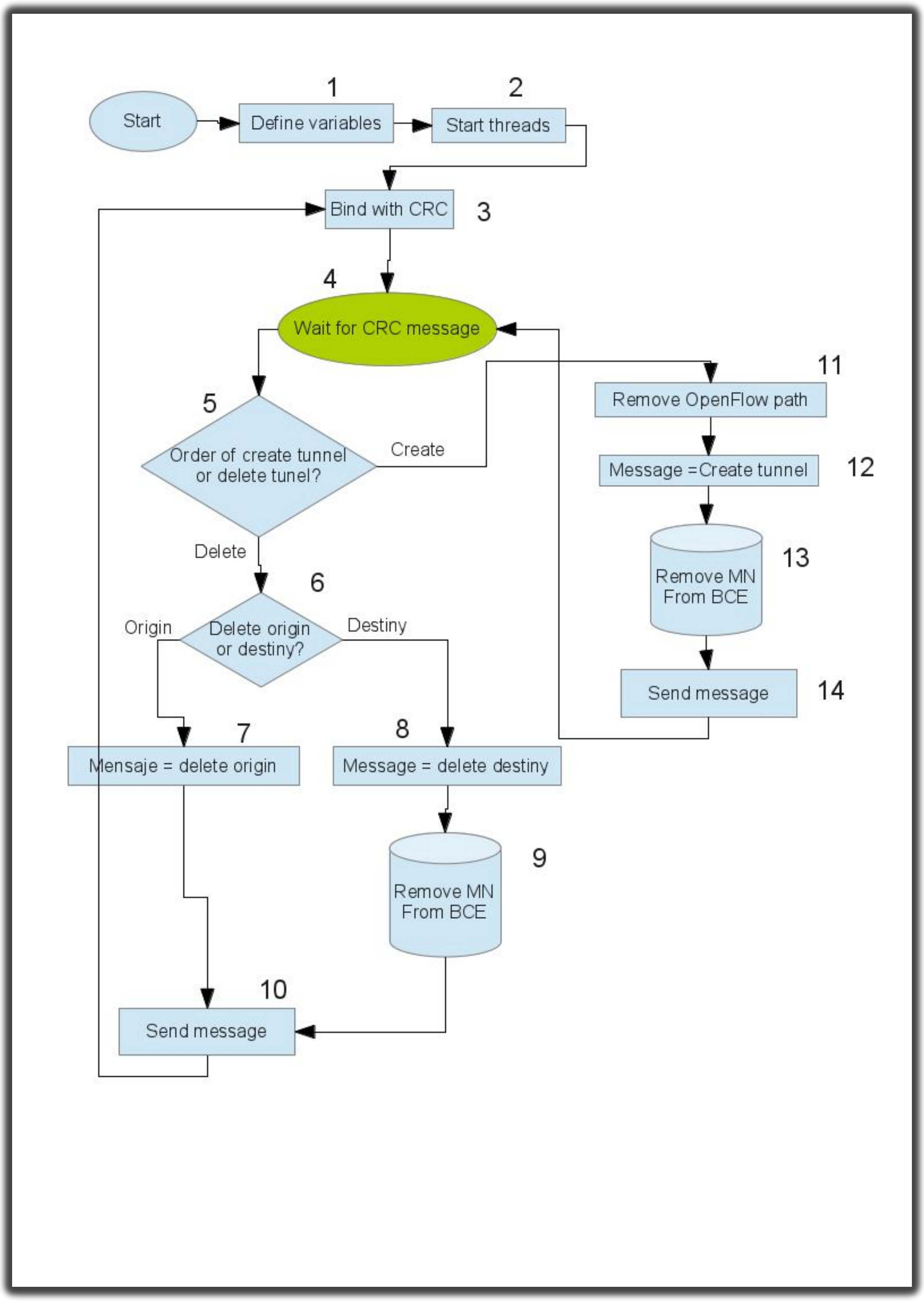


Figure 16: CLC UDP listener's flow diagram

### 5.5.2. THE REGIONAL CONTROLLER:

The regional controller is a TCP listener that reacts to two different notifications. The first one is the ask message. It happens when a CLC asks the CRC if an incoming MN comes from a handover or not. If it is not, the CRC adds it to its BCE and notifies the CLC with an “unknown MN” answer. If the MN comes from a handover, it modifies its BCE and sends the appropriate messages to the two CLC involved so they can create the IPv6 Tunnel between them and redirect the traffic. The other kind of message is the notification. It comes when a CLC has ended the attachment process with a MN. The CLC sends this message to the CRC in order to notify the new connection parameters.

The operation described in [Figure 17: CRC's flow diagram]:

1. The global variables of the CRC are defined: TheCLC's TCP and UDP connection parameters, the MN's Cache and the IPv6-tunnel cache. It also starts listening on a TCP port connection.

2.The CLC gets into listening mode waiting for incoming OpenFlow messages from the switches until one arrives, when that happens it unpacks the message.

3. When a message arrives the CRC checks if the message is type "ask" or type "notify". The ask message indicates an unknown attachment at a district and is a request for the MN information (prefix, gateway) stored by the CRC. The notify message indicates the end of an attachment and reports the MN’s assigned values to the CRC.

4. If the message is a "notify" (an update), the CRC corrects the attachment of the MN’s new district on its MN cache and also the Gateway when it is necessary.

5. If the message is an "ask", the CRC checks if the MN is already on its MN cache or if the cache is empty.

6. If the cache is empty or the MN is not in it, the CRC creates with an "unknown MN" type message, which contains only the “unknown MN” identifier character and empty data in all the other fields.

7. If the MN is already in the cache an inter-district handover is happening. So the CRC checks in the tunnel cache if there exists a tunnel for that MN whose origin (a gateway) matches with the gateway reported in the ask message. If it doesn’t it means that the MN is moving from its home district to another (Roaming). In the other hand, if the gateways match, it means that the MN is coming back from its home district (Regression).

8.If there is no regression, the CLC sends the appropriate messages to both (the old one and the new one) CLC's UDP listeners with the information necessary to create the IPv6 over IPv6 tunnel between the DMM Gateways of the two districts (remote gateway, tunnel label and MN IPv6 address). After that it adds the IPv6 tunnel to its cache.

9.Once the tunnel is created, the CRC creates the CLC with a "known MN" type message containing the information necessary to answer the Router solicitation. That is, the prefix of the old district to keep global reachability without any change of IPv6.

10. The CRC updates its cache with the new attachment of the MN.

11.If the MN is performing regression, the CRC sends the appropriate messages to both (the old one and the new one) CLC's UDP listeners with the information necessary to delete the IPv6 over IPv6 tunnel between the DMM Gateways of the two districts (remote gateway, tunnel label and MN IPv6 address).

12. The CRC pops the tunnel from its IPv6 tunnel cache.

13. The CRC creates the answer message to the CLC with a "known MN" type message containing the information necessary to answer the Router solicitation with the prefix of the old district to keep global reachability without any change of IPv6.

14.The CRC sends the answer message created in steps 6,9 or 13 to the CLCs via TCP protocol.

