Evaluation of OpenFlow Controllers

Guillermo Romero de Tejada Muntaner

October 15, 2012

Abstract

The main objective of this project has been to perform a study and evaluation of different types of OpenFlow controllers that exist today. Initially, has been performed a detailed study on the OpenFlow technology and different types of controllers that can be implemented (among other the NOX, Beacon, Trema, SNAC and Maestro Controllers). Then, have been designed two applications, using Beacon and Trema Controllers, both applications with the same purpose: redirect the Ping messages. Finally, have been evaluated these two applications designed and learning-Switch application of the following Controllers: NOX, Beacon, Trema and Maestro.

OpenFlow is a new technology based on the Software-Defined Networking (SDN) concept that consists in break with the conventional model of network where is the Switch that decides the actions that have to do. OpenFlow technology separates the Control Plane from the Data Path and this allows for network researchers to develop their own algorithms to control data flows and packets. OpenFlow technology implements the control logic on an external controller (typically an external PC) and this controller is responsible for deciding the actions that the Switch must perform. The communication between the controller and the Data Path is made, on the network itself, using the protocol that provides OpenFlow (OpenFlow Protocol). There is a non-profit organization called Open Network Foundation (ONF), organization that is responsible to control and publish the different OpenFlow specifications and will give the trademark license "OpenFlow Switching" to companies that adopt this standard.

Abstract

Det främsta syftet med detta projekt har varit att göra en studie och utvärdering av olika typer av OpenFlow styrenheter som finns idag. Inledningsvis, har genomfört en ingående undersökning av OpenFlow teknik och olika typer av regulatorer som kan genomföras (bland annat NOX, Beacon, Trema, SNAC-och Maestro-styrenheter). Sedan, har utformats två program med Beacon och styrenheter Trema, båda ansökningarna med samma syfte: omdirigera Ping-meddelanden. Slutligen, har utvärderat dessa två utformade program och lärande-Switch tillämpningen av följande styrenheter: NOX, Beacon, Trema och Maestro.

OpenFlow är en ny teknik baserad på Software Defined-Nätverk (SDN) begrepp som består i brytning med den traditionella modellen av nätverk där är Switch som avgör vilka åtgärder som har att göra. OpenFlow separerar Control planet från Data Path och detta gör det möjligt för nätverksadministratörer forskare att utveckla sina egna algoritmer för att styra dataflöden och paket. OpenFlow teknik implementerar styrlogiken på en extern styrenhet (typiskt en extern PC) och denna styrenhet är ansvarig för att besluta de åtgärder som Switch måste utföra. Kommunikationen mellan styrenheten och Data Path görs i nätverket själva, med hjälp av protokoll som tillhandahåller OpenFlow (OpenFlow Protocol). Det är en icke-vinstdrivande organisation som heter Open Network Foundation (ONF), organisation som ansvarar för att kontrollera och offentliggöra de olika OpenFlow specifikationer och kommer att ge varumärket licens "OpenFlow koppling" till företag som antar denna standard.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

At the present has started a evolution in the field of networking and data centers. This evolution has been caused by high cost for control of data centers and is spreading to other areas. One technology that has revolutionized the data centers is the virtualization technology. Virtualization technologies have enabled more efficient use of IT resources and greater levels of IT agility and control. In addition, together with virtualization technology have emerged new technologies. One of these new technologies is the "Cloud Computing". Both technologies, virtualization and "cloud computing" allow the companies to better meet the demand of resources and have more agility.

Unfortunately, these different technologies can not be supported today for enterprise networks for design reasons [7]. Corporate networks were not designed to support the virtualization or cloud computing techniques. It is for this reason that the progress of virtualization technology and "Cloud Computing" is limited by the state or the inefficiency of networks today. The result is that the network acts as a bottleneck, limiting the ability of the network of any company to adapt to these new technologies. After recognizing the problem, the networking community is working on an alternative to traditional networks. The Software-Defined Networking (SDN) [1] is a concept that is to break with the conventional model of network where is the network switch that decides the actions that have to do. The SDN concept is based on defining a model where all switches move the capacity of decision to a central element, to a controller.

OpenFlow is based on the Software-Defined Networking (SDN) concept [1]. OpenFlow is a new standard, a new open source protocol, designed by the Open Networking Foundation (ONF) [2] and with the aim to provide programmable networks. Typically, in traditional networks architecture, the data path and the decision-making process of switching or routing are collocated on the same device. OpenFlow defines a standard upon which these two concepts can be abstracted. This means that these two functions no longer have to occur on the same device. This allows control logic to be moved to an external controller (typically an external PC) and the controller communicates with the Data Path, over the network itself, using the OpenFlow protocol.

There are different types of OpenFlow Controllers. Depending on the programming language, there is the possibility to find Java-based controllers as Beacon [3] and Maestro [4] Controllers, controllers based on Ruby and C as the Trema controller [5] and finally con-

trollers based on C++ and Python as the NOX controller [6]. The main objective of the controller is to centralize the network intelligence, while the network maintains a distributed forwarding plane through OpenFlow Switches and routers. There are different Controller configurations depending on different parameters as for example the location or behavior.

The OpenFlow implementation provides multiple benefits including the possibility to have the ability to dynamically modify the network depending on the requirements, allowing the new rules propagation throughout the entire network and another benefit is to have a simplification of network management.

1.1 Goals

The goals of this master thesis can be described as follows:

- Investigate current network virtualization technologies.
- Study and design of a controller using OpenFlow.
- The evaluation, from the performance point of view, of the different controllers that have been designed and the learning-Switch application.

1.2 Thesis outline

This work is organized as follows:

- *Chapter 2* Background Study about virtualization technologies (VLANs, Virtual Private Networks,..).
- *Chapter 3* Is about the OpenFlow technology. In this chapter specifies the devices that are part of OpenFlow technology; the controller and the OpenFlow Switch. Moreover is describes the OpenFlow Protocol used to communicate these two devices.
- *Chapter 4* Design an application with the different OpenFlow Controller (Beacon, Trema and SNAC).
- *Chapter 5* This chapter is divided in two. In one of them explains the performance evaluation of the design controller and in the other section explains the evaluation of the Learning-Switch application of different Controllers (NOX, Maestro, Beacon and Trema).
- *Chapter 6* Conclusions and future work.

Chapter 2

Background study

This chapter presents an in-depth study of virtualization technologies that exist today. Below describes the following virtualization techniques: virtual local area networks (VLANs) and the different types: port based VLANs, MAC address based VLANs and protocol based VLANs, specifying for each their advantages and disadvantages. Virtual private networks (VPN) and different types depending on which layer of the TCP/IP protocol is based and overlay networks. Finally a clear explanation about the network virtualization, the architecture, the key features and the benefits of network virtualization.

2.1 Vlan

VLAN [8] [9] is an acronym for virtual local area network. The VLAN devices introduce in all frames a VLAN ID in the medium access control (MAC) header and the VLAN Switches use this new VLAN ID and the destination MAC address to forward all frames to the destination. The main objective of a VLAN is to divide a network into different broadcast domains. This division allows a network manager to create multiple groups of logically networked devices that are independent (act as if they were on different networks, on an independent network) but in reality are interconnected (see Figure 2.1). This network division into different broadcast domains allows the network manager to implement different types of Security and Access rules for different users that are using the network.

There are many reasons to implement a VLAN. The first reason is that through VLANs can be decreased the costs for network operations and together with the cost reduction, the network administrator has a greater ability to adapt the network to new organizational requirements. For example, only the administrator must reconfigure the new port to become part of a new network. The second reason is about the security. Without VLANs all devices or users are in the same broadcast domain. This scenario doesn't prohibit communication between different devices o users. A VLAN implementation can be used to divided the network into multiple broadcast domains. This allows the network manager to implement different types of Security rules for different users or devices that are using the network. Another reason is that with the VLANs implementation has a high level of isolation between different VLANs . Finally, it is possible decrease considerably the broadcast traffic through to the division into different broadcast domains. There are three basic types for determining how a packet gets assigned to a VLAN: Port-based VLANs, MAC address-based VLANs and finally Protocol-Based VLANs.

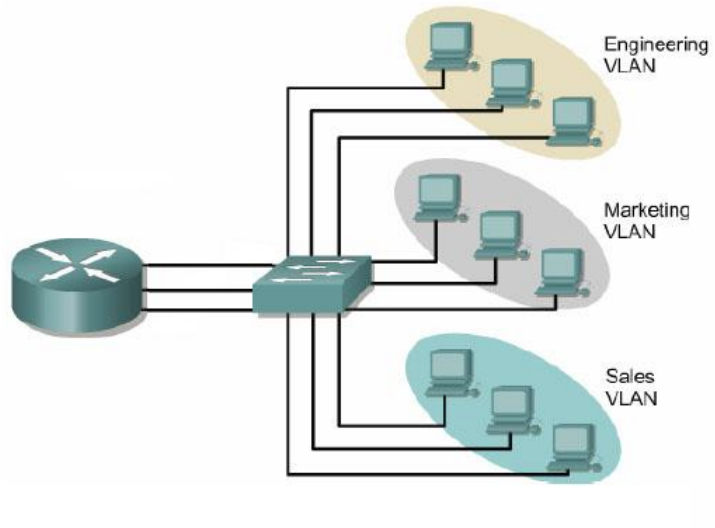


Figure 2.1: Virtual Local Area Network (VLAN)

2.1.1 Port-Based Vlan

To implement a port-based VLAN a manual assignment is performed, where each port of a switch is assigned to a specific VLAN. This model of VLAN can be configured by creating the VLAN and then adding participating ports. In a port-based VLAN, the administrator assigns the switch's ports to a specific VLAN. However, you can assign a port to the same VLAN or multiple ports to the same VLAN, but port-based VLANs cannot overlap.

2.1.1.1 Benefits

- Allows divide the network into different broadcast domains or groups and this division facilitates the network traffic for real time application servers.
- It offers network manager the ability to decide about different parameters (for example user bandwidth) of a specific port on the switch. They are easy to configure and, because all changes occur at the switch, they are transparent to the users.
- Segmenting the network improves the performance and would reduce the amount of broadcast traffic.

2.1.1.2 The main disadvantages of this method are:

- There is a limitation with respect to the mobility of devices or users inside the network. For each change made in the network, the administrator will need to reconfigure the VLAN, reassigns the port to the new VLAN.
- If a hub is connected to a port that is part of a VLAN, all devices connected to the hub are also part of the VLAN. There is no way to exclude an individual device on that hub from becoming part of the VLAN.
- Limited number of tags.

2.1.2 MAC Address-Based Vlan

In the case of MAC address-based VLANs an assignment between MAC and VLAN is performed. For this reason it is necessary that the switches have tables with these assignments (MAC – VLAN). The Switches are the devices responsible for maintaining and updating the MAC - VLAN assignment. Unlike of the port-based VLAN type, this type of VLAN provides users with greater mobility. If a workstation is moved inside the same VLAN, this not requires any configuration. Only is required to perform a configuration if the workstation is moved to other VLAN. In the next picture (See Figure 2.2) you can see a schema of MAC address-based VLANs:

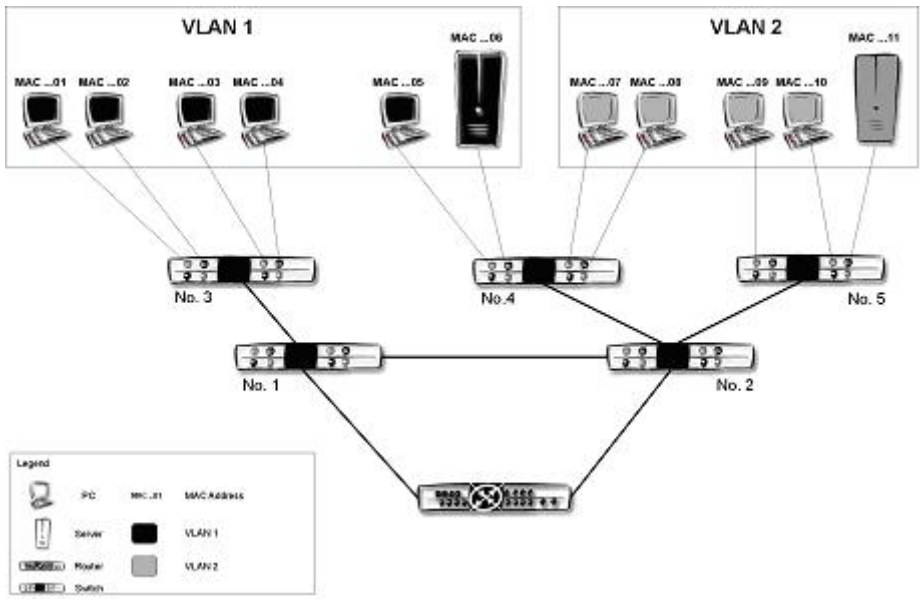


Figure 2.2: MAC address-based VLANs [10]

The main advantage is the mobility and this causes that it's not necessary the switch configuration, only if a workstation is moved inside the same VLAN. But as everything also has some drawbacks. Between the disadvantages could include the following:

- It's necessary to add manually the MAC address initially, during initial installation if the auxiliary tools are not available.
- It's impossible to assign a single MAC address to multiple VLANs.

2.1.3 Protocol-Based Vlan

VLANs can also be defined by their network or Layer 3 addresses. In this method, the devices are assigned to VLANs by using the protocols and Layer 3 addresses and the Switch delivery of packets taking into accounts the protocols and Layer 3 addresses. Specifically, in Layer 3 based VLANs, to determine a VLAN membership is taken into account two options: the protocol type (if multiple protocols are supported) or the network-layer address. The main difference with other VLAN implementations is respect the method to indicate a membership when a packet is forward. There are two different methods (See Table 2.1):

Implicit	Explicit
The MAC address indicate the VLAN membership of a packet.	A tag that is added to the package indicates the VLAN membership of a packet.
All devices that implement VLAN should share a common table with MAC addresses and their assignments.	The IEEE standard 802.1Q defines this method.

Table 2.1: Protocol-Based Vlan: Two different methods

There are advantages to defining VLANS at Layer 3. These are:

- Allows partitioning by protocol type.
- Users have mobility without having to configure each device.
- It's possible to eliminate the need for tagging using VLANs at Layer 3 and reducing the transport overhead.

The main problem with using VLANs at Layer 3 is that it affects to the performance. Compared with looking at MAC addresses in frame, VLANs at Layer 3 implies a higher processing time and consequently this causes a greater consumption. For example, devices that use Layer 2 information for VLAN definition are faster than devices that use Layer 3 information. Therefore, in terms of performance, it is better to use devices that use Layer 2 information for VLAN definition.

2.2 Virtual Private Networks

Virtual Private Networks [11], VPN, is a data network that utilizes a portion of a shared public network to extend a customer's private network. VPN connecting one or more enterprises use private and secured tunnels over a shared public network (See Figure 2.3). The main reasons for the implementation and development of virtual private networks are: the high cost of separating the private network solutions and that these can not be updated quickly to adapt to changes, the possibility of extends the limits of the physical network and finally that Internet does not guarantee privacy.