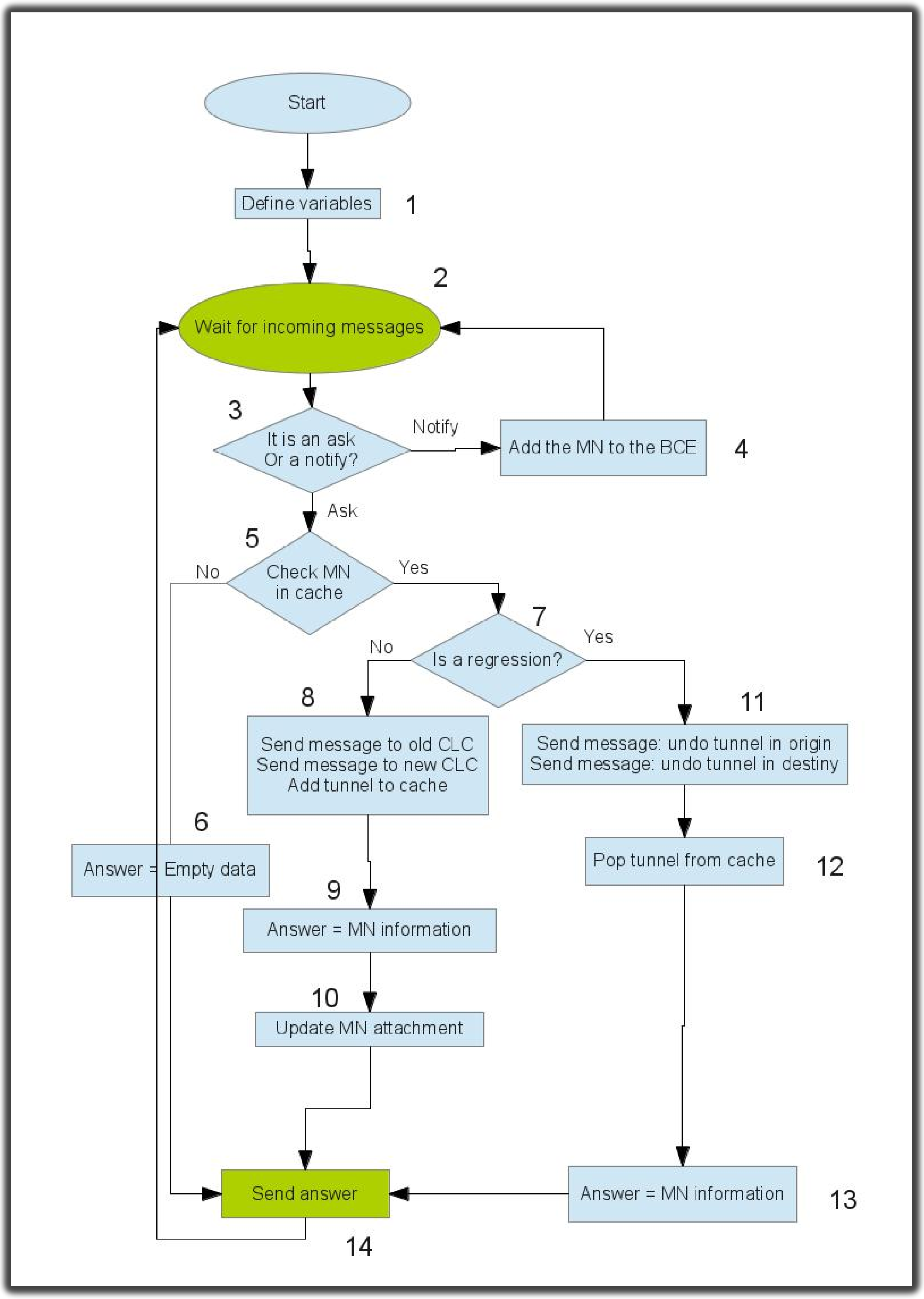


Figure 17: CRC's flow diagram

# 6. EXECUTION

## 6.1. PERFORMANCE ANALYSIS:

We successfully developed the test bed and performed normal PMIPv6 connections with the Internet through the OpenFlow network. We proceed now to measure and evaluate the results.

## 6.1. MEASUREMENTS

In order to test the efficiency of the solution we have measured some of the performance parameters. These measuresare taken to assess the solution and are focused to the three goals developed. The results are detailed in the following way:

### Attachment

We had to measure the attachment timing to the network in order to evaluate the Controller's ability to attach a host to the network and establish the routing path through it. The measures were obtained clocking the time taken at the MN between the Authentication Request sent and the Router Advertisement received. The breakout is detailed in the Annex section [Annex 1: Attachment measures]. The statistical results are the following[Figure 15: Attachment measures CDF]:

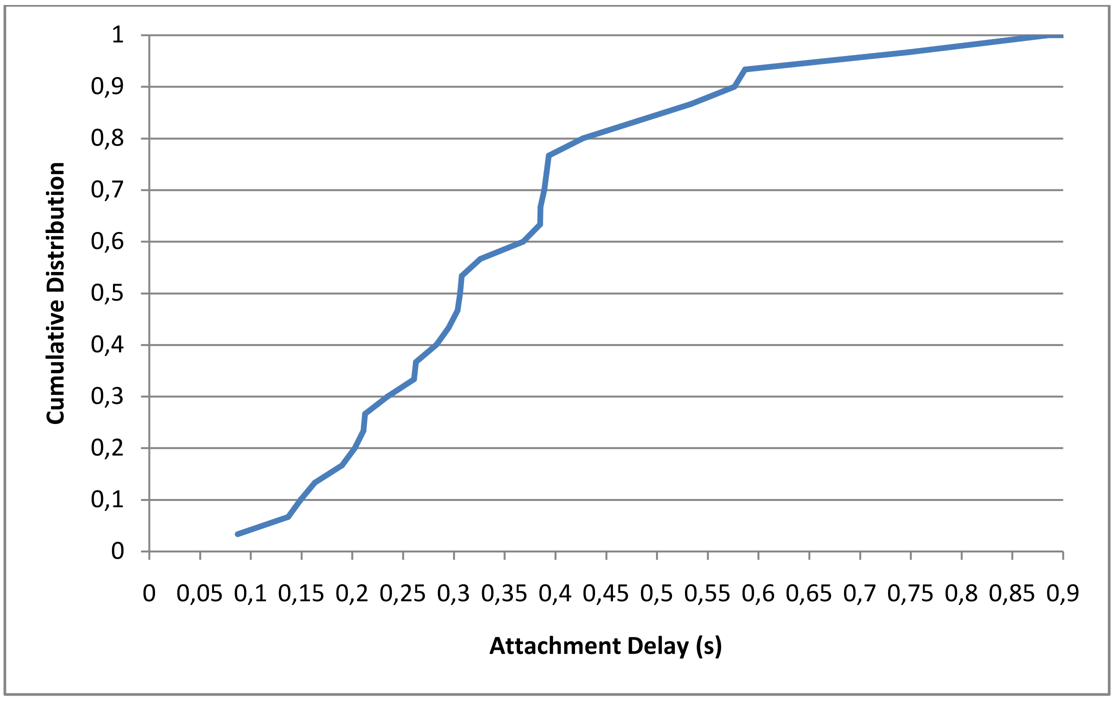


Figure 18: Attachment measures CDF

| | |
|---|---|
| Mean of the process (s) | 0,349353033 |
| Variance | 0,032268278 |