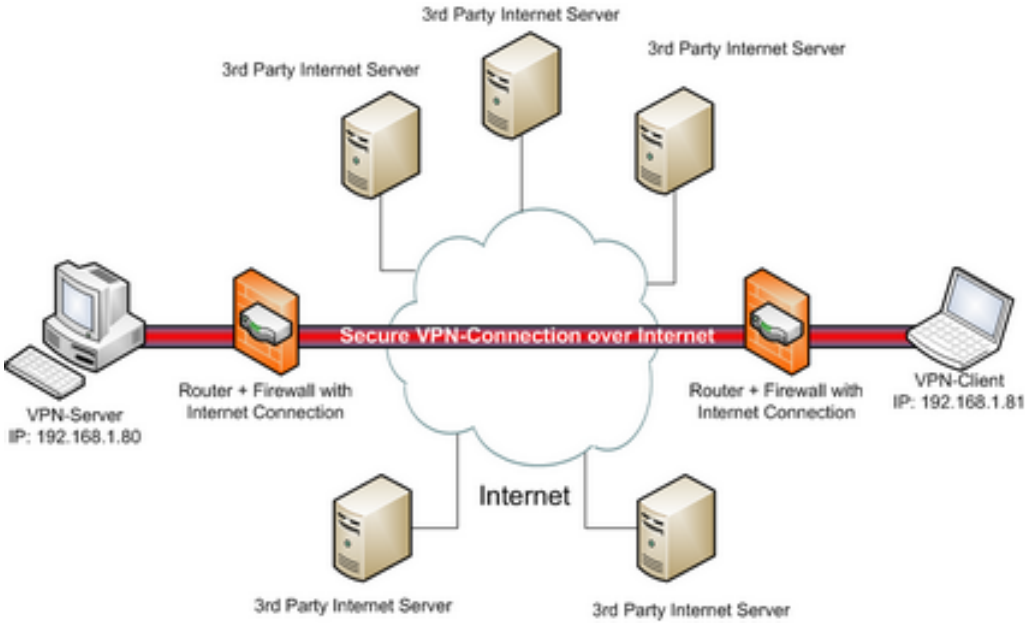


Figure 2.3: Virtual Private Networks

The are three main networks protocols used in virtual private networks. These three main networks protocols are: "Point-to-Point Tunneling Protocol" (PPTP), "Layer 2 Tunneling Protocol" (L2TP) and finally "Internet Protocol Security" (IPSec). The benefits of implementing a virtual private network are the flexibility of growth (scalability of the network) and security. The Virtual Private Networks provide encryption and additional security measures to ensure that only authorized users have access to the network and its data. Moreover, the traffic is encrypted in both directions while travels for the public shared network.

There are also some drawbacks due to the implantation of virtual private networks. One of these drawbacks would be that the availability and network performance depends on external factors. The second disadvantage is the requirement regarding the implementation of virtual private network. It should make a detailed study of public network security issues. Finally, the public shared network bandwidth limited the number of concurrent VPN connections.

In the following sections is explained in detail the different types of VPNs that exist, the devices needed to implement a VPN, and finally the different categories in which you can classify a Virtual Private Network based on the protocols used in the data plane.

2.2.1 VPN Types

In this section is explained the types of virtual private network. There are two different types of implementing a virtual private network, remote access and site-to-site. Inside site-to-site type there are two types: Intranet VPNs or Extranet VPNs (See Figure 2.4).

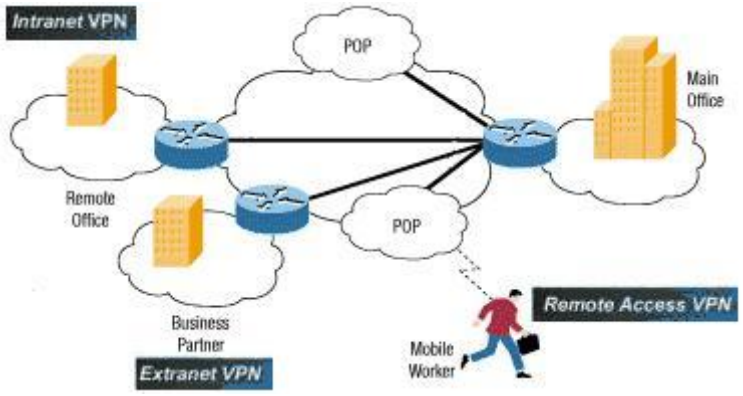


Figure 2.4: Different types of Virtual Private Network [12]

2.2.1.1 Site-to-Site VPN

Site-to-site VPNs has been developed to interconnect different sites (between the same company or between different companies) using a virtual private network. Inside site-to-site type there are two types: Intranet VPNs or Extranet VPNs.

An Intranet VPN connects resources within the same enterprise through that company's infrastructure. An Extranet VPN connects resources between different companies. Here, in Site-to-site VPNs is necessary to provide security and control the user access. Users with access to virtual private network can use all the resources of the network as if they were physically connected to it.

2.2.1.2 Remote Acces VPN

Remote access networks are an extension of the physical networks. Remote access VPNs has been implemented to provide mobility to users. With this type of virtual private network allows users to connect to the enterprise network from any location, and perform their job remotely. Here, in Remote access VPNs, is necessary to control user access and providing security. Through software provides a secure connection between remote users and their corporate networks. This secure connection is made using private and secured tunnels over shared physical network (for example the Internet).

2.2.2 VPN Devices

This section describes the relevant devices, customer and provider network devices, in a site-to-site VPN. The following figure (see figure 2.5) shows a schematic of a VPN. In this figure illustrates customer and provider network devices.

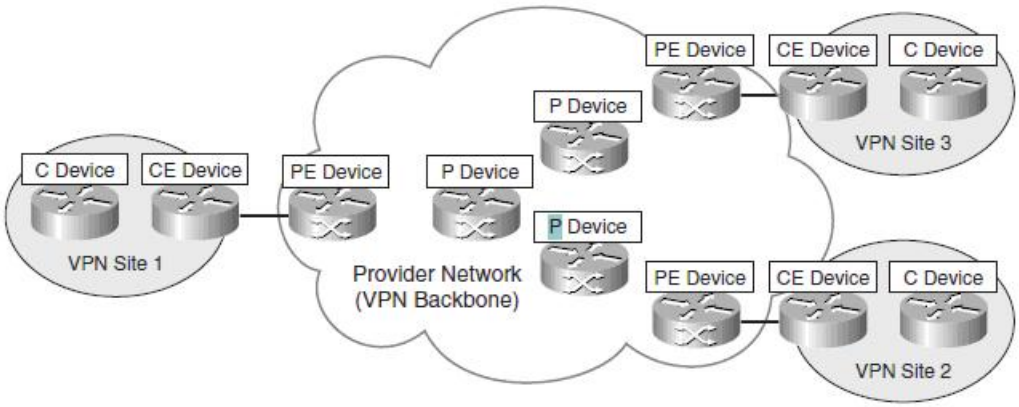


Figure 2.5: Different devices in a Virtual Private Network [13]

There is the possibility of divide the customer network Devices into two different categories (See Table 2.2):

Customer (C) devices	Customer Edge (CE) devices
These devices are simply devices as routers and switches.	Unlike C devices, CE devices are located at the edge of the customer network.
These types of devices are located inside the customer network and don't have direct connectivity to the service provider network.	The objective of these devices is to connect to the provider network (via Provider Edge [PE] devices).
C devices are outside the VPN (not aware of the VPN).	

Table 2.2: Two devices categories in the Customer Network

Also, there is the possibility of divide the service provider network devices into two categories (See Table 2.3):

Inside the PE devices, there are 3 different types of devices that can be classified as follows:

Service Provider (P) devices	S. Provider Edge (PE) devices
These devices are simply devices as routers and switches.	Unlike P devices, PE devices connect directly to the customer networks via CE devices.
These types of devices are located inside the provider network and don't have direct connectivity to the customer networks.	PE devices are aware of the VPN in PE-based VPNs, but are unaware of the VPN in CE-based VPNs.
P devices are outside the VPN (not aware of customer VPNs).	

Table 2.3: Two devices categories in the Service Provider Network

- Provider Edge routers (PE-r) or Provider Edge switches (PE-s)
- Provider Edge devices: are capable of both routing and switching (PE-rs).

2.2.3 VPN Categories

It's possible to classify virtual private networks (VPN) in different categories. There are three different categories. The first category of virtual private network is based on where VPN functions are implemented. It's possible to implement these functions in two types of devices: Customer Edge (CE) and Provider Edge (PE). Another classification of VPN is based on "services provider's role" and finally the last classification is based on protocol layer. Inside this classification there are actually different types of virtual Private Networks and depending on the functional requirements, several different methods of constructing each type of VPN is available.

Inside the classification based on protocol, there are actually different types of virtual Private Networks: Layer 1 VPN, Layer 2 VPN and Layer 3 VPN. It is necessary to consider the functional requirements, the different methods available to build a VPN and other parameters to decide what type of VPN use. These parameters should consider the following: the problem to solve, risk analysis of security offered, issues of scalability of the VPN and finally about the complexity of implementation and maintenance of the VPN.

This section is performed a brief description of the different types of virtual private networks based on protocol.

2.2.3.1 Layer 1 VPN

Layer 1 VPN (L1VPN) has been developed by the need to extend layer 2/3 (L2/L3) packet switching VPN concepts. The main characteristic is the possibility that have the customers

to offer their own services (whose payload may be any layer) using the multi-service backbone that provides this VPN type. With this, each service network has a total isolation from the other VPNs. The main difference between different types of VPN (according to the protocol) is that in Layer 2 or 3 data plane connectivity guarantee control plane connectivity and in Layer 1 VPN is not guaranteed. Layer 1 VPN can be classified in two types: Virtual Private Wire Services (VPWS) and Virtual Private Line Services (VPLS). VPWS are point-to-point while VPLS are point-to-multipoint.

2.2.3.2 Layer 2 VPN

Layer 2 VPN (L2VPN) has been implemented to transport Ethernet data (typically Ethernet data but can transport other types of Layer 2 frames) between participating sites. The main advantage is that it is simpler than the higher level protocols and is therefore more flexible than Layer 3 VPN. The main drawback is that there is no control plane for managing access through the VPN.

2.2.3.3 Layer 3 VPN

Layer 3 VPN (L3VPN) has been implemented to transport data between the distributed CEs in the physical shared network infrastructure using the layer 3 protocols, such as the IP network protocol. It's possible to classify Layer 3 VPN in two different categories: CE-based VPN and PE-based VPN.

- In the CE-based VPN, only CE devices are responsible for creating, maintaining and eliminate the tunnels. In this category of L3VPN, the shared Service Provider Network (SP) doesn't have any knowledge of the customer VPN. For Service Provider Network (SP) the network is a normal IP network, the SP is completely unaware of the existence of a VPN, and treats the packets as if it were a normal network.
- In PE-based VPN, the PE devices are responsible for maintaining all the states and connected CE devices can act as if the devices were connected to a private network. In this category of L3VPN, the PE device identifies that the traffic is VPN traffic and processes this type of traffic accordingly.

2.2.3.4 Higher-Layer VPNs

There is also a Higher-Layer VPN. This type of VPN used higher-layer protocols as the transport protocol, session protocol or application protocol. There is a VPN type, a Higher-Layer VPN, based on SSL that is very popular to have advantages in firewall and NAT traversals from remote sites. Remember that all virtual private networks (VPN) are easy to install and use and that all VPN provide security to users.

2.3 Overlay Network

Overlay network is a virtual network, a logical network built on top of a physical network with the aim of providing network services that are not available in the physical network. The most popular Overlay network is Internet. This network is a network built using an existing network, the Switched Telephone Network. It is possible the coexistence of various and different logical networks over the same physical network where each network offers its own particular service. The overlay network nodes are connected through links and these links are tunnels through the physical network.

In overlay networks, the participation is voluntary and the participants give their resources to the network. Geographically, the overlays networks are not restricted. The implementation of the overlay network is very easy. It does not require any change in the physical network to implement the overlay network. Because of this, the overlay networks have been implemented with easily since it's not necessary to make any changes to the physical network on which they are deployed. This easy implementation allows developing many different application layer overlay designs and finally the overlay network is a network that adapts easily to changes and is very easy to implement compared to other networks. Only is necessary to add an extra layer of virtualization and changes properties in specific areas of physical network.

As benefits, you can say that the overlay networks allow:

- It is not necessary to modify existing protocols or implement new equipment or modify the software existing. Only has to implement new software on top of existing software.
- It is not necessary to deploy the overlay network at every node. It is not necessary to implement the overlay network service in all nodes, all the time.

Similarly, the overlay network has disadvantages. The first drawback of overlay networks is that through these networks adds overhead, causing the need to process additional packet headers. Another drawback is that with the use of Overlay networks added complexity. More layers of functionality imply more complexity and more possible unintended interaction between layers.

2.4 Network Virtualization

Network virtualization [15] is a method, a technology that allows create isolated logical network partition on a common or physical network infrastructure. In other words, the Virtual networks are independent networks or more precisely isolated networks implemented on the same network infrastructure. In this network type, the logical network partition are independent, isolated of the other network partitions and each logical network partition can offer the same type of service that can provide in a common infrastructure, similar as if it were a non-virtualized network. These partitions, also called LINP (Logically Isolated Network Partitions), are created over the common network infrastructure. Each partition or LINP is independent from each logical partition and their objective is to provide the same services can be offered on the common network infrastructure. For the end user, the experience is how to be connected to a dedicated network that provides a set of services, including security and privacy and for managers it is easier to manage the network.

These isolation logical partitions, as shown in the following Figure 2.6 is the result in multiple heterogeneous virtual networks that coexist simultaneously using common infrastructure.

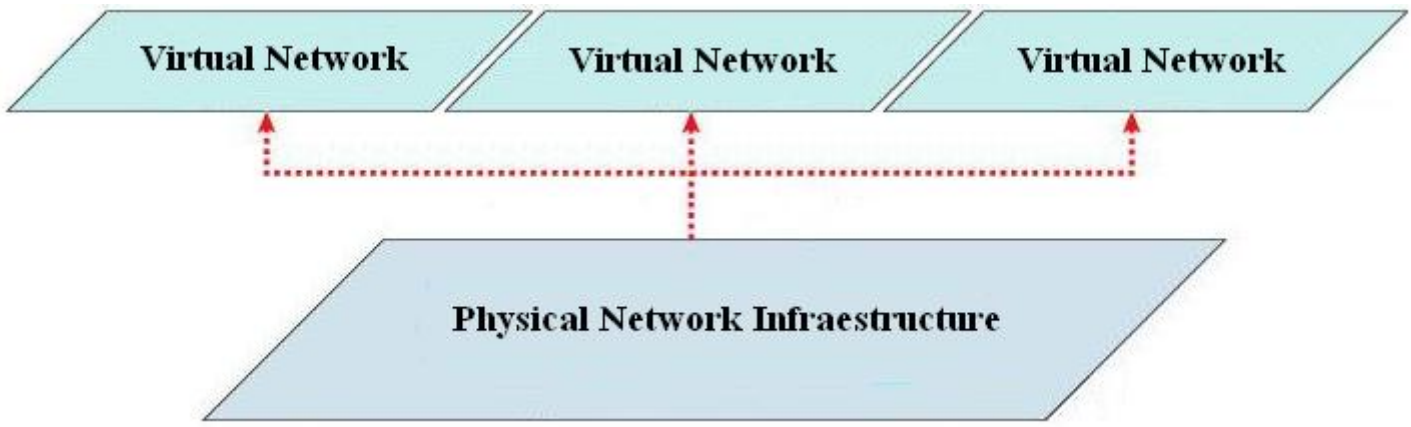


Figure 2.6: Logical Network partitions on a shared network infrastructure

The virtualization allows the combination of available resources in a network into a single platform. This combination of resources is the division of available bandwidth channels, with each channel being independent of each other and being able to reallocate these channels in real time to a device or a server. Moreover, with the virtualization technique is made better use of common network infrastructure, because it reuses the same network for multiple other network instances or used together all these resources to provide greater functionality.

These resources are used together with the aim to provide a greater functionality. Between different resources, there are routers, switches and virtual machines such as network components. So, with the use of virtualization technology can decrease the total cost by sharing network resources. For us, the concept of virtualization is not new because there are technologies such as the virtual private networks (VPN) and the virtual local area networks (VLANs) that are based on a virtualization.

2.4.1 Architecture of Network Virtualization

The virtualized network architecture is composed or divided into three main components. These are: access control, path isolation, and finally the third component is called virtual services. The following figure 2.7 shows a summary of the three components that form the architecture of a VN.

The first VN component is the access control. This component identifies users or devices, which have allowed access to the network, and are assigned to the corresponding group into the appropriate logical partition. The Users can be segmented into different groups: employees, contractors/consultants and/or guests. Also, with the access control is provided authentication, crucial for the security. The second component is Path isolation. This component refers to the creation of independent logical traffic paths over a shared physical network infrastructure. Through this component can be maintained the traffic partitioned over a common physical network infrastructure.

The main objective of creating the independent logical traffic paths over a common physical network infrastructure is to preserve and in many cases increase the scalability and security services available in a physical network. Finally, the last component is called virtual services which provide access to shared network services.

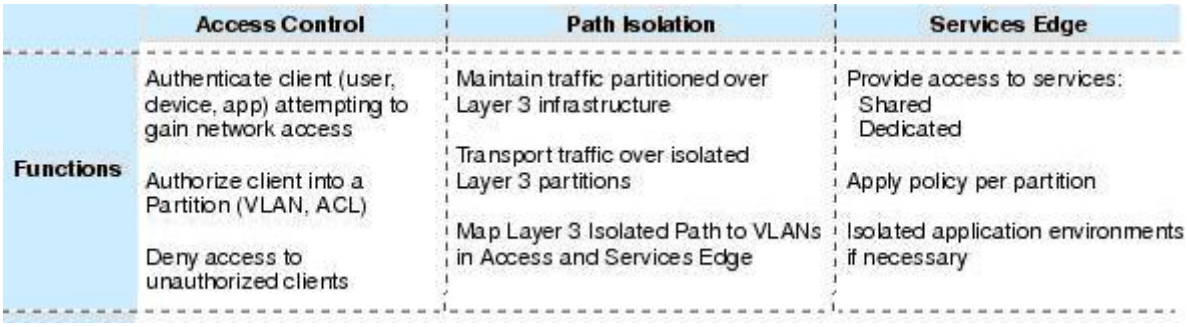


Figure 2.7: Architecture of Network Virtualization

2.4.2 Key Features of Network Virtualization

The virtualization technology allows users to have high performance resources adding multiple resources into single resource. Then, explains the main features of the network virtualization technology. Among the many features will explain the three most important: partitioning, isolation, abstraction.

The partitioning is one of the main features. Through this characteristic, the partitioning, it is possible creating different logical partition with a programmable control plane. Logical partitions allow different users to use all protocols and topologies network and functions according to their needs. With the ability to reconfigure the network, logical partitions could easily create and modify network topologies always depending on user requirements and network status.

The following key feature is the isolation between logical network partitions. Virtualization technology provides isolation between different logical partitions and through this isolation is ensured that there is no interference between the different partitions, interference which could affect network performance.

Finally, the third feature is the abstraction. With the technique of abstraction can hide the characteristics of network elements and thus allow other network elements, applications or users interact with the network elements and divide the infrastructure instances and control frameworks of network virtualization.

2.4.3 Benefits of Network Virtualization

The virtualization technology provides several benefits. Among these benefits three stand out. These three benefits are in the following table: 2.4:

Benefits
Virtualization technology offers businesses and organizations the opportunity to improve the efficiency and scalability and reducing operational costs and complexity.
The technology of network virtualization offers users an efficient and centralized management of network resources, simplifying the different maintenance tasks.
Through the virtualization technique is made more efficient use of network infrastructure in order to obtain greater capabilities.

Table 2.4: Benefits of Network Virtualization

Chapter 3

OpenFlow

3.1 Introduction

This chapter is performed a brief introduction to the Software-Defined Networking (SDN) concept and a detailed study of OpenFlow technology. OpenFlow [24] is a new technology based on the concept Software-Defined Networking (SDN). This concept is to break with the conventional model of network where is the Switch that decides the actions that have to do. OpenFlow technology moves the control logic to an external controller (typically an external PC) and this controller is responsible for deciding the actions that the Switch must perform. This communication between the controller and the Data Path is made, on the network itself, using the protocol that provides OpenFlow (OpenFlow Protocol).

There is an organization called Open Network Foundation (ONF) that is responsible to control and publish the different OpenFlow specifications. This organization has been formed recently, has a nonprofit objective and will give the trademark license "OpenFlow Switching " to companies that adopt this standard.

3.2 Software-Defined Networking (SDN)

The Software-Defined Networking (SDN)[19] [1] is a concept that is to break with the traditional networks where the switch decides the actions to do. The SDN concept was introduced by Nick Mckeown [20], a professor at Stanford University and is based on defining a model where all switches move the capacity of decision to a central element, to a controller.The SDN concept is closely related to the following idea: the Network as a Service (NaaS). That is, the service provided consists to the assignment of a certain amount of network resources that can be managed and used by the user, where each of these allocations of network resources, logical or virtual, are defined as Slice. Each Slice works completely independently from the rest (Traffic separation) although all slices share the same physical network infrastructure.

3.2.1 Traditional Network Architectures

A network switch is an interconnection device capable of analyzing the headers of the received packets and makes decisions about how to route packets. There are different types of switches, depending on the layer of the OSI model in which they work.

- Switch Layer 1 (HUB): working in the physical layer and are used as signal repeaters and regenerators.
- Switch Layer 2 (Switch): working on the physical and link layer and are used to interconnect devices that belong to the same network.
- Switch layer 3 (Router) working in the physical, link and network layer and are used to interconnect devices that belong to different networks.

The architecture of the network switches consists of two levels:

- Data Path: Responsible for performing packet switching.
- Path Control: Responsible for making decisions about how they have to route packets.

A conventional switch, when receiving a packet decides the output and communicate this decision to Data Path for implementing it. Typically, in classical network architecture, the data path and the decision-making process of switching or routing are collocated on the same device and it is communicate via an internal proprietary bus. In the case of a switch for instance, each port is configured to send and receive packets, and perform tasks that were determined by the "brain" of the device, which is where higher-level decisions are made.

The Software-Defined Networking consists to break with conventional architecture of the switches. Thus, OpenFlow defines a standard upon which these two concepts can be abstracted. This means that these two functions no longer have to occur on the same device. This is that the control logic is moved to an external controller (typically a external PC) and the controller talks to the Data Path, over the network itself, using the OpenFlow protocol.

Since the control plane has been separated from the data path, it becomes possible for network researchers to develop their own algorithms to control data flows and packets.

OpenFlow provides amongst other things. There are the following:

- More speed, scale, fidelity of vendor hardware.
- More flexibility and control of software and simulation.
- Vendors don't need to expose implementation.
- Leverages hardware inside most switches today (ACL tables).

The main components of a controller-based OpenFlow network are OpenFlow Switches and Controllers running on a server.

3.3 OpenFlow Switch

It's possible divide an OpenFlow Switch in three parts (See Figure 3.1):

1. A flow table to indicate to the switch as it has to process the flow. This flow chart is composed of actions, actions that are associated with each flow.

2. A Secure Channel necessary to connect the switch with a remote control device (controller). Through this secure channel, OpenFlow provides a communication between the switch and the controller through the use of the OpenFlow protocol.

3. The third component is the OpenFlow Protocol. Using this protocol, OpenFlow provides a standard and open communication between the controller and the switch. The OpenFlow Protocol specifies a standard interface through which entries in the Flow Table can be defined externally. Finally, it is not necessary that the researchers programmed the switch, because the OpenFlow switch avoids this necessity.

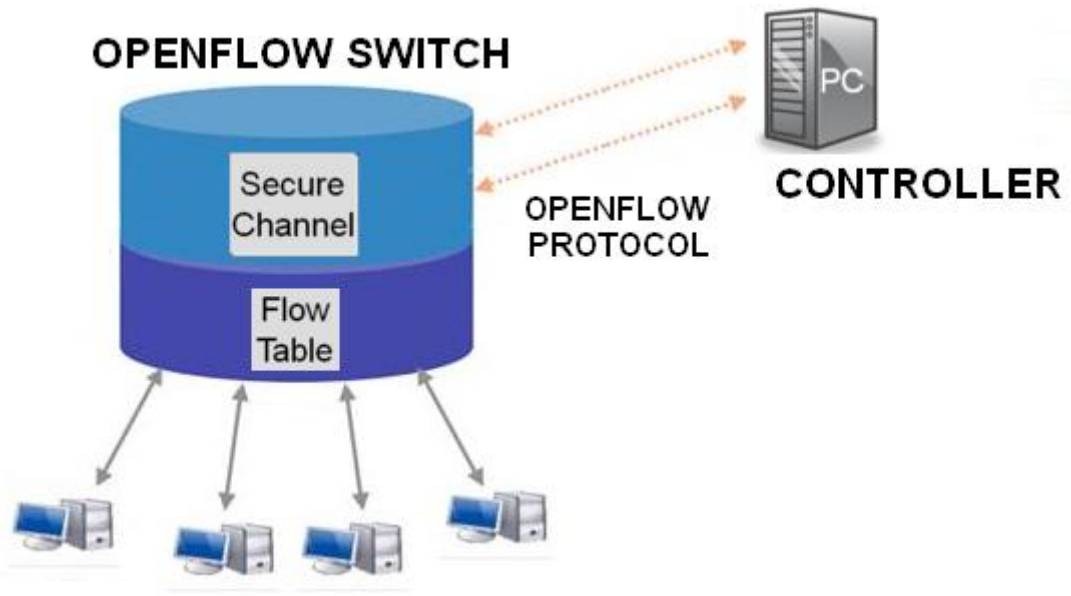


Figure 3.1: OpenFlow Switch [23]

The OpenFlow Switch can have one or multiple flow tables and different group table, which performs packet lookup and forwarding. Each flow table contains a set of flow entries to apply to matching packets or action associated with each flow entry. The collection of flow entries on a network device is called the "flow table". An entry in the Flow-Table has three fields:

- A packet header that defines the flow

- Counters to update for matching packet: Counters are maintained per-table, per-flow, per-port and per queue. Duration refers to the time the flow has been installed in the switch.

- Actions to apply to matching packets: Each flow entry is associated with zero or more actions that dictate how the switch handles matching packets. Action lists for inserted flow entries must be processed in the order specified and a switch is not required to support all action types. When the connection is made between the controller and the switch, this latter device indicates to the Controller the optional actions that implements. If there is not forwarding actions, the switch will decide according to their own configuration what to do with these specific packages. In this case, the switch can forward packets to the Controller, can remove the packages or finally can continue to the next flow table. In contrast, if a matching entry is found, the Switch will execute the instructions associated with this specific flow entry.

The set of actions supported by an OpenFlow Switch is extensible, but the minimum requirement for all switches is that the data path must have flexibility to provide high performance and low cost. Therefore, in the OpenFlow Switch Specification [22], are defined all the set of actions supported by an OpenFlow Switch. There is the possibility to classify the OpenFlow Switches into two types: Dedicated OpenFlow Switches and OpenFlow-Enabled Switches. The Dedicated OpenFlow Switches do not support normal Layer 2 and Layer 3 processing and in the OpenFlow Enabled Switches, have been added as a new feature the OpenFlow Protocol and interfaces.

3.3.1 Dedicated OpenFlow switches

A dedicated OpenFlow Switch is a dumb data Path element that forwards packets between ports, as defined by a remote control process. Figure 3.1 shows an example of an OpenFlow Switch. In this context, flows are broadly defined, and are limited only by the capabilities of the particular implementation of the Flow Table. For experiments involving non-IPv4 packets, a flow could be defined as all packets matching a specific (but non-standard) header. Each flow-entry has a simple action associated with it; the three basic ones (that all dedicated OpenFlow switches must support) are:

1. Forward this flow's packets to a given port (or ports). This allows packets to be routed through the network.

2. Encapsulate and forward this flow's packets to a controller. Packet is delivered to Secure Channel, where it is encapsulated and sent to a controller. Typically used for the first packet in a new flow, so a controller can decide if the flow should be added to the Flow Table. Or in some experiments, it could be used to forward all packets to a controller for processing.

3. Drop this flow's packets. Can be used for security, to curb denial of service attacks, or to reduce spurious broadcast discovery traffic from end-hosts.

3.3.2 OpenFlow-enabled switches

Some commercial switches, routers and access points will be enhanced with the OpenFlow feature by adding the Flow Table, Secure Channel and OpenFlow Protocol. Typically, the Flow Table will re-use existing hardware, such as a TCAM; the Secure Channel and Protocol will be ported to run on the switch's operating system.

Finally, the interconnection between the Controller and the OpenFlow Switch is through the Secure Channel. Through the Secure Channel, the Controller exchanges messages with other devices, normally Switches, with the aim of configure and maintain the Switch, receive events from the switch and finally send packets out the switch. OpenFlow provides a protocol to perform the communication between the Controller and the OpenFlow Switch. All messages exchanged by the Controller and the Switch must follow the format indicated by the OpenFlow protocol.

The detailed requirements of an OpenFlow Switch are defined by the OpenFlow Switch Specification [22].

3.4 OpenFlow Protocol

The OpenFlow Standard [22] defines an OpenFlow Protocol for communication between the OpenFlow switch and the controller. The OpenFlow Switch must be able to establish communication with a controller at a user-configurable IP address, using a user-specified port. If the switch knows the IP address of the controller, the switch initiates a standard TCP connection to the controller.

When an OpenFlow connection is first established, each side of the connection must immediately send an HELLO message with the version field set to the highest OpenFlow protocol version supported by the sender. Upon receipt of this message, the recipient may calculate the OpenFlow protocol version to be used as the smaller of the version number that it sent and the one that it received. If the negotiated version is supported by the recipient, then the connection proceeds. Otherwise, the recipient must reply with an ERROR message (HELLO-FAILED)and then terminate the connection.