## Intra-district handover

To measure the timing taken during an intra-district handover we have send a flow of echo packets through the OpenFlow network (from a district's AP to its DMM Gateway), then we performed an intra-district handover and counted the packets lost. With that number and the timing between packets we can estimate how much time takes the system to recover connection during a handover. The breakout is detailed in the Annex section [Annex 1: Intra-district measures]. The statistical results are the following [Figure 16: Intra-district measures CDF]:

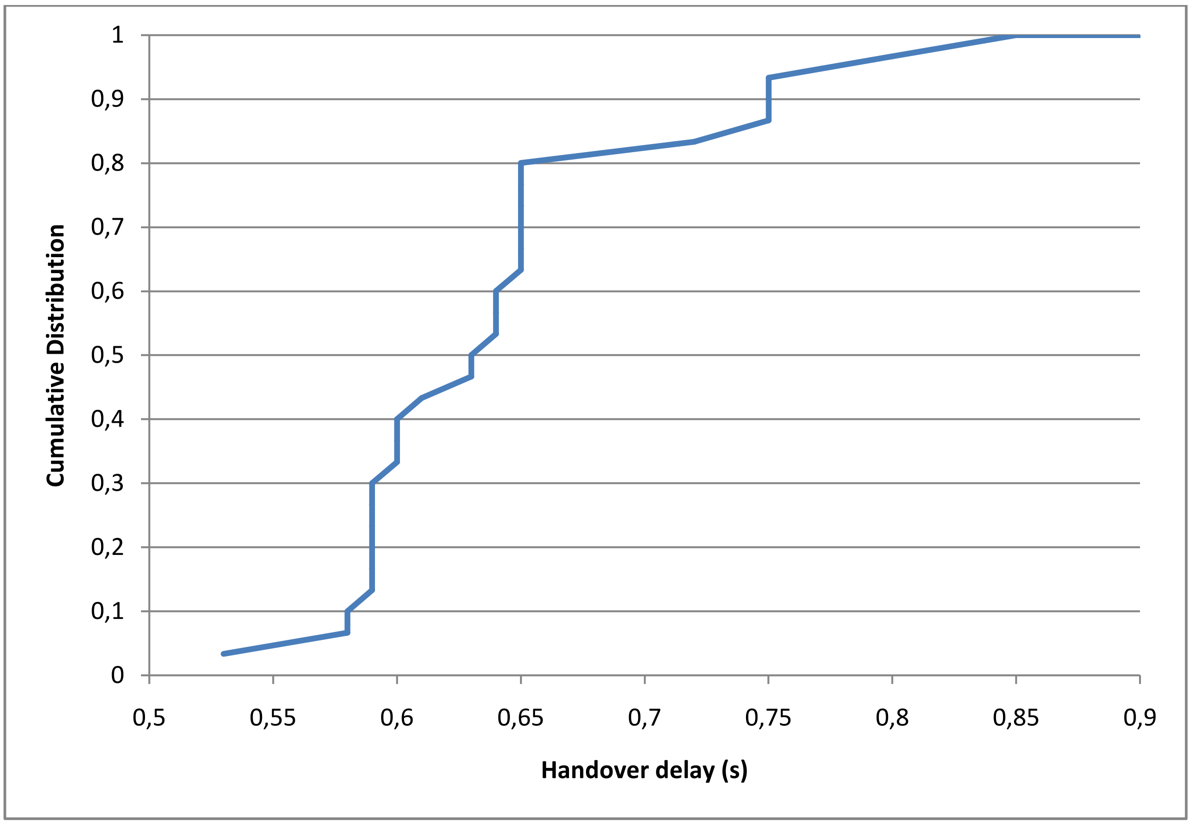


Figure 19: Intra-district measuresCDF

| | |
|---|---|
| Mean of the process (s) | 0,64466667 |
| Variance | 0,00526023 |

## Inter-district handover

To measure the timing taken during an inter-district handover we have send a flow of echo packets through the OpenFlow network (from a district's AP to its DMM Gateway), then we performed an inter-district handover and counted the packets lost. With that number and the timing between packets we can estimate how much time takes the system to recover connection during a handover. The breakout is detailed in the Annex section [Annex 2: Inter-district measures]. The timing is expected to be higher than the taken by the intra-district handovers due to the delay imposed by the IPv6 tunnel and the TCP communications between the CLCs and CRC. The statistical results are the following [Figure 17: Inter-district measures CDF]:

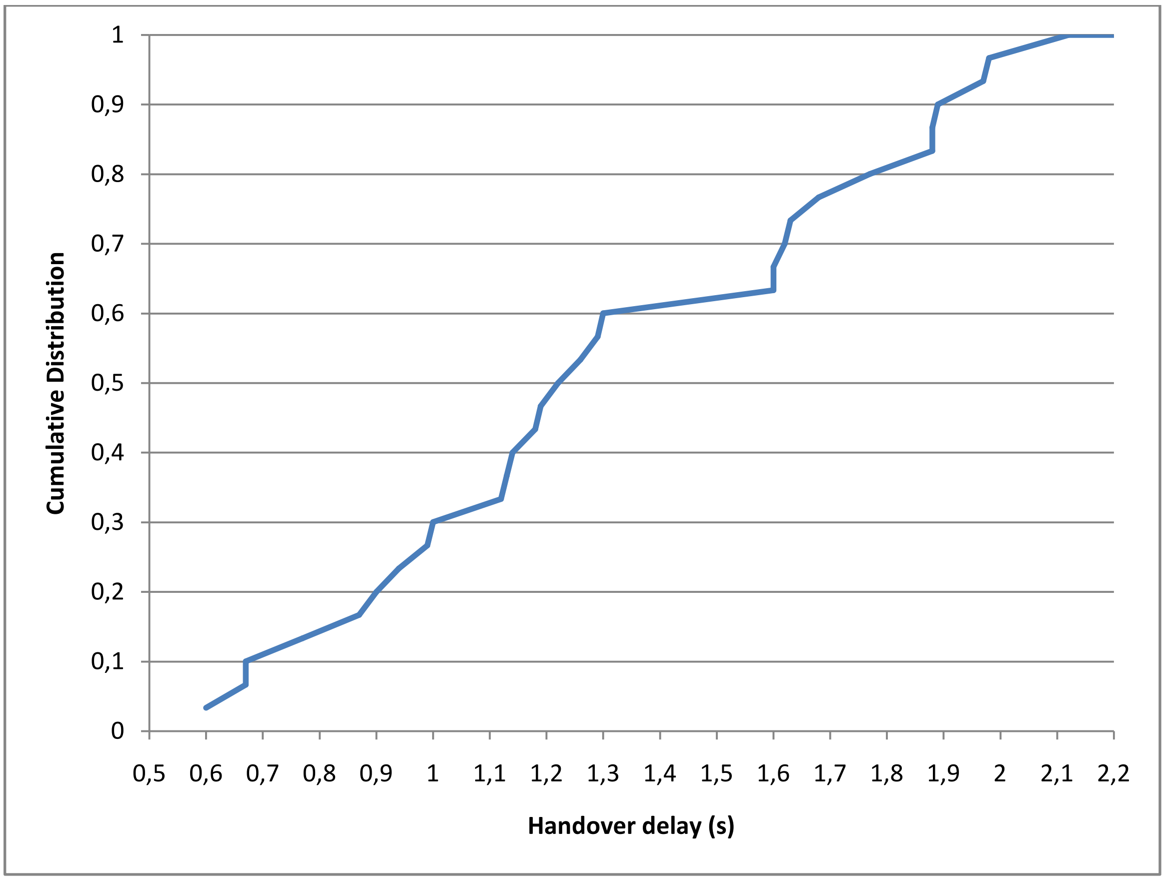


Figure 20: Inter-district measures CDF

| | |
|---|---|
| Mean of the process (s) | 1,328666667 |
| Variance | 0,195977471 |