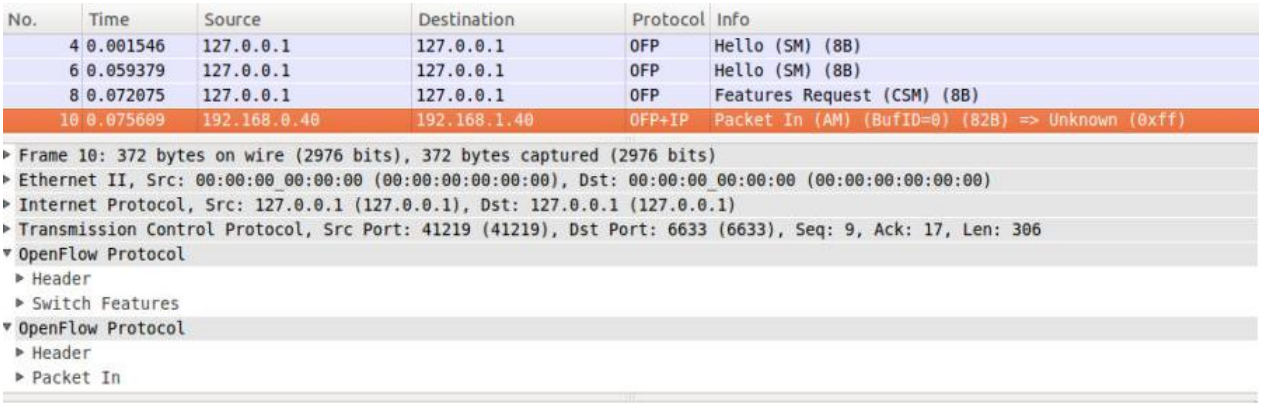


Figure 3.2: Communication between OpenFlow Switch and Controller

The OpenFlow protocol supports three different message types that are the followings:

3.4.1 Controller to Switch Message

Initiated by the controller, used to directly manage or inspect the state of the switch and sent over the Transport Layer Security (TLS) session establishment or the Transmission Control Protocol (TCP) session establishment. This message may or may not require a response from the switch. With this message, the controller sends a features request message to the switch. The switch must reply with a features reply that specifies the capabilities supported by the switch.

Inside this message can distinguish four types. Modify-State messages are sent by the controller to manage state on the switches. Their primary purpose is to add/delete and modify flows in the flow tables and to set switch port properties. Read-State messages are used by the controller to collect statistics from the switchs flow-tables, ports and the individual flow entries. Send-Packet is used by the controller to send packets out of a specified port on the switch. The last message is the Barrier request/reply message. This is used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.

3.4.2 Asynchronous Message

Switches send asynchronous messages to the controller to denote a packet arrival, switch state change, or error. Inside the Asynchronous message there are four main asynchronous message types that it is described below.

The message Packet-in is sent to the controller for all packets that do not have a matching flow entry or if a packet matches an entry with a send to controller action. If the switch has sufficient memory to buffer packets that are sent to the controller, the packet-in events contain some fraction of the packet header (by default 128 bytes) and a buffer ID to be used by the controller when it is ready for the switch to forward the packet. Switches that do not support internal buffering (or have run out of internal buffering) must send the full packet to the controller as part of the event.

The flow modify message is send when a flow entry is added to the switch, an idle timeout value indicates when the entry should be removed due to a lack of activity, as well as a hard timeout value that indicates when the entry should be removed, regardless of activity. The flow modify message also specifies whether the switch should send a flow removed message to the controller when the flow expires. Flow modify message which delete flows may also because flow removed messages.

The port-status message is send by the switch but it is expected to send port-status messages to the controller as port configuration state changes. These events include change in port status or a change in port status as specified by 802.1D.Finally, the error message is send by the switch to notify the controller of problems using error messages.

3.4.3 Symmetric Message

Symmetric messages are sent without solicitation, in either direction. Inside of the symmetric messages can distinguish three types. One type is a hello message that it is exchanged between the switch and controller upon connection startup. Another type is the echo request/reply message. It can be sent from either the switch or the controller, and must return an echo reply. They can be used to indicate different parameters of a controller-switch connection. Vendor message is the third type of message. This message provides a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space. This is a staging area for features meant for future OpenFlow revisions.

3.5 Controller

The controller is the main device, it is responsible for maintains all of the network rules and distributes the appropriate instructions for the network devices. In others words, the OpenFlow controller is responsible for determining how to handle packets without valid flow entries, and it manages the switch flow table by adding and removing flow entries over the secure channel using the OpenFlow protocol. The controller essentially centralizes the network intelligence, while the network maintains a distributed forwarding plane through OpenFlow Switches and routers. For this reason the controller provides an interface for manage, controlling and administrate the Swtche's flow-tables. Typically, the Controller runs on a network-attached server and there are different control configurations depending on:

3.5.1 Location

According to how the delegation of switch management to controllers is performed. One can distinguish two types of settings regarding the location of the controller. One of them would be a Centralized configuration where a single controller manages and configures all devices and another configuration possible would be the distributed configuration, where is available one controller for each set of switches.

3.5.2 Flow

- Flow Routing: Every flow is individually set up by controller. Exact-match flow entries. Flow table contains one entry per flow. Good for, for example, campus networks.

- Aggregated: One flow entry covers large groups of flows. Wildcard flow entries. Flow table contains one entry per category of flows. Good for large number of flows, e.g. backbone.

3.5.3 Behavior

- Reactive: First packet of flow triggers controller to insert flow entries. Efficient use of flow table. Every flow incurs small additional flow setup time. If control connection lost, switch has limited utility.

- Proactive: Controller pre-populates flow table in switch. Zero additional flow setup time. Loss of control connection does not disrupt traffic.

Depending on the configuration, controllers are more sophisticated than others. It's possible to configure a simple controller that dynamically add/remove flows and where the researcher can control the complete network of OpenFlow Switches and is responsible to decide how all flows are processed. Also can be imagined a sophisticated controller which can support multiple researchers, each with different accounts and permissions, enabling them to run multiple independent experiments on different sets of flows. This flows can be identified as under the control of a particular researcher and can be delivered to a researcher's user-level control program which then decides if a new flow-entry should be added to the network of switches.

For more information about the OpenFlow Controller see the OpenFlow Standard [22]. Today there are different controller implementations available. The following table 3.1 shows the top 5 controllers available today and describes the main feature.

Controller	Main characteristic
NOX	Open-source OpenFlow controller that provide a simplified platform for writing network control software in C++ or Python
Beacon	Beacon is a fast, cross-platform, modular, Java-based controller that supports both event-based and threaded operation.
Trema	Full-Stack OpenFlow Framework for Ruby/C
Maestro	A scalable control platform written in Java which supports OpenFlow switches
SNAC	SNAC is an OpenFlow controller, which uses a web-based policy manager to manage the network.

Table 3.1: The top 5 Controllers available today and the main features

Chapter 4

OpenFlow Controllers Design

This chapter explains the design of an application for a Controller. Have been designed three different applications (one for each different type of controller). Therefore, there are three different applications, each based on a different programming language: a Java-based application (Beacon Controller), Ruby based application (Trema OpenFlow Controller) and finally C++ based application (SNAC Controller).

4.1 Design a Controller

4.1.1 Goal of the Application

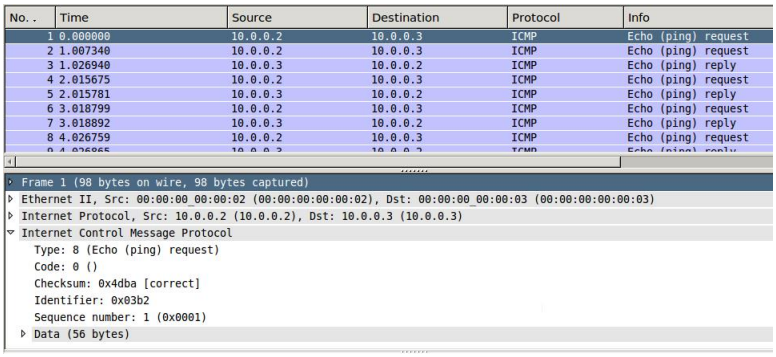
The goal of the application is: "redirect pings or ICMP Packages (Ethernet Packages) sent between the host 2 and host 3 to the host or device 4." (see Figure 4.1). All applications have the same functionality/goal, regardless of the type (Beacon, Trema and SNAC OpenFlow Controller). Remember that Pings or ICMP messages consist of two types of messages: Ping Request and Ping Reply. The Ping Request is the message that sends the origin to the destination and Ping Reply is the confirmation, sent by the receiver to the origin, to inform that has received the message Ping Request.

4.1.2 The Network Design Decisions

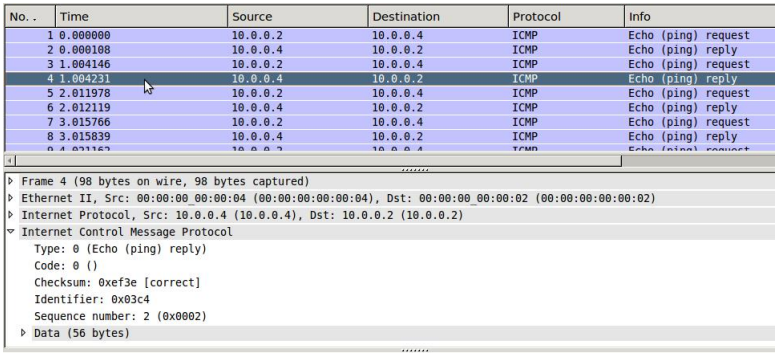
The Network Design Decisions [26] is taken into account to design of different controllers. The Network Design Decisions [26] define the possible modes of operation of a controller. These possible modes of operation of a controller are the followings: Centralized Control vs. Distributed Control, Flow based or aggregated and finally about the Rule Creation two options, Reactive vs. Proactive.

Based on the Network Design Decisions [26], the controllers operation mode is the following:

- Centralized, since only need a controller to manage the Switch (in our case a single switch but if we had more devices or switches will continue with a single controller to control all devices).
- Flow based because each flow is set individually by the controller and flow table contains one entry per flow.



(a) Ping Origin Host 2 and Destination Host 3



(b) Redirect Ping to the Host 4

Figure 4.1: Redirect Pings or ICMP Packets to the Host 4

- Reactive and Proactive.

4.1.2.1 Controller Working in a Reactive Mode

The controller is designed initially to do nothing until it receives the first message, the Ping Request Message. When he Controller receives the first message (Ping Request Message) introduce the rule in the Switch Flow Table in order to forward the packet to the host 4 (see Figure **??**). This operation method would be the controller with a **Reactive Mode**. After, the OpenFlow Switch will use the specific rule to send the packet to the correct destination.

4.1.2.2 Controller Working in a Proactive Mode

The controller anticipates the response message, the Ping Reply Message, and finally before receiving the Ping Reply message, the controller inserts the rule for this message in the Switch Flow Table. Then, when the switch receives the the Ping Reply message, the switch has the rule in its table to send the packet to the source (see Figure **??**). This mode of operation is **Controller working in a Proactive Mode**.

4.1.3 Scenario

The scenario 4.2 used to verify the functionality of the designed application, ie the forwarding of messages ping - is composed by 3 host, 1 switch and the controller designed. These 3 devices and the switch have been emulated by MiniNet tool [28].

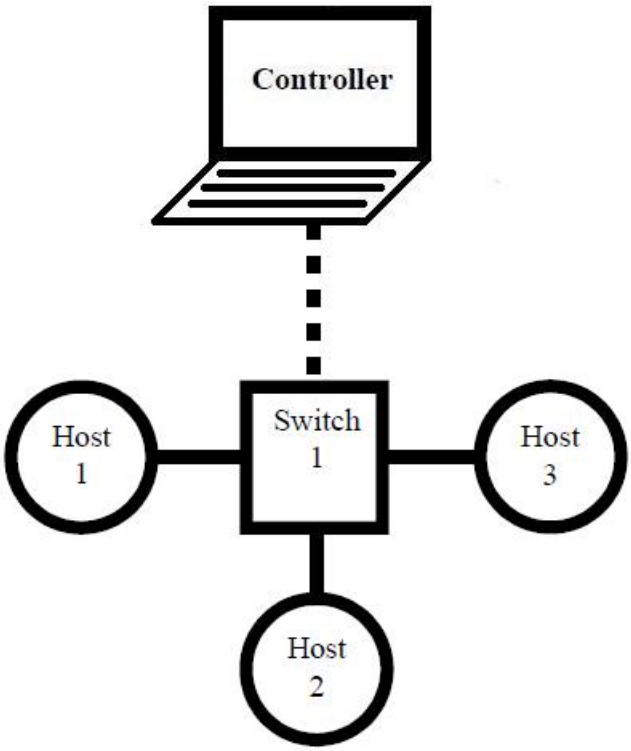


Figure 4.2: Scenario used to verify the functionality of the designed application

4.1.3.1 MiniNet

MiniNet [28] supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a laptop or other PC. With MiniNet is possible to create a scalable (up to hundreds of nodes, depending on your configuration) networks on a single PC by using Linux processes in network namespaces. Also, MiniNet allows you to quickly create, interact with, customize and share a software defined network prototype, and provides a smooth path to running on hardware.

MiniNet[28] provides an easy way to get correct system behavior and experiment with topologies. The code you develop and test on MiniNet, for an OpenFlow controller, modified switch, or host, can move to a real system with no changes, for real-world testing, performance evaluation, and deployment.

4.1.3.2 Requirements to implement the Scenario

The requirements to implements this scenario are the followings (see Figure 4.1). For more information about the requirements: (See Appendix A).

OS Type	OS Version	V. Software	X Server Terminal	Terminal
Linux	Ubuntu 10.04	VirtualBox	X server	gnome terminal + SSH

Table 4.1: Requirements

4.1.3.3 Evaluation Test

The test is run on the same device the designed application and the MiniNet tool to check the Controller behavior. With the MiniNet tool have been emulated a Switch connected directly to the controller. Each Switch has 3 devices connected and consequently these devices are connected directly to the controller.

The command to create this network in the Virtual Machine with MiniNet is the following 4.2. This tells MiniNet to start up a 3-host, single-(openvSwitch-based)switch topology, set the MAC address of each host equal to its IP, and point to a remote controller which defaults to the localhost. Exactly, using this command are performed the following actions:

- Create 3 virtual hosts, each with a separate IP address.
- Create a single OpenFlow software switch in the kernel with 3 ports.
- Connected each virtual host to the switch with a virtual ethernet cable.
- Set the MAC address of each host equal to its IP.
- Configure the OpenFlow switch to connect to a remote controller.

sudo mn –topo single,3 –mac –switch ovsk –controller remote –ip 192.168.56.1 –arp

Table 4.2: Command MiniNet

There have been performed several types of simulations, sending messages between host 2 and 3 and checking the behavior of the Controller.

4.1.4 Flow Diagram

The followings Flow Diagram shows graphically the operations doing the OpenFlow Switch (See Figure 4.3) and OpenFlow Controller (See Figure 4.4) when they receive a package.

4.1.4.1 Flow Diagram of OpenFlow Switch.

On receipt of a packet, an OpenFlow Switch performs the functions shown in Figure 4.3. Initially, the Switch receives the packet and checks its routing table if there is a rule to indicate what action has to perform the Switch with this package. Here, there are two possible cases:

- If the routing table of Switch has a specific rule for that packet, the Switch will apply the rule and send the packet according to the rule.
- Otherwise, if there is no rule for this type of package, the Switch forwards the packet to the Controller and waits for a response from the Controller. In this response, the Controller will indicate to the Switch the actions to be performed on this package.

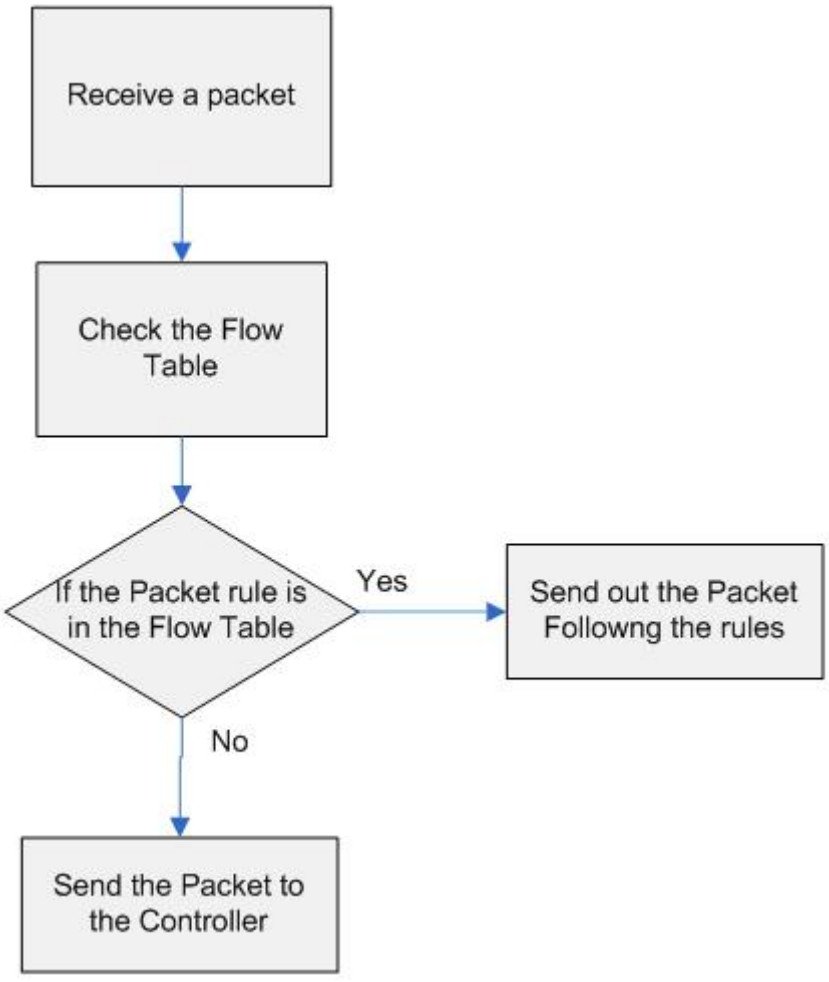


Figure 4.3: Flowchart detailing packet flow through an OpenFlow switch

4.1.4.2 Flow Diagram of the Controller

On receipt of a packet, an OpenFlow Controller performs the functions shown in Figure 4.4.

If the switch does not know what actions to perform on a specific package, forward the packet to the controller. The controller will receive the packet from the switch and the same controller examines the package to verify that it is an Ethernet packet type. Here, there are two possible cases:

- If the packet is not Ethernet packet, the Controller discards the packet.
- If is an Ethernet packet, the controller will analyze the packet headers to obtain the source and destination information (MAC address and IP address). After, the Controller inserts the necessary rules in the Switch in order that the Switch can route the packet to the destination.

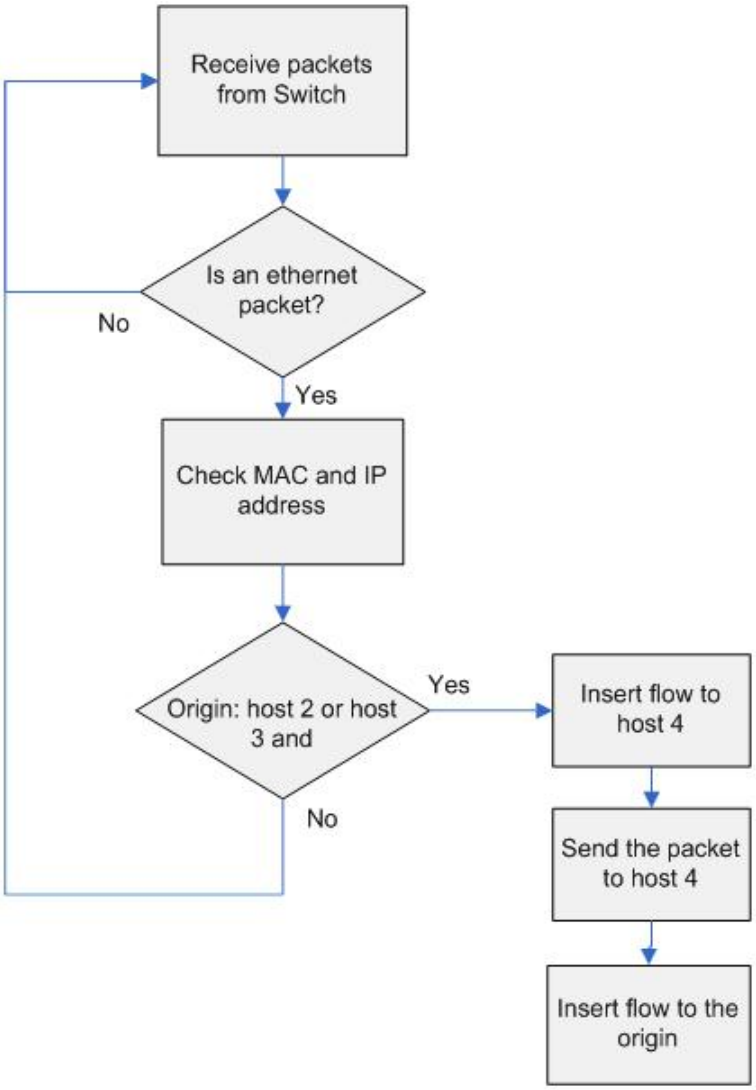


Figure 4.4: Flowchart detailing packet flow through an OpenFlow Controller

4.2 Trema OpenFlow Controller

Trema [5] is an open source OpenFlow controller platform for research and academia. It is a free OpenFlow controller platform (GPL v2). Assists anyone who wants to develop his/her own OpenFlow controller and Trema is not targeted for any specific OpenFlow controller implementation. Trema has a Multi-process modular architecture that permits an extensibility and stability and can be extended to distributed controller. Trema provides an integrated testing and debugging environment. With it, Trema is able to provide system support to manage, monitor, and diagnosis entire system.

Trema allows implementing OpenFlow controllers in C and Ruby. Initially, before the year 2011, only had full supported with C and partially supported with Ruby, but planned to be fully supported in the year 2011. Now, the controller has a complete support in C and Ruby and includes a complete set of APIs available for both C and Ruby programmers. For this, the user modules can be written in C or Ruby. The controller was originally developed by NEC research lab.

4.2.1 Goals for Trema Project

- Provide fairly good quality OpenFlow controller platform to researchers, developers and a continuous development, maintenance, bug-fixes and user support from the project team (researchers and professional programmers).
- Researchers develop their own controllers on top of Trema and contribute to the community. Recycling controller modules accelerates our research activities.

4.2.2 Programming language

C [35] is a general-purpose programming language. It has been closely associated with the UNIX operating system where it was developed, since both the system and most of the programs that run on it are written in C. The language, however, is not tied to any one operating system or machine; and although it has been called a system programming language because it is useful for writing compilers and operating systems, it has been used equally well to write major programs in many different domains. C is a low-level programming language and it's easy to learn. Finally C has proven to be an extremely effective and expressive language for a wide variety of programming applications.

Ruby [36] is an object-oriented programming language, written in C and that combine some of the best features of C, Perl, and Python. This is the ruby main objective: incorporate or combine the strengths of languages like Perl, Python and C. Moreover, Ruby is a programming language portable and runs under GNU/Linux as well as DOS, MS Windows and MAC.

4.2.3 Design a Trema Application

The following Flow Diagram shows graphically the Trema Controller operation mode when they receive a package (see Figure 4.5). For more information about the design application code, see appendix B.1.

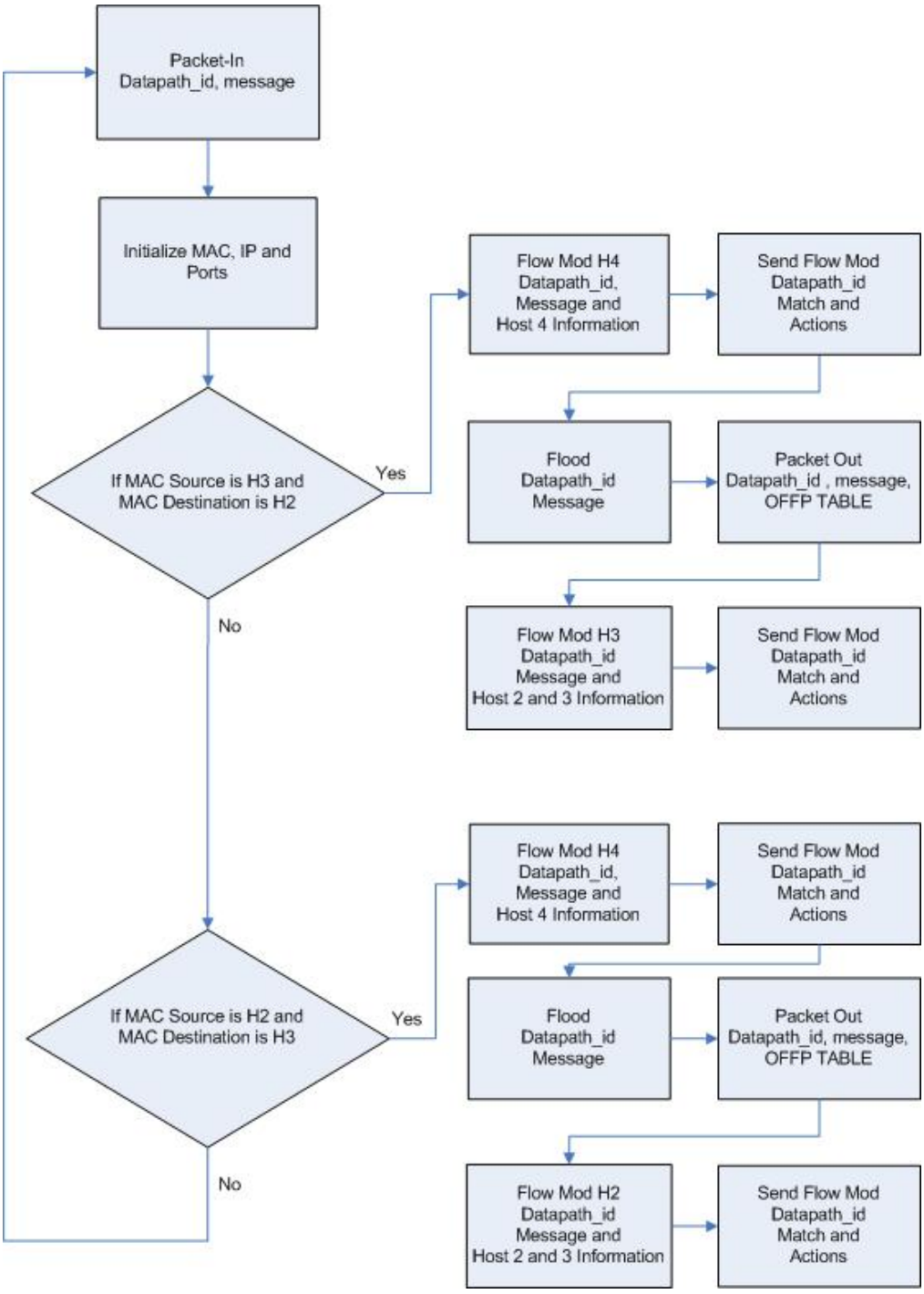


Figure 4.5: Flow Diagram to show the Trema Application design

4.3 Beacon OpenFlow Controller

The OpenFlow ecosystem has numerous controllers in multiple languages (C, C++, Java, Python and Ruby for starters) and **the Beacon Controller** [3] is one of those controllers. It's a fast, cross-platform, modular that supports both event-based and threaded operation. Beacon is a Java-based OpenFlow controller platform and it is very easily developed using the Eclipse Integrated Development Environment, and best run in your host environment rather than a VM. It is necessary download Beacon and install Eclipse on your host machine.

4.3.1 Main Features

The following section, explains the key features for the Beacon Controller. The stability is one of the key characteristic. Beacon has been in development for a year and a half, and has been used in several research projects, networking classes, and trial deployments. The language programming is another feature: Beacon is written in Java and runs on many platforms, from high end multi-core Linux servers to Android phones. Beacon is Open source because is licensed under a combination of the GPL v2 license and the Stanford University FOSS License Exception v1.0. Beacon is a Dynamic controller, code bundles in Beacon can be started, stopped, refreshed, installed at runtime, without interrupting other non-dependent bundles and Beacon is easy to get up and running. Java and and rapid development for the reason that eclipse simplify development and debugging of your applications. Finally, the last characteristic is that the Beacon Controller is multithreaded. Beacon optionally embeds the Jetty enterprise web server and a custom extensible UI framework.

4.3.2 Programming language

Java [38] is a general-purpose, concurrent, class-based, object-oriented programming language with a lot of features. Among many other features, Java is a simple programming language because the programs are easy to write and debug because java does not use the pointers explicitly. Another main feature is that Java is Platform Independent. This last feature is one of the most important characteristic of java language that makes java as the most powerful language. The programs written on one platform can run on any platform provided the platform must have the JVM. The concept of Platform independent is Know as Write-once-run-anywhere (WORA). This concept, WORA, makes that the java code is portable. The programs executed on one device can run on any devices provided the system must have interpreter for the JVM. Finally, Java is a Secure programming language because Java uses the public key encryption system to allow the java applications to transmit over the internet in the secure encrypted form.

In general, the language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java is currently one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users.

4.3.3 Design Beacon Application

The following Flow Diagram shows graphically the Beacon Controller operation mode when they receive a package (see Figure 4.6). For more information about the design application code, see appendix B.2.