## Throughput

We have measured the throughput offered by the OpenFlow network using Iperf command. We started a connection between the MN attached to the OpenFlow network and the other host through the internet. We launched the Iperf server at the host and the client at the MN. The test was performed on TCP and UDP. In the following graphic [Annex 2: Throughput CPF] the statistical results are the following [Figure 18: Throughput without IPv6 tunnel CDF] contains the CDF when the measures were taken without the IPv6 tunnel and [Figure 19: Throughput with IPv6 tunnel CDF] contains the CDF of TCP (Blue) and (UDP) when there was an IPv6 tunnel:

**Without the IPv6 tunnel**

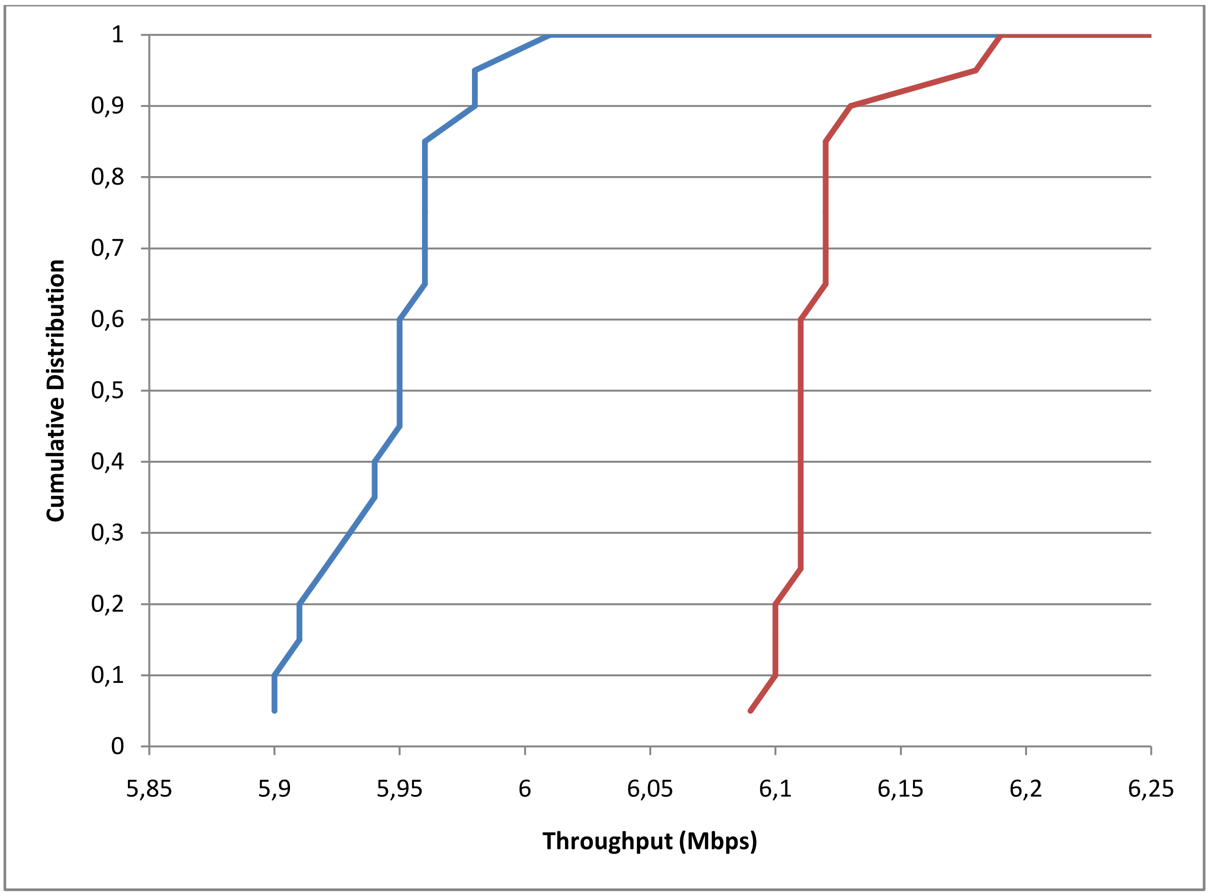


Figure 21: Throughput without IPv6 tunnel CDF

| TCP | |
|---|---|
| Mean of the process(Mbps) | 5,946 |
| Variance | 0,000804211 |

| UDP | |
|---|---|
| Mean of the process(Mbps) | 6,1185 |
| Variance | 0,000602895 |