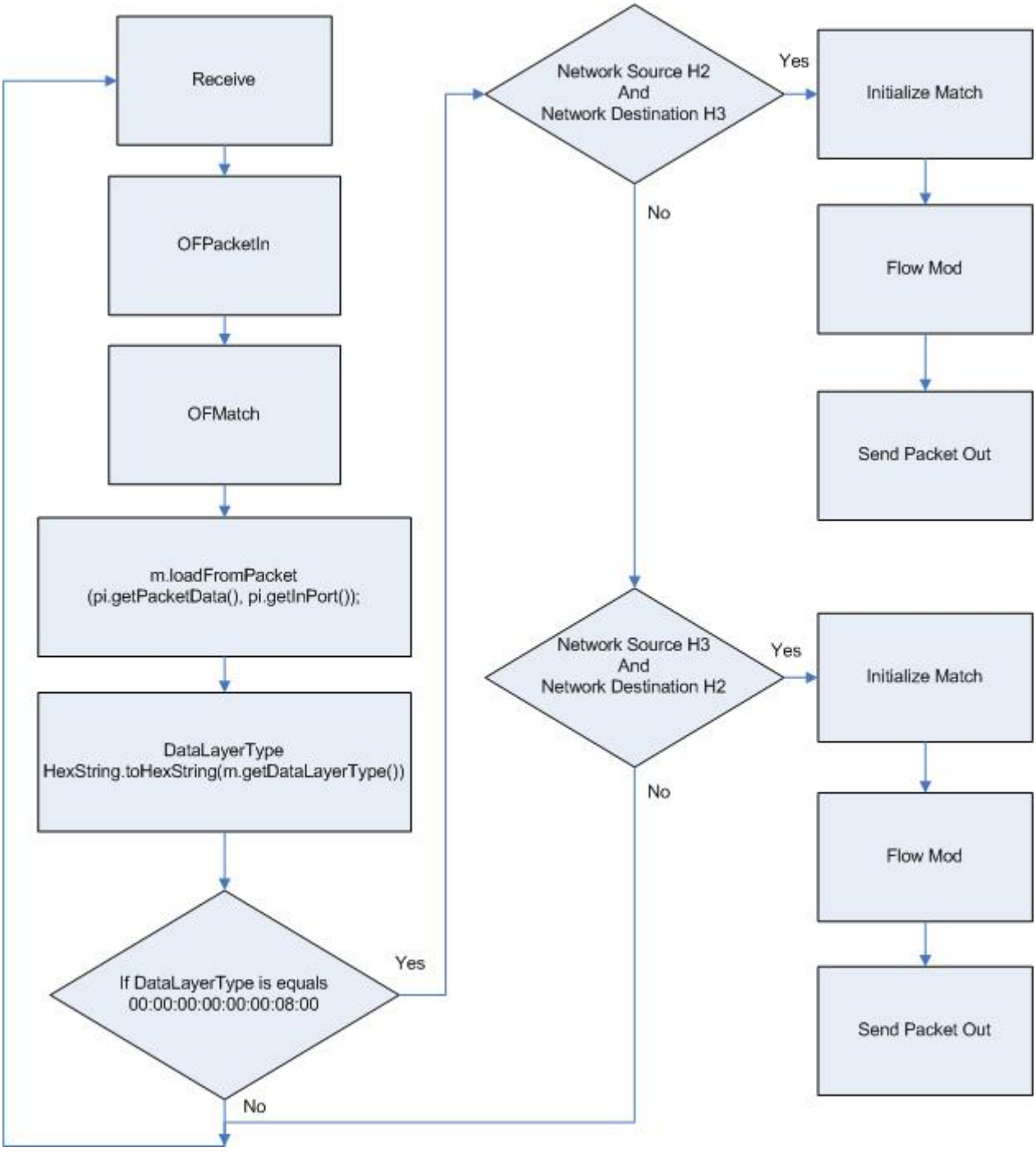


Figure 4.6: Flow Diagram to show the Beacon Application Design

4.4 SNAC OpenFlow Controller

SNAC [**?**] [39] is an open source OpenFlow controller with a graphical user interface which uses a web-based policy manager to manage the network. SNAC is built as a module of NOX for this reason requires the correct version (the latest NOX version from Noxrepo will not work) of the NOX controller. Specifically, the source code for SNAC controller comes in two parts: source code for NOX and source code for SNAC. It is distributed under the GNU Public License and incorporates a user-friendly interface to configure devices and monitor events and a flexible policy definition language.

Now, it is described the most important features that provide SNAC.

- Increased Visibility: SNAC reports many flow-level traffic details over a web UI and over a HTTP-based API.
- Captive Portal: New hosts joining a network are automatically directed to SNAC for authentication.
- Flexible policy manager: Allows administration to control NOX routing module's behavior without having to edit code.
- Tested: Used in campus networks for over 18 months.
- SNAC web-GUI allows the admission control, adding policies to control routing and shows the switches and links, the hosts and locations, the network usage and finally the network events.

SNAC was funded in part by Stanford Clean Slate Program, Nicira and GENI/NSF and was developed by a number of people including: Ben Pfaff, Bryan Forbes, Christopher Anderson, Dan Wendlandt, Eugene Lazutkin, Justin Pettit, Keith Amidon, Martin Casado, Masayoshi Kobayashi, Natasha Gude, Peter Balland, Reid Price, Srini Seetharaman, Teemu Koponen and Torrey Rice.

4.4.1 Design SNAC Application

It has been impossible to design an application with SNAC and NOX. The problem was not able to install the packages or prerequisites required to install the SNAC. This has been due to two reasons:

- Unfortunately, SNAC has a hard dependency on libboost-filesystem1.34.1. Thus, is encourage users to run SNAC on Debian systems with this package or Ubuntu 9.10. It does not support Ubuntu 10+ or 11+ at this point.
- Ubuntu 9.10 is not supported. For this reason has been impossible to install the prerequisites, necessary to run SNAC. The pre-requisite packets for Ubuntu are: openssl libboost-test1.34.1 libboost-filesystem1.34.1 libboost-serialization1.34.1 libxerces-c28 python2.5 python-twisted python-simplejson python-mako python-openssl tmpreaper python-sphinx libldap2-dev.

Chapter 5

OpenFlow Controllers Evaluation

In this chapter there are two different evaluations. First of all is performed an evaluation of the different controllers that have been designed and after is performed an evaluation of the learning-Switch application. This application has been implemented with the following controllers: Beacon, Trema, NOX and finally Maestro.

5.1 Evaluation of the different controllers designed

In this section are evaluated the designed application. Has been performed the same application using three different types of controllers. These controllers are: SNAC Controller, Beacon Controller and Trema Openflow Controller. These three controllers are explained in the previous chapter. Remember that there has been no assessment of the SNAC controller because it has been impossible to design an application due to the inability to install the Controller (problems with dependency to install the necessary requirements).

There are many different and various parameters to evaluate for an application or controller, many of them very difficult to assess due to its subjectivity, such as the "extensibility" of a controller and other parameters more simpler, less subjective, as the different programming languages used to implement these applications and their limitations. Specifically, this assessment or evaluation will focus on three basic aspects that define any application: the performance - from the point of view of throughput and latency- and the programming language.

To measure the throughput and latency you can use a benchmark tool. Benchmark is a test to compare the performance of: a system or subsystem, different computer architecture and different software. The benchmark test consists of executing a computer program or a set of programs, benchmark tools, with the aim to evaluate the performance of an object. Now, it's necessary to define and explain the throughput and latency parameters, definitions needed for a correct evaluation and to successfully perform the evaluations.

5.1.1 Scenario

The scenario used to evaluate the designed application is composed by a Controller and "N" Switches connected directly to the controller 5.1. Each Switch has a million connected devices and consequently these devices are connected directly to the controller. Switches and devices have been emulated with the Cbench tool 5.1.1.1 [31]. Below is explained in detail the Cbench tool used to emulate "N" Switches.

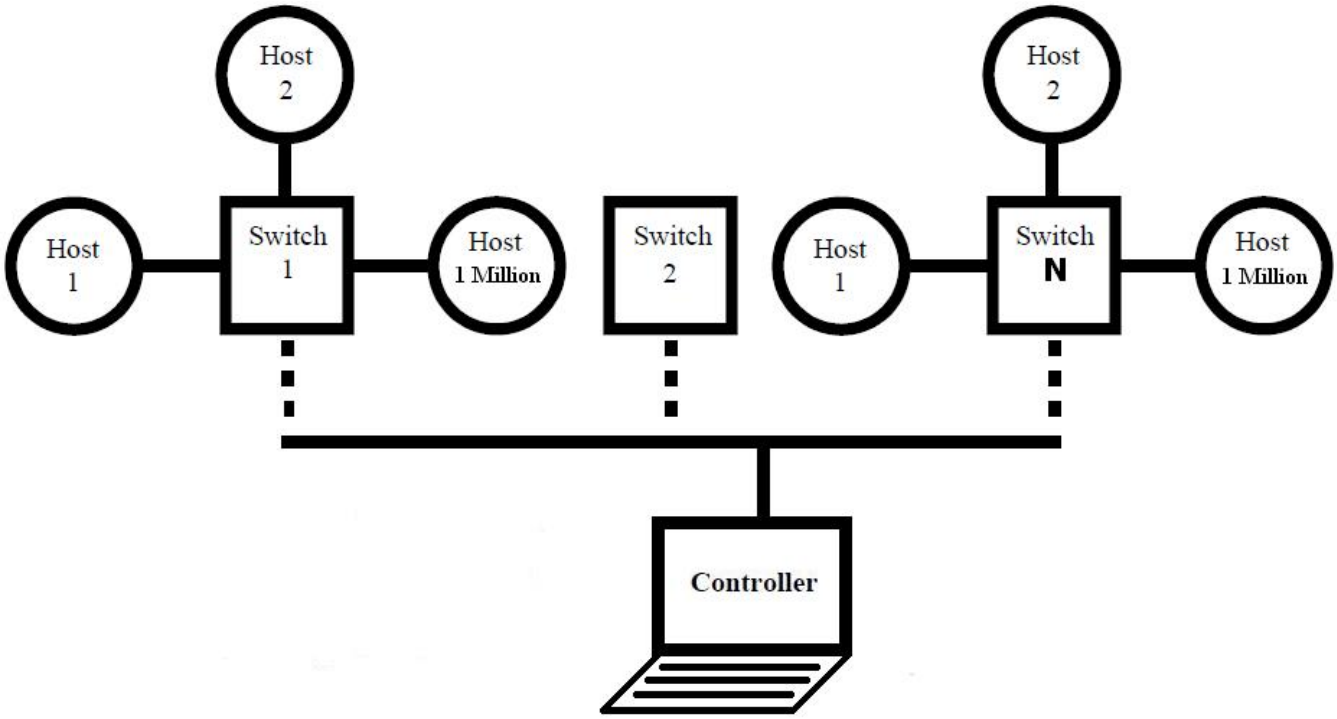


Figure 5.1: Scenario used to evaluate the designed application

5.1.1.1 Cbench Tool

The Cbench [31] exact definition is that "is a Benchmarking tool that allows to evaluate the parameters: latency and throughput of the Controllers". This tool is included in OFlops, that it is another tool to evaluate the OpenFlow Switches. OFlops implements a modular framework for adding and running implementation-agnostic tests to quantify an switch's performance.The Cbench operation is very simple: pretending to be N switches, creating N OpenFlow sessions to the controller. It has two modes:

- **Throughput mode:** for each session while the buffer is not full, store packets in and count the flow mod back.
- **Latency mode:** for each session sends a packet back and waiting in a flow mod. These two steps are repeated and count how many times occur per second.

5.1.2 Experiment setup and methodology

5.1.2.1 Characteristics to Evaluate

The throughput or network throughput is the number of transactions per second that an application can handle. There are different form of measuring this parameter, one would be using bits per second (bit/s or bps) and another way to measure the throughput performance would be through data packets per second.

Latency in a packet-switched network is the time required for a packet to arrive at its destination through the network. The delay can be caused, for example, by the physical environment or devices forming the network (Switches, routers...). There are two ways to measure the latency: the first is to measure the time it takes for a packet from source to destination (One-Way) and the second is to measure the time it takes a packet to go and return (Round-Trip). This last form of measuring the latency, Round-Trip, is the most used because it allows a single device to measure the latency of a network.

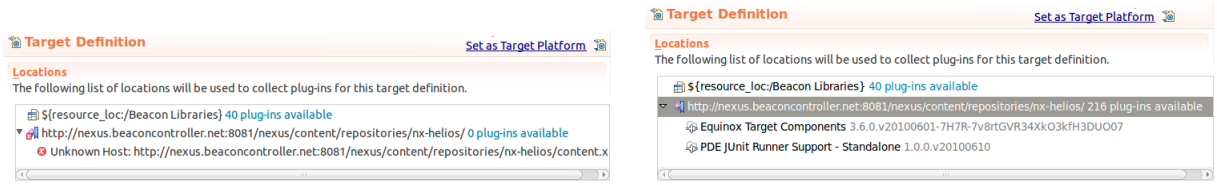
5.1.2.2 Evaluation Test

The test is run on the same device the designed application and the Cbench evaluation tool 5.1.1.1 [31] to evaluate the Throughput and Latency. With the Cbench tool have been emulated "N" Switches connected directly to the controller. Each Switch has a million connected devices and consequently these devices are connected directly to the controller.

There have been performed several types of simulations, increasing the number of switches available ("N" Switches) and therefore increasing the number of devices connected to the controller. For each switch added, increases the number of devices ("N" Devices) connected in a million, in order to send the maximum number of Messages PacketIn to evaluate controller performance.

5.1.3 Beacon OpenFlow Controller Evaluation

Here, at this point, is explained the procedure required to evaluate the Beacon Controller which has been designed as explained in the previous chapter. Initially, there was a problem with the proxy software hosting. Exactly, the problem was: "Cannot satisfy dependency" with the helios repository in the eclipse. To fix this problem, you need to do the followings actions: exit eclipse, delete the workspace directory, start Eclipse again, create a new workspace, and import your existing projects, then open the target (see Figure 5.2) and set it again, and it should work.



(a) (b)

Figure 5.2: Beacon Target

Also, it was necessary to export the Beacon code from the Eclipse to run the Controller from a terminal and thus consume less CPU resources, necessary step to make the test more accurately because it used all the resources available to the test in order to obtain results more reliable, accurate and reliable.

To export the code has followed the rules or instructions of the tutorial Beacon[37], checking that the configuration is correct. Furthermore, in our case, it was necessary to download the delta pack, including it in the main.target to solve the following problem: Processing inclusion from feature org.eclipse.equinox.executable. Unable to find plug-in: org.eclipse.equinox.launcher.gtk.linux. After it has had to reconfigure the main.target again and check that the necessary plugin, which initially was not included, is now included and that the problem is resolved.

It was necessary to set different parameters of the controller to correctly perform the test or evaluation. The first file to configure is the **Beacon.Properties File**. You must create inside the Beacon folder the Beacon.Properties file with the following parameters (see Table 5.1). Threads changed in beacon.properties via controller.threadCount. It is generally advisable to set the ThreadCount parameter to match the number of cores in your system.

controller.threadCount=1
controller.switchRequirementsTimer=false

Table 5.1: Beacon.Properties

XX:+AggressiveOpts
Xmx3000M

Table 5.2: Beacon.Ini

The second and final configuration is related to the **Beacon.Ini File** and requires modify the Beacon.Ini file by entering the following parameters (see Table 5.2).

5.1.4 Trema OpenFlow Controller Evaluation

In this chapter is evaluated the Full-Stack OpenFlow Framework for Ruby/C Trema OpenFlow Controller. In this point is explained the necessary procedure to evaluate the Trema controller. First of all, before downloading the code, is necessary install the prerequisites because Trema requires several third-party software. The software to install are: gcc make ruby ruby-dev irb sudo file libpcap-dev libsqlite3-dev. After, Download Trema source tree and run its build script to install Trema: build.rb.

There have been two different assessments for the Trema Controller because there is confusion. Initially the intention was evaluate the Trema application with the Trema Cbench tool but was found with the following explanation that said that you couldn't assess the designed application with Cbench. The reason is 5.3:

Cbench suite requires single-purpose controllers dedicated to Cbench, and does not work with general-purpose controllers like learning switch.

Table 5.3: Cbench suite

Regarding this confusion, Cbench suite requires single-purpose controllers dedicated to Cbench, and does not work with general-purpose controllers like learning switch, they recommended using for this reason the Cbench-Switch.rb code for the evaluation. Finally, thanks to the support offered by Trema, the confusion is resolved. The clarification regarding the use of the tool Cbench is 5.4:

Cbench is a micro-benchmark for checking Flowmods per second performance. For that purpose the controller measured with Cbench must send only Flowmods and not other messages. Another application (Learning switch) can work with Cbench as you said but the messages it sends are almost packet-outs (flooding).