**With the TCP tunnel**

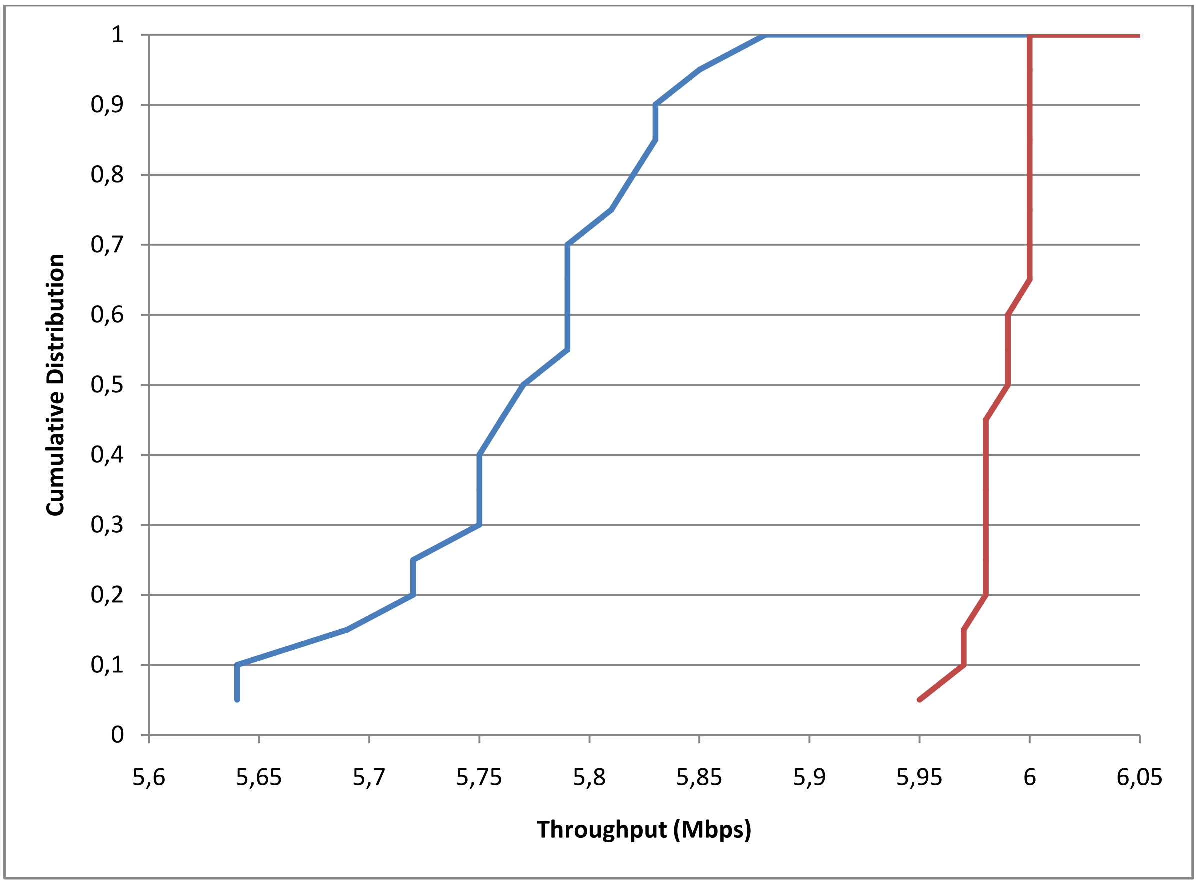


Figure 22: Throughput with IPv6 tunnel CDF

| TCP | |
|---|---|
| Mean of the process(Mbps) | 5,7685 |
| Variance | 0,004097632 |

| UDP | |
|---|---|
| Mean of the process(Mbps) | 5,987 |
| Variance | 0,000190526 |

## 7. PROBLEMS

In this section we list the different problems identified while implementing the solution:

**1. Memory space of the WRT54GL**: As we chose Linksys WRT54GL as the device to turn into an OpenFlow switch, we found its specifications quite short for the tests. For example: A DMM Gateway with the Backfire image and all the necessary packagesinstalled (IPv6, IP-tunnel etc.) has no available memory to install the TCPDUMP package. This package is not mandatory for the network, but it is very useful to monitoring the traffic through that device. Some time we have been forced to add some additional devices to perform packet sniffing and guest what was happening in the network when it didn't work as expected.

**2. Ryu's Ofproto version:** The first idea of this project was to develop an OpenFlow network whose switches perform IPv6 matching. That can be done with an OpenFlow 1.3 version or higher. Ryu provides support for this 1.3 version, but we didn't manage to run it at all. So, as we couldn't perform IPv4 matching, we decided to use 1.0 and modify the code to obtain MAC addresses from IPv6 link locals and perform a combination of MAC matching and input port Matching. On the switches, the original idea was to create a customized WRT54-GL image with OpenFlow 1.3 version but when we couldn't run it on the controller we decided to use the 1.0 image provided by OpenFlow to the WRT54GL, the Pantou.

**3. TCP:** We first tried to establish communications between the CRC and CLC using a secure TCP protocol. When the two programs were created, we tried to reduce the timing of any message exchange so we decided to switch TCP for UDP to reduce message exchanging.

**3. UDP listener:** As the network was developed and tested modularly, a problem appeared when the communication between the CLC and the CRC was added to the code. This communication was planned to happen by an exchange of UDP messages. The problem appeared when we tried to listen to these UDP message in the CLC listener method. When the CLC sends a message that must be answered by the CRC it stopped its execution until the answer arrived. This was unacceptable. So we first developed a second program with the function of listening to the UDP messages from the CRC and perform the necessary actions on the Gateway and that solved the problem. But then another issue appeared. When a Mobile Node leaves a district, it must be removed from the BCE of that district's CLC's. The BCE is part of the code of the CLC so the second program was unable to access to that element. The final solution was to unify both programs in one. This was done by launching a second thread at the beginning of the execution of the controller. The problem was solved.

**4. SSH timing:** The original idea of the DMM Gateway dynamic configuration by the CLCs was to perform it via SSH. The Controller performed SSH connections to the DMM Gateway and then launched the appropriate Unix commands to configure it. To achieve this we had to install open keys on the DMM Gateway in order to make the full process automatic. The problem appeared when we performed the first intra-district handover of all. The process takes from eight to nine seconds, but almost all of this time was expended on the SSH authentication performed on the DMM Gateway. The solution adopted for this was to change the SSH connection for an UDP listener on the DMM Gateway, always open to messages from the CLC. For this we only had to remove the SSH command from the subprocess (python package) and let the other commands in the same way. For the DMM Gateway we had to cross-compile a C program and upload it. This program is a listener on an UDP port, when it receives a message it just executes it as a UNIX command.

# 8. PROJECT PLANNIG

The project makes use of two main resources: a developer engineer and the director of the project. The engineer is in charge of the full development of the project and the Director is in charge of supervising and draws the lines of action. The time has been assigned to the engineer dividing the project's development in these twelvephases:

1. Study: We took the problem of mobility in dense networks and study the solutions proposed by the DMM Group and the definitions provided by CROWD. Then we studied SDN as a proposition to apply in the CROWD districts.

2. Analysis: We analyzed OpenFlow as a valid and tested SDN implementation. We studied various SDN controllers with OpenFlow support and chose Ryu.

3. Building: We build the first phase network with the four Linksys, the controller and the mobile node.

4. Phase 1: We developed a python IPv4 CROWD Local Controller to be run over Ryu.

5. Phase 1 test: We tested the IPv4 controller mobility in de district built (IPv4 intra-district mobility).

6. Phase 2: We extend the controller with IPv6 management.

7. Phase 2 test: We tested the IPv6 implementation and intra-district IPv6 mobility.

8. Extension: We expanded the network duplicating the district and adding an element to have the function of a CROWD Regional Controller.

9. Phase 3: We developed the software of the CROWD regional controller and adapted de CROWD Local Controllers with TCP communication with the CRC.

10. Phase 3 test:We tested the full implementation and we performed handovers inter and intra-district to until we validate the operation of all the elements.