Table 5.4: Cbench Confusion Clarification

In addition to this confusion there was a limitation with the external tool Cbench. This limitation was as follows 5.5:

The Cbench external tool distributed under git://gitosis.stanford.edu/oflops.git is not compliant with OpenFlow 1.0.0. Thus, it does not work with controllers developed on top of Trema.

Table 5.5: Cbench external tool: Limitation

Regarding this limitation, the Cbench tool is designed for OpenFlow v0.8.9 and not speaking OpenFlow v1.0. So, if you connect original Cbench, Trema parser correctly finds protocol error and disconnects the session. For this reason, for the evaluation of Trema Controller has used the Cbench tool that comes with Trema because the Cbench distribution that comes with Trema is modified for v1.0 actually.

Resolved the confusion and thanks to the Trema developers who have made a patch so you can evaluate a controller with the Cbench external tool, has been evaluated the application following the same procedure to evaluate the application or Beacon Controller.Therefore, there have been two different evaluations:

- The Cbench-Switch evaluation with the Trema Cbench tool
- The Design application evaluation with the external Cbench tool

5.1.4.1 Cbench-Switch evaluation with the Trema Cbench tool

The Cbench-Switch.rb code to evaluate is located in src/examples/cbench-switch and is as follows:

```
# A simple openflow controller for benchmarking.
# Author: Nick Karanatsios <nickkaranatsios@gmail.com>
# Copyright (C) 2008-2012 NEC Corporation
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License, version 2, as
# published by the Free Software Foundation.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along
# with this program; if not, write to the Free Software Foundation, Inc.,
# 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

class CbenchSwitch < Controller
  def packet_in datapath_id, message
    send_flow_mod_add(
      datapath_id,
      :match => ExactMatch.from( message ),
      :buffer_id => message.buffer_id,
      :actions => ActionOutput.new( :port => message.in_port + 1 )
    )
  end
end
### Local variables:
### mode: Ruby
### coding: utf-8-unix
### indent-tabs-mode: nil
### End:
```

To execute the Cbench tool and the Trema Controller you must run the following commands:

- How to Run the Controller

./trema run ./src/examples/cbench-switch/cbench-switch

- How to Run the Trema Cbench tool

./objects/oflops/bin/cbench -c localhost -p 6633 -m 10000 -l 6 -s 6 -M 1000000 –delay 3000

5.1.4.2 Evaluation of the design application with the external Cbench tool

To execute the Cbench tool for evaluate the Controller and the Trema Controller you must run the following commands:

- How to Run the Controller

./trema run ./src/examples/learning Switch/learning-Switch.rb

- How to Run the Cbench external tool

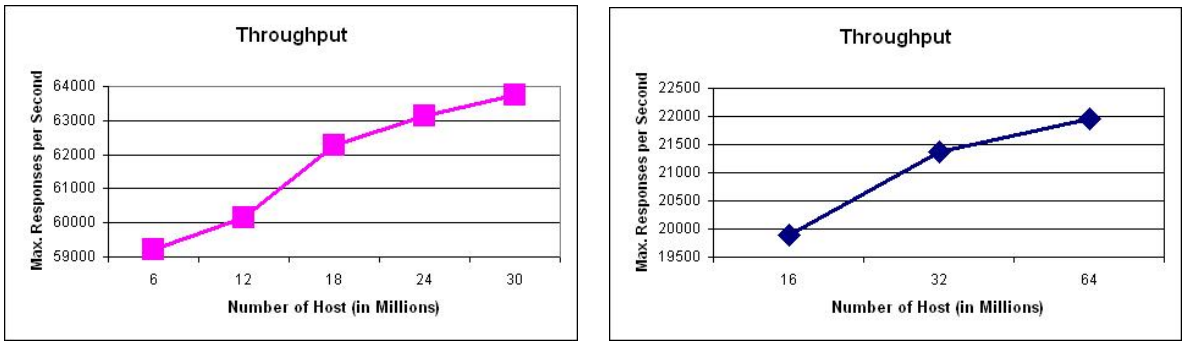
cd /home/romero/Descargas/oflops/cbench/
./cbench -c localhost -p 6633 -m 10000 -l 6 -s 6 -M 1000000 –delay 3000

5.1.5 Results of Evaluation

Throughput and Latency are the main two parameters that are more important or determinants for the performance of any application. Along with these two parameters is evaluated the code, comparing both codes and detailing the number of lines of each code and finally explains the reasons or causes why performance can be affected or limited.

5.1.5.1 Throughput

The first parameter to evaluate is **the Throughput**. As you can observed, if you increase the number of switches in the system, this produces an increased throughput, i.e. , an increasing of the requests per second number and the corresponding increase in the number of responses (see Figure 5.3). This result is quite logical, a larger number of switches, (each switch with 1 million MAC addresses), greater number of requests, (a request for each MAC address) and consequently greater number of responses (a response for each request).



(a) Beacon (b) Trema

Figure 5.3: Throughput Performance Tests

Thus, for example, with the Beacon controller you can see a significant increase in responses per second when you double the number of switches in the system, from 6 initial Switches to 12 Switches and finally to 24 Switches (see Figure 5.3(a)).

Regarding the evaluation of Trema controller performance, one must distinguish two cases:

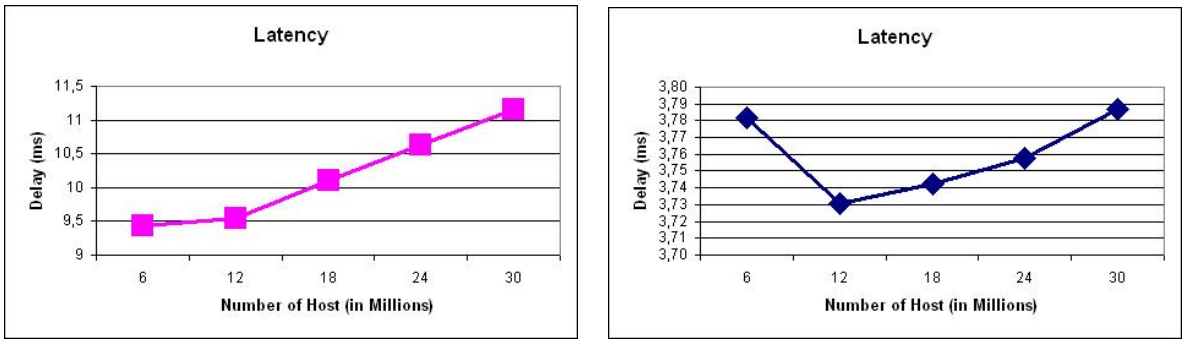
- Throughput evaluation with the Cbench external tool.
- Throughput evaluation using the Cbench tool that Trema provided.

With respect to the Throughput evaluation of Trema Controller with the Cbench external tool say that it was impossible to evaluate the performance for the following reason: have been obtained in numerous tests invalid values, causing in the test results not realistic. Regarding the performance evaluation using the Cbench tool that the controller provides, you can be seen that an increasing number of switches in the system cause an increase in the number of responses per second (see Figure 5.3(b)). This behaviour is similar that the Beacon Controller behaviour. But if you compare the behavior of the performance evaluation of Trema Controller with the Beacon OpenFlow controller, you can see that the latter has a better performance because it is capable of supporting twice of requests that the Trema Controller.

5.1.5.2 Latency

Respect to latency there are two cases:

Comparison between Trema and Beacon Controller: there is a remarkable difference between the controllers. So you can see that there is a difference between the latency of the Trema and Beacon controller, using in both cases Cbench external tool to evaluate both controllers (see Figure 5.4). It can be seen in the graphs that the latency of the Beacon controller is superior to Trema Controller (see Figure 5.4(a)). This difference is due to the programming language. Remember that while the Trema controller is based on Ruby or C, the Beacon Controller is based on Java. The latter, in terms of speed is slower and consumes more resources (memory, CPU, ..) due to its characteristics compared with native compiled languages such as C or C++.



(a) Beacon OpenFlow Controller

(b) Trema OpenFlow Controller

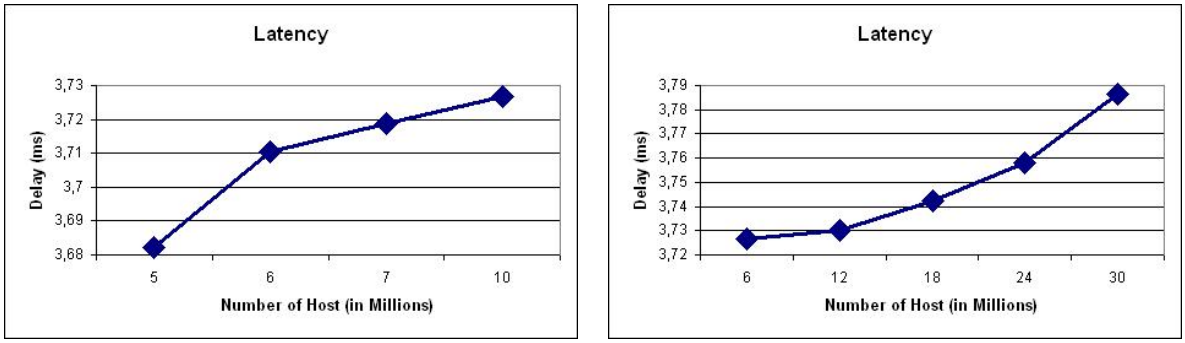
Figure 5.4: Latency Performance Tests

Verify the Trema Controller behavior It can be observed in the evaluation of Trema Controller that the latency has a minimum value when there are 12 million connected devices and not 6 million connected devices.

The logical behavior would have a minimum latency value with the minimum number of connected devices, in these evaluation 6 million connected devices, and that an increased number of connected devices cause an increase in latency. Thus, with 12 million connected devices should have a higher latency that with 6 million connected devices.

To verify that the Trema Controller has a logical and expected behavior and therefore the latency with 6 million devices is less than the latency with 12 million devices and not the opposite, a new test is performed with 5, 6, 7 and 10 million devices (see Figure 5.5(a)).

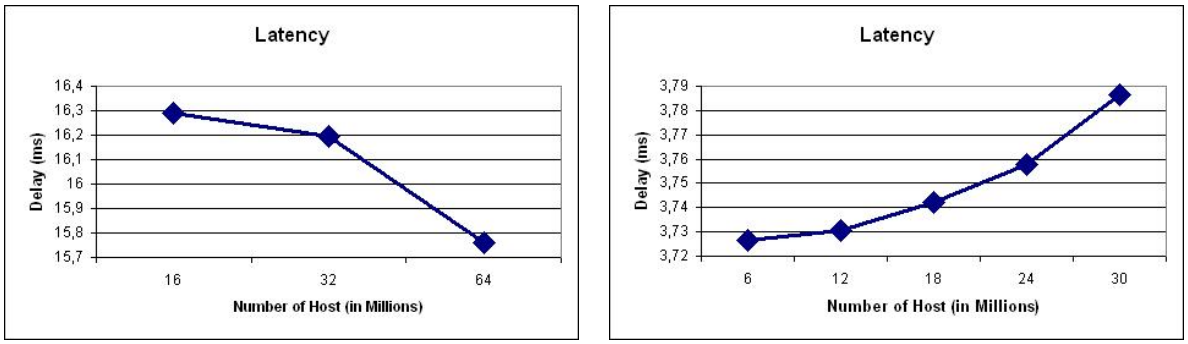
As can be seen in the test performed (see Figure 5.5(b)), the latency has a minimum value with the minimum number of connected devices and with the increased number of network devices there is an increase in latency. This behavior is logical and expected.Finally, the evaluation of Trema controller would be:



(a) Trema Latency Performance Tests (b) Trema Latency with external Cbench tool

Figure 5.5: Trema Latency Performance Tests

Comparison between different Trema Controller: there is a big difference in terms of latency depending on which tool is used to evaluate the controller. For example in the case of evaluating the code Cbench-Switch with the Cbench tool that provides Trema, is obtained the maximum latency with the lowest number of switches available on the network and increasing the number of devices or switches in the network, the latency decreases (see Figure 5.6(a)).



(a) Trema Latency Performance Tests (b) Trema Latency with external Cbench tool

Figure 5.6: Latency Performance Tests II

If the designed application is evaluated by Cbench external tool, is obtained a latency values that increases when increasing the number of switches in the network (see Figure 5.6(b)). This last evaluation shows the expected behaviour: increasing the number of devices, this will produce a load increase in the controllers, causing a latency increase.

5.1.5.3 Others Parameters that can affect the Performance Evaluation

Below, you can see various aspects that may affect or limit the performance of any application. These aspects are the following:

- The Code.
- The Controller design (operating mode).
- The programming language used to develop the controller.
- The operating system on which various controllers are implemented.

The code: the Beacon code is much longer than the Trema code, being the Controller Trema much more compact. The number of lines of code to design the same application is: 303 lines for Beacon Controller and 88 lines for the Trema Controller.

The controller design is the main key aspect that can affect the performance evaluation. So depending on how you design the controller, always based on the "Networks Design Decision", may or may not affect performance and therefore its evaluation. Thus, a controller designed with a reactive flow configuration will have much more influence on performance that a proactive flow configuration or setup. Finally, as a conclusion that in terms of performance, the controllers are so much faster than the switches and the controllers will likely not be the bottleneck in your system.

The programming language can be another limitation that affects the performance of any application. Thus, for example, the main disadvantage of Java is speed that it can affect to the application performance. Although Java's ability for producing portable, architecturally neutral code is desirable, the method used to create this code is inefficient. Once Java code is compiled into byte code, an interpreter called a Java Virtual Machine, specifically designed for computer architecture, runs the program.

Specifically, if we talk about performance, Java can be perceived as significantly slower and more memory-consuming than natively compiled languages such as C or C++. One possible solution is, if you find the performance to have a high variance, increasing the RAM permitting. The reason for the performance to have a high variance is because it is possible that the JVMs garbage collector is thrashing internally due to a lack of memory, it should on average be consuming only half of CPU time, tools such as JVisualVM can attach to the running JVM and show you what is happening.

Finally, depending on the device where you do the performance testing may obtain more accurate and precise data or results. Thus, for example, making use of the taskset tool that provides Ubuntu, will get more accurate results if a controller is evaluated in a device having a processor with 2 Cores and each Core multiple Threads (between 2 and 8) that if a controller is assessed on a machine with a single core and a single Thread.

5.2 Evaluation Switch Application of different Controllers

In this section is made an evaluation of the learning-Switch application.This application has been implemented with the following controllers: Beacon, Trema, NOX and Maestro.

Initially are defined the NOX and Maestro Controllers, describing their main characteristics and the requirements for installation and execution. Regarding Trema and Beacon controllers, these are the two types of controllers on which it has done before a specific application and therefore not defined in this chapter because both controllers have been previously defined in the previous chapter 4. Finally is performed the Controller evaluation and explains the results.

5.2.1 Scenario

The second scenario 5.7 used to evaluate the learning-Switch application is composed by a Controller and "N" Switches connected directly to the controller. Each Switch has a million connected devices and consequently these devices are connected directly to the controller. Switches and devices have been emulated with the Cbench tool (see 5.1.1.1) [31].