11. Problem solutions: We found some performance problems.

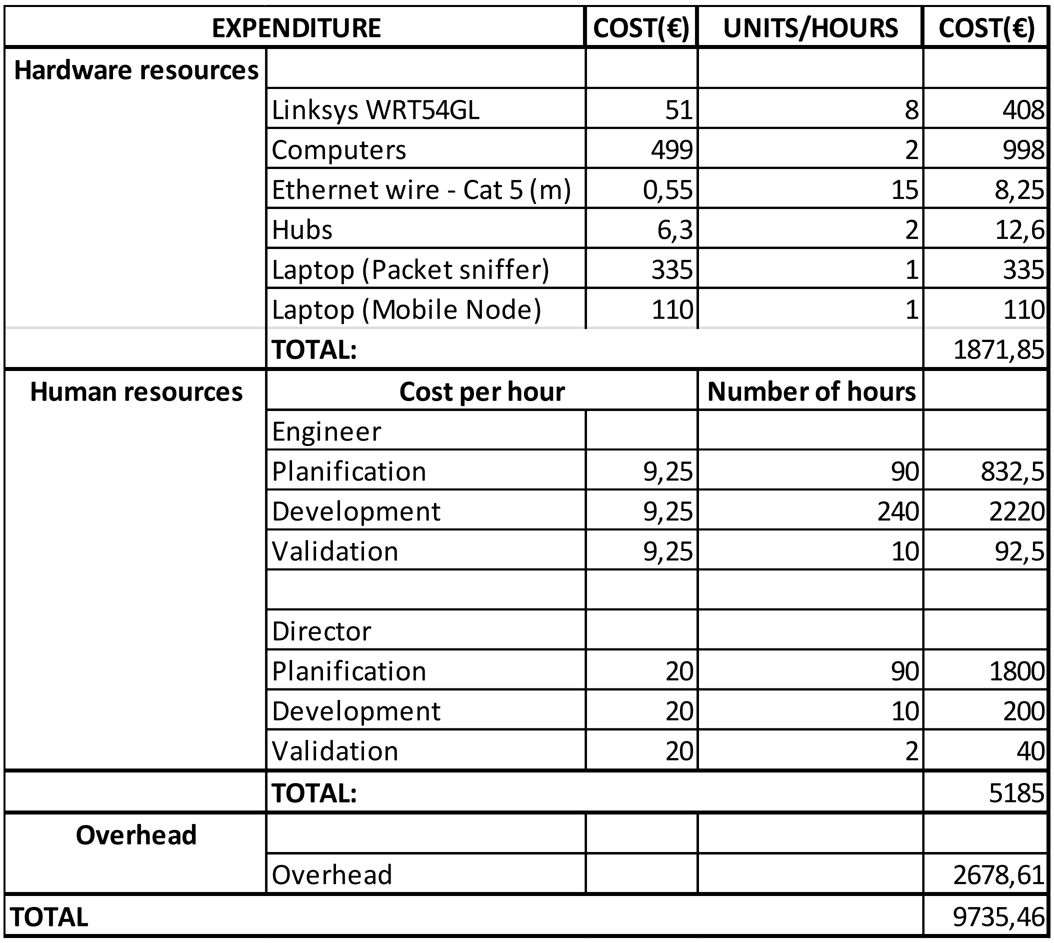
12. Validations: We evaluated the solution.

The critic tasks are: Phase 1 due to the importance of adaptation to the OpenFlow operation, Phase 2 due to the complications derived of the IPv6 adaptation and Test 3 due to the bigger dimension of the network and the possible complications of the communication between the CLCs and CRC. These phases may be extended if they aren't surpassed in time so we have assigned them some extra time.

| Month/Week | | Study | Analysis | Build | Phase 1 | Test 1 | Phase 2 | Test 2 | Extension | Phase 3 | Test 3 | Problems | Validation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MAY | 1 | 🟦 | | | | | | | | | | | |
| | 2 | 🟦 | | | | | | | | | | | |
| | 3 | 🟦 | | | | | | | | | | | |
| | 4 | 🟦 | | | | | | | | | | | |
| JUN | 1 | | 🟦 | | | | | | | | | | |
| | 2 | | 🟦 | | | | | | | | | | |
| | 3 | | 🟦 | | | | | | | | | | |
| | 4 | | 🟦 | | | | | | | | | | |
| JULY | 1 | | 🟦 | | | | | | | | | | |
| | 2 | | | 🟦 | | | | | | | | | |
| | 3 | | | 🟦 | | | | | | | | | |
| | 4 | | | 🟦 | | | | | | | | | |
| AUGUST | 1 | | | | ■ | | | | | | | | |
| | 2 | | | | ■ | | | | | | | | |
| | 3 | | | | ■ | 🟦 | | | | | | | |
| | 4 | | | | | 🟦 | | | | | | | |
| SEPTEMBER | 1 | | | | | | ■ | | | | | | |
| | 2 | | | | | | ■ | | | | | | |
| | 3 | | | | | | ■ | | | | | | |
| | 4 | | | | | | ■ | 🟦 | | | | | |
| OCTOBER | 1 | | | | | | ■ | 🟦 | | | | | |
| | 2 | | | | | | | 🟦 | | | | | |
| | 3 | | | | | | | | 🟦 | | | | |
| | 4 | | | | | | | | | 🟦 | | | |
| NOVEMBER | 1 | | | | | | | | | 🟦 | | | |
| | 2 | | | | | | | | | 🟦 | ■ | | |
| | 3 | | | | | | | | | | ■ | | |
| | 4 | | | | | | | | | | ■ | | |
| DECEMBER | 1 | | | | | | | | | | ■ | | |
| | 2 | | | | | | | | | | ■ | | |
| | 3 | | | | | | | | | | ■ | | |
| | 4 | | | | | | | | | | ■ | | |
| JANUARY | 1 | | | | | | | | | | ■ | | |
| | 2 | | | | | | | | | | | 🟦 | |
| | 3 | | | | | | | | | | | 🟦 | |
| | 4 | | | | | | | | | | | 🟦 | |
| FEBRUARY | 1 | | | | | | | | | | | 🟦 | |
| | 2 | | | | | | | | | | | | 🟦 |
| | 3 | | | | | | | | | | | | 🟦 |
| | 4 | | | | | | | | | | | | |

Figure 23: Project's Gantt diagram

The costs of the project have been expressed in hardware resources and human resources. The hardware resources prices are founded at amazon[18], for the Engineer and Director's cost per hour we have averaged market values from the telecommunications sector. The cost breakdown of the project is the following:



| EXPENDITURE | | COST(€) | UNITS/HOURS | COST(€) |
|---|---|---|---|---|
| **Hardware resources** | | | | |
| | Linksys WRT54GL | 51 | 8 | 408 |
| | Computers | 499 | 2 | 998 |
| | Ethernet wire - Cat 5 (m) | 0,55 | 15 | 8,25 |
| | Hubs | 6,3 | 2 | 12,6 |
| | Laptop (Packet sniffer) | 335 | 1 | 335 |
| | Laptop (Mobile Node) | 110 | 1 | 110 |
| | **TOTAL:** | | | 1871,85 |
| **Human resources** | **Cost per hour** | | **Number of hours** | |
| | Engineer | | | |
| | Planification | 9,25 | 90 | 832,5 |
| | Development | 9,25 | 240 | 2220 |
| | Validation | 9,25 | 10 | 92,5 |
| | | | | |
| | Director | | | |
| | Planification | 20 | 90 | 1800 |
| | Development | 20 | 10 | 200 |
| | Validation | 20 | 2 | 40 |
| | **TOTAL:** | | | 5185 |
| **Overhead** | | | | |
| | Overhead | | | 2678,61 |
| **TOTAL** | | | | 9735,46 |

Figure 24: Project's accounting breakdown

We have researched about the cost that an SDN development may have in order to place the costs of ours in the within a framework. We found some information at an article[23] about commercial SDN developments in 2013. The article describes a full SDN building but as we only developed the software we won't consider services, security or training. In that framework the article places the SDN development between 7280 and 36340 euros (10000-50000 $). In that spectrum we can consider our project one of the cheapest extreme because our project didn't reached the 10000 euros including the hardware resources and the overhead.

[18]www.amazon.com

# 9. CONCLUSIONS AND ASSESSMENT

We have studied which is the dense network mobility problem and what solutions are being proposed. We have analyzed the current protocols and what alternatives are being developed.We have chosen one of the DMM working group solutions which consist on developing a network with control plane distribution reusing the existing IPv6 extension such as MIPv6 and PMIPv6. We have developed a full implementation of an SDN network system: the hardware and the software, all in the framework of the dense network problem. That is traduced in two network elements (the CLC and the CRC) and a network test bed. With all that we have evaluated the ease and flexibility of an SDN implementation.

The results were not as high as expected and we have analyzed the reasons for this. The code can be rewritten in a more efficient way by saving some timing switching some TCP communications for UDP ones and improving the message exchange between the network elements. Moreover, the full architecture showed very hardware-dependent and some elements showed bottlenecks when they had to handle big data flows. Also, the OpenFlow version used was the 1.0 and the 1.3 has been released now and it may produce faster and better performance than 1.0. So there is still much room for improvement and a wide space to optimize the software developed.

The results of this project will help the CROWD Project as feedback, and it also will contribute to the development of the 802.1cf IEEE standard.

## 9.1. SKILLS DEVELOPED

From the beginning of the project to its end, we have developed some new skills as engineer and reinforced concepts on others acquired during the past few years. We have researched and learned about some of the most innovative trends in telecommunications such as SDN or DMM, and we also had to develop the software in python so we learned some notions of python programming. Moreover, we have applied some technologies that we already knew such as MIPv6, PMIPv6, UDP and TCP communication and system threads. In addition, the entire project has been developed over Linux so we increased our knowledge in Shell. Considering all this, we think has been a big addition to the knowledge acquired during the degree.

# 10. ANNEX 1: Measurements

Table 1: Attachment measures

| Test | Authentication Request | Router Advertisement | Blackout (s) |
|---|---|---|---|
| 1 | Feb 20, 2014 17:13:43.506166000 | Feb 20, 2014 17:13:43.986924000 | 0,480758 |
| 2 | Feb 20, 2014 17:24:29.449168000 | Feb 20, 2014 17:24:29.536321000 | 0,087153 |
| 3 | Feb 20, 2014 17:27:04.485332000 | Feb 20, 2014 17:27:04.878686000 | 0,393354 |
| 4 | Feb 20, 2014 17:28:05.520791000 | Feb 20, 2014 17:28:05.888961000 | 0,36817 |
| 5 | Feb 20, 2014 17:29:51.503938000 | Feb 20, 2014 17:29:51.738390000 | 0,234452 |
| 6 | Feb 20, 2014 17:31:19.768288000 | Feb 20, 2014 17:31:20.062959000 | 0,294671 |
| 7 | Feb 20, 2014 17:32:12.709318000 | Feb 20, 2014 17:32:12.899180000 | 0,189862 |
| 8 | Feb 20, 2014 17:33:11.693106000 | Feb 20, 2014 17:33:11.829800000 | 0,136694 |
| 9 | Feb 20, 2014 17:34:09.241664000 | Feb 20, 2014 17:34:09.390708000 | 0,149044 |
| 10 | Feb 20, 2014 17:34:56.857417000 | Feb 20, 2014 17:34:57.059815000 | 0,202398 |
| 11 | Feb 20, 2014 17:37:23.495038000 | Feb 20, 2014 17:37:23.707556000 | 0,212518 |
| 12 | Feb 20, 2014 17:38:51.149910000 | Feb 20, 2014 17:38:51.410630000 | 0,26072 |
| 13 | Feb 20, 2014 17:39:41.223944000 | Feb 20, 2014 17:39:41.810565000 | 0,586621 |
| 14 | Feb 20, 2014 17:40:35.496079000 | Feb 20, 2014 17:40:36.072315000 | 0,576236 |
| 15 | Feb 20, 2014 17:41:28.540749000 | Feb 20, 2014 17:41:28.925769000 | 0,38502 |
| 16 | Feb 20, 2014 17:42:04.379887000 | Feb 20, 2014 17:42:04.590902000 | 0,211015 |
| 17 | Feb 20, 2014 17:43:15.445765000 | Feb 20, 2014 17:43:15.979206000 | 0,533441 |
| 18 | Feb 20, 2014 17:46:05.635654000 | Feb 20, 2014 17:46:06.383687000 | 0,748033 |
| 19 | Feb 20, 2014 17:47:06.666244000 | Feb 20, 2014 17:47:06.928884000 | 0,26264 |
| 20 | Feb 20, 2014 17:47:40.663142000 | Feb 20, 2014 17:47:40.966639000 | 0,303497 |
| 21 | Feb 20, 2014 17:48:59.409150000 | Feb 20, 2014 17:49:00.294848000 | 0,885698 |
| 22 | Feb 20, 2014 17:49:46.923173000 | Feb 20, 2014 17:49:47.350388000 | 0,427215 |
| 23 | Feb 20, 2014 17:50:34.847925000 | Feb 20, 2014 17:50:35.154106000 | 0,306181 |
| 24 | Feb 20, 2014 17:51:59.122170000 | Feb 20, 2014 17:51:59.511316000 | 0,389146 |
| 25 | Feb 20, 2014 17:52:31.890312000 | Feb 20, 2014 17:52:32.216312000 | 0,326 |
| 26 | Feb 20, 2014 17:53:22.373949000 | Feb 20, 2014 17:53:22.759154000 | 0,385205 |
| 27 | Feb 20, 2014 17:53:58.009260000 | Feb 20, 2014 17:53:58.316898000 | 0,307638 |
| 28 | Feb 20, 2014 17:55:18.291389000 | Feb 20, 2014 17:55:18.574335000 | 0,282946 |
| 29 | Feb 20, 2014 17:56:59.053440000 | Feb 20, 2014 17:56:59.216346000 | 0,162906 |
| 30 | Feb 20, 2014 17:57:39.501702000 | Feb 20, 2014 17:57:39.893061000 | 0,391359 |

Table 2: Intra-district measures

| Test | Timing (s) | Packets missed | Blackout time (s) |
|---|---|---|---|
| **1** | 0,01 | 53 | 0,53 |
| **2** | 0,01 | 59 | 0,59 |
| **3** | 0,01 | 58 | 0,58 |
| **4** | 0,01 | 60 | 0,6 |
| **5** | 0,01 | 65 | 0,65 |
| **6** | 0,01 | 65 | 0,65 |
| **7** | 0,01 | 75 | 0,75 |
| **8** | 0,01 | 61 | 0,61 |
| **9** | 0,01 | 75 | 0,75 |
| **10** | 0,01 | 85 | 0,85 |
| **11** | 0,01 | 65 | 0,65 |
| **12** | 0,01 | 63 | 0,63 |
| **13** | 0,01 | 58 | 0,58 |
| **14** | 0,01 | 65 | 0,65 |
| **15** | 0,01 | 60 | 0,6 |
| **16** | 0,01 | 63 | 0,63 |
| **17** | 0,01 | 59 | 0,59 |
| **18** | 0,01 | 64 | 0,64 |
| **19** | 0,01 | 59 | 0,59 |
| **20** | 0,01 | 59 | 0,59 |
| **21** | 0,01 | 59 | 0,59 |
| **22** | 0,01 | 72 | 0,72 |
| **23** | 0,01 | 60 | 0,6 |
| **24** | 0,01 | 65 | 0,65 |
| **25** | 0,01 | 80 | 0,8 |
| **26** | 0,01 | 64 | 0,64 |
| **27** | 0,01 | 59 | 0,59 |
| **28** | 0,01 | 65 | 0,65 |
| **29** | 0,01 | 64 | 0,64 |
| **30** | 0,01 | 75 | 0,75 |

Table 3: Inter-district measures

| Test | Timing (s) | Packets missed | Blackout time (s) |
|---|---|---|---|
| **1** | 0,01 | 67 | 0,67 |
| **2** | 0,01 | 94 | 0,94 |
| **3** | 0,01 | 198 | 1,98 |
| **4** | 0,01 | 77 | 0,77 |
| **5** | 0,01 | 177 | 1,77 |
| **6** | 0,01 | 60 | 0,6 |
| **7** | 0,01 | 119 | 1,19 |
| **8** | 0,01 | 162 | 1,62 |
| **9** | 0,01 | 90 | 0,9 |
| **10** | 0,01 | 67 | 0,67 |
| **11** | 0,01 | 87 | 0,87 |
| **12** | 0,01 | 188 | 1,88 |
| **13** | 0,01 | 189 | 1,89 |
| **14** | 0,01 | 197 | 1,97 |
| **15** | 0,01 | 130 | 1,3 |
| **16** | 0,01 | 160 | 1,6 |
| **17** | 0,01 | 168 | 1,68 |
| **18** | 0,01 | 212 | 2,12 |
| **19** | 0,01 | 163 | 1,63 |
| **20** | 0,01 | 129 | 1,29 |
| **21** | 0,01 | 188 | 1,88 |
| **22** | 0,01 | 114 | 1,14 |
| **23** | 0,01 | 160 | 1,6 |
| **24** | 0,01 | 100 | 1 |
| **25** | 0,01 | 113 | 1,13 |
| **26** | 0,01 | 122 | 1,22 |
| **27** | 0,01 | 126 | 1,26 |
| **28** | 0,01 | 99 | 0,99 |
| **29** | 0,01 | 118 | 1,18 |
| **30** | 0,01 | 112 | 1,12 |

Table 4: Network throughput (Mbits/s)

| **Test** | **Without IPv6 tunnel** | | **With IPv6 tunnel** | |
|---|---|---|---|---|
| | **UDP (Mbps)** | **TCP (Mbps)** | **UDP (Mbps)** | **TCP (Mbps)** |
| **1** | 6,12 | 5,95 | 5,98 | 5,83 |
| **2** | 6,19 | 5,9 | 6 | 5,72 |
| **3** | 6,18 | 5,93 | 6 | 5,79 |
| **4** | 6,11 | 5,91 | 5,97 | 5,79 |
| **5** | 6,12 | 5,96 | 5,98 | 5,85 |
| **6** | 6,13 | 5,96 | 6 | 5,75 |
| **7** | 6,12 | 5,95 | 5,99 | 5,81 |
| **8** | 6,11 | 5,9 | 6 | 5,76 |
| **9** | 6,12 | 5,92 | 5,98 | 5,75 |
| **10** | 6,11 | 5,96 | 5,98 | 5,82 |
| **11** | 6,1 | 5,96 | 5,98 | 5,75 |
| **12** | 6,11 | 6,01 | 6 | 5,64 |
| **13** | 6,1 | 5,96 | 5,99 | 5,83 |
| **14** | 6,11 | 5,98 | 6 | 5,79 |
| **15** | 6,09 | 5,95 | 5,98 | 5,72 |
| **16** | 6,1 | 5,95 | 6 | 5,88 |
| **17** | 6,11 | 5,94 | 6 | 5,79 |
| **18** | 6,11 | 5,98 | 5,97 | 5,64 |
| **19** | 6,12 | 5,91 | 5,99 | 5,77 |
| **20** | 6,11 | 5,94 | 5,95 | 5,69 |

# 11. REFERENCES

[1] Thomas D. Nadeau, Ken Gray, “SDN: Software Defined Networks”, O’Reilly Media - August 2013.

[2] “OpenFlow (Wikipedia): http://en.wikipedia.org/wiki/OpenFlow” 21 February 2014.

[3] “Software-Defined Networking (Wikipedia): http://en.wikipedia.org/wiki/Software-defined_networking” 21 February 2014.

[4] C. Perkins, Ed. Tellabs, Inc. D. Johnson, J. Arkko “Mobility Support in IPv6”, IETF RFC 6275 – July 2011: http://tools.ietf.org/html/rfc6275

[5] “Mobile IP (Wikipedia): http://en.wikipedia.org/wiki/Mobile_IP”February 21, 2014.

[6] S. Gundavelli, Ed. K. Leung, V. Devarapalli, K. Chowdhury, B. Patil “Proxy Mobile IPv6”, IETF RFC 5213 – August 2008: http://tools.ietf.org/search/rfc5213

[7] “Proxy Mobile IP (Wikipedia): http://en.wikipedia.org/wiki/Proxy_Mobile_IPv”February 21, 2014

[8] H. Chan “Requirements of distributed mobility management draft-ietf-dmm-requirements-01”, IETF Internet-Draft – January 13, 2013

[9] Hassan Ali-Ahmad, Claudio Cicconetti, Antonio de la Oliva, Martin Dräxler, Rohit Gupta, Vincenzo Mancuso, Laurent Roullet, Vincenzo Sciancalepore, “CROWD: An SDN Approach for DenseNets”http://www.ict-crowd.eu/

[10]Hassan Ali-Ahmad, Claudio Cicconetti, Antonio de la Oliva, Vincenzo Mancuso, Malla Reddy Sama, Pierrick Seite, Sivasothy Shanmugalingam “An SDN-based Network Architecture for Extremely Dense Wireless Networks: http://www.ict-crowd.eu/downloads/ali-ahmad13sdn.pdf” February 21  2014

[11]“OpenFlow: http://archive.openflow.org/” February 21 2014

[12]“Ryu component-based software defined networking framework: http://osrg.github.io/ryu/” February 21 2014

[13]“Ryu application API: http://ryu.readthedocs.org/en/latest/ryu_app_api.html” February 21 2014

[14]“Pantou: OpenFlow 1.0 for OpenWRT: http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT” February 21 2014

[15]“OpenDaylight: http://www.opendaylight.org/” February 21 2014

[16]“Open Daylight Project (Wikipedia): http://en.wikipedia.org/wiki/Open_Dailight_Project” February 21 2014

[17] “Getting involved in OmniRAN EC Study Group: http://www.ieee802.org/OmniRANsg/” February 21 2014

[18]“OpenFlow Whitepaper: http://archive.openflow.org/documents/openflow-wp-latest.pdf” February 21 2014

[19] "TCAM (Wikipedia): http://en.wikipedia.org/wiki/Content-addressable_memory#Ternary_CAMs " February 21 2014

[20] "CAM (Wikipedia): http://en.wikipedia.org/wiki/Content-addressable_memory " February 21 2014

[21]“Wireless MANs-WiMax: http://standards.ieee.org/about/get/802/802.16.html” February 21 2014

[22]“Top 5 items for your 2013 SDN budget: http://www.sdncentral.com/market/top-5-items-for-your-2013-sdn-budget/2012/08/” February 21 2014

[23]“Asymmetric Double-Agents Architecture for Fast Handoff and Efficient Routing: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5501871&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5501871” February 21 2014

[24]P. Bertin, Servane Bonjour, and J.-M. Bonnin. A distributed dynamic mobility management scheme designed for flat ip architectures. In New Technologies, Mobility and security, 2008. NTMS ’08, pages 1-5, nov. 2008. February 21 2014

[25]“Linksys WRT54G, WRT54GL, WRT54GS: http://wiki.openwrt.org/toh/linksys/wrt54g”. February 21 2014