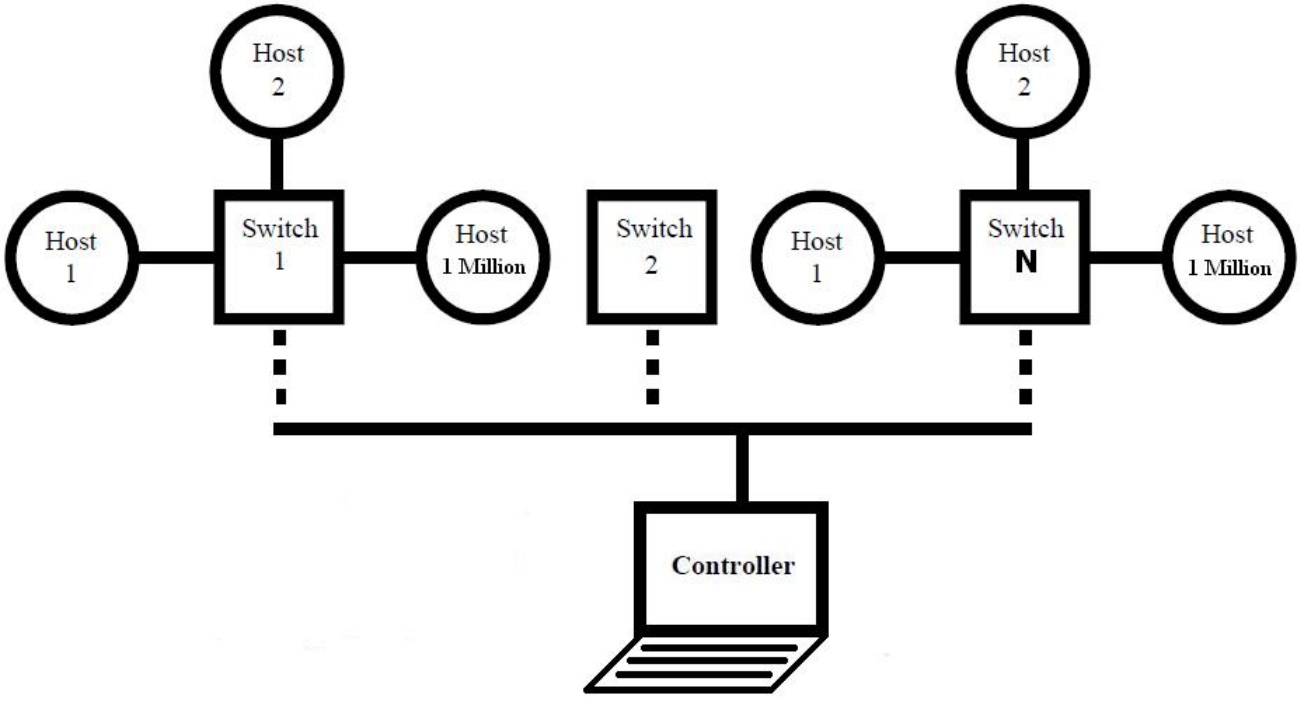


Figure 5.7: Scenario used to evaluate the learning-Switch application

5.2.2 Experiment setup and methodology

5.2.2.1 Characteristics to Evaluate

The throughput or network throughput is the number of transactions per second that an application can handle. There are different form of measuring this parameter, one would be using bits per second (bit/s or bps) and another way to measure the throughput performance would be through data packets per second.

Latency in a packet-switched network is the time required for a packet to arrive at its destination through the network. The delay can be caused, for example, by the physical environment or devices forming the network (Switches, routers...). There are two ways to measure the latency: the first is to measure the time it takes for a packet from source to destination (One-Way) and the second is to measure the time it takes a packet to go and return (Round-Trip). This last form of measuring the latency, Round-Trip, is the most used because it allows a single device to measure the latency of a network.

5.2.2.2 Evaluation Test

The test is run on the same device the learning-Switch application and the Cbench evaluation tool 5.1.1.1 [31] to evaluate the Throughput and Latency. With the Cbench tool have been emulated "N" Switches connected directly to the controller. Each Switch has a million connected devices and consequently these devices are connected directly to the controller.

There have been performed several types of simulations, increasing the number of switches available ("N" Switches) and therefore increasing the number of devices connected to the controller. For each switch added, increases the number of devices ("N" Devices) connected in a million, in order to send the maximum number of Messages PacketIn to evaluate controller performance.

5.2.3 NOX OpenFlow Controller

NOX [6] is a network control platform, an OpenFlow controller for developing management functions for enterprise and home networks. According to the definition provided by the NOX Wiki [6], the NOX OpenFlow Controller is based in C++ or Python programming language and provides a high-level programmatic interface upon which network management and control applications can be built. The current version contains a set of example applications and some built in libraries which provide useful network functions such as host tracking and routing.

The NOX Controller runs on a single device and manages the forwarding tables of multiple switches. NOX is able to do this in a scalable manner by operating on network flows (as opposed to every packet). For each new flow on the network, the first packet is sent to NOX which passes it to interested applications.

NOX exports a programmatic interface on top of which multiple network programs (which we call applications) can run. These applications can hook into network events, gain access to traffic, control the switch forwarding decisions, and generate traffic. The applications can then: determine whether (and how) to forward the flow on the network; collect statistics; modify the packets in the flow (e.g. adding a VLAN tag); or view more packets within the same flow to gain more information.

Applications can be written in C/C++ and/or Python. For example, there are applications that have already been built on NOX with the aim of reconstruct the network topology and track hosts as they move around the network.

5.2.3.1 Primary NOX Goals

- To provide a platform which allows developers and researchers to innovate within home or enterprise networks using real networking hardware. Developers on NOX can control all connectivity on the network including forwarding, routing, which hosts and users are allowed etc. In addition, NOX can interpose on any flow.
- To provide usable network software for operators. The current release includes central management for all switches on a network, admission control at the user and host level, and a full policy engine.

5.2.3.2 NOX Versions

There is always two versions of NOX that are supported:

- **Master** where things are relatively stable and the majority of the commits are bug fixes.
- **Destiny** where "random" things are committed and tried out.

5.2.4 Maestro OpenFlow Controller

Maestro [40] is an operating system that according to their definition, the maestro platform definition [4], provides interfaces for implementing modular network control applications to access and modify state of the network, and coordinate their interactions. Maestro is a platform for achieving automatic and programmatic network control functions using these modularized applications.The programming framework of Maestro provides interfaces for:

- Introducing new customized control functions adding modularized control components.
- Maintaining network state on behalf of the control components.
- Composing control components by specifying the execution sequencing and the shared network state of the components.

Moreover, Maestro tries to exploit parallelism within a single machine to improve the system's throughput performance. The fundamental feature of an OpenFlow network is that the controller is responsible for every flow's initial establishment by contacting related switches. Thus the performance of the controller could be the bottleneck. In designing Maestro we try to require as little effort from programmers as possible to manage the parallelization.

5.2.4.1 Functionality Structure

Any OpenFlow controller will share these similar functionalities, although it may implement them in different ways. Maestro OpenFlow Controller is composed of the following functionality structure (see Figure 5.8).

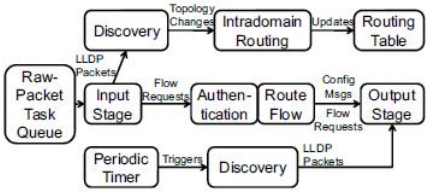


Figure 5.8: The overall structure of Maestro [40]

Maestro sends and receives OpenFlow messages to and from network switches via TCP connections. The "Input Stage" and "Output Stage" handle low level details of reading from and writing to socket buffers, and translating raw OpenFlow messages into and from high level data structures. These low level functionalities stay fixed with a particular version of the OpenFlow protocol. Other high level functionalities may vary and are implemented in modules called "applications" in Maestro. Programmers can flexibly modify the behaviour of these applications, or add new applications to meet different goals. These applications in the functionality structure are: "Discovery", "IntradomainRouting", "Authentication" and "RouteFlow".

Discovery application periodically sends out and receives probing messages to the neighbours of each switch when switches join the network. With this method, the application can discover the topology of the network.

When "Discovery" finds topology changes, it initiates the "IntradomainRouting" application to update the "RoutingTable" data structure accordingly. This "RoutingTable" contains the all-pair-shortest paths for the entire network, and will be used by the "RouteFlow" application to find paths for flow requests. But before, when Maestro receives a flow request from a switch, this request will be first checked against security policies in the "Authentication" application. Only when allowed, the "RouteFlow" application will try to find a path for this request, and generate one flow configuration message for each of the switches along the chosen path. After that, all flow configuration messages will be sent to their destination switches, and the flow request packet itself will be sent back to the origin switch.

All three application execution paths run in parallel. Maestro makes sure that if the "RoutingTable" data structure gets updated while the "RouteFlow" application is already running, "RouteFlow" continues with the older version of "RoutingTable" to generate consistent results. Next time when "RouteFlow" executes it will start to use the latest "RoutingTable".

5.2.4.2 Programming Language

Maestro is based on Java programming language. It has been chosen Java to be the programming language for Maestro OpenFlow Controller for the following reasons.

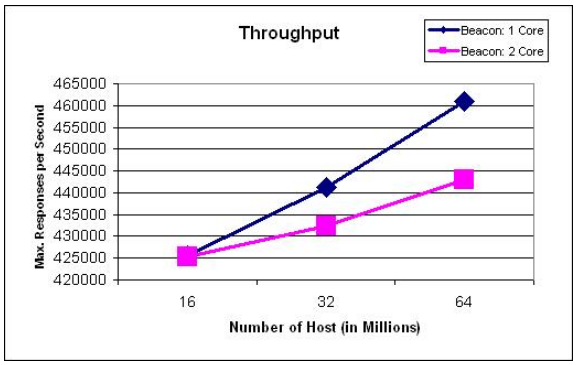
- Java programming language are considered to be easy to write and the Java programs are considered to be easy to maintain and these are more secure, so it is relatively easier to debug and to maintain.
- Java can support dynamic loading of applications and data structures without recompiling and restarting the whole system more easily, so it will make Maestro very flexible to extend.
- It is very easy to migrate Java code to different platforms as long as there is Java Virtual Machine support on that platform. Usually the code needs very little or even no modification to work on another platform, which makes Maestro more flexible.
- Java is considered to be less efficient than C or C++, but we argue and show by evaluation that, Maestro can achieve overall good performance and scalability by incorporating the right design and optimization techniques.

5.2.5 Results of Evaluation

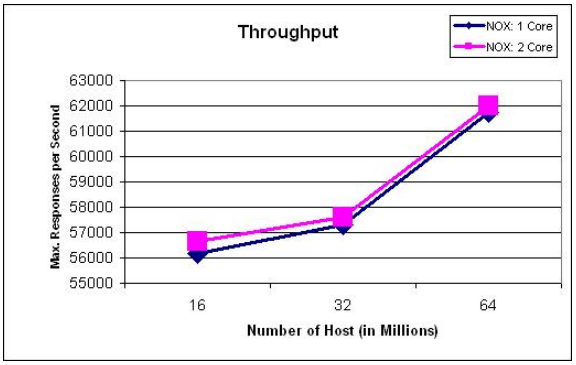
5.2.5.1 Throughput

The first parameter to evaluate is the throughput. As you can see,regardless of the threads number used, if the switches number in the system increase, this cause to a throughput increase, ie, an increase of the requests number per second and logically, an increase in the responses number.

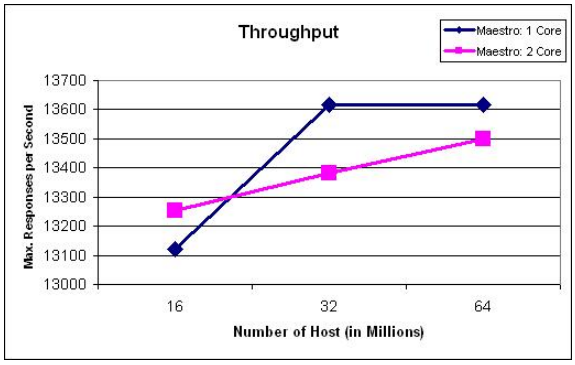
This behavior is expected, since the behavior and operation of the system is as follows: each switch has a number of MAC addresses (exactly in the test, each switch has 1 million of MAC addresses) and makes a request for each MAC address. Therefore increasing the number of switches, emulated in the system, will cause an increase in the number of requests and consequently a greater number of responses that will produce an increased throughput.



(a) Beacon Throughput



(b) NOX Throughput



(c) Maestro Throughput

Figure 5.9: Throughput Performance Tests

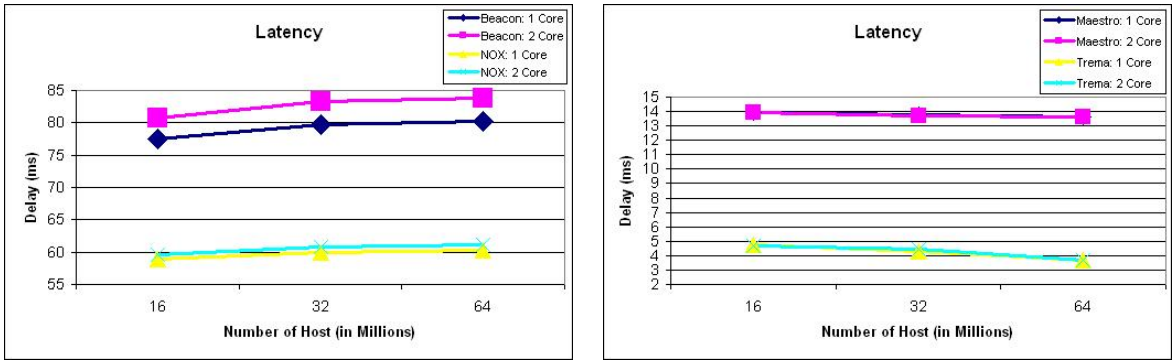
As shown in the Figure above 5.9, the controller that processes a greater number of requests per second is the controller Beacon, followed by the NOX controller and finally the Master Controller. This allows us to affirm, in terms of performance, that between the 3 controllers that have been evaluated, which offers better performance is the Beacon.

These data or results are the same or similar with the following evaluation: Controller Performance Comparisons [47]. As can be seen, a smaller scale, the controller which has a better throughput in both cases is the Beacon Controller, followed by the NOX controller and finally the Maestro Controller. Similarly, you can see that in both assessments is produced a significant increase in responses per second when you double the number of switches in the system, initial from 16 Switches to 32 Switches and finally to 64 Switches or in the case of evaluation "Controller Performance Comparisons" [47] when increasing the number of Threads.

There is an exception with the Trema Controller evaluation. It has been impossible to assess the Trema Controller Throughput for the following reason: invalid results are obtained in numerous tests, causing that the test results were unrealistic.

5.2.5.2 Latency

In terms of **Latency**, there is a big difference between the controllers. So you can see, there is a large difference between the latency of Beacon controller and the rest of controller tested, especially the Maestro Controller. Both controllers are based on Java so this does not affect the programming language.



(a) Beacon and NOX Latency

(b) Maestro and Trema Latency

Figure 5.10: Latency Performance Tests

Also, there is a considerable difference in terms of latency between Beacon and Maestro controllers and the other two controllers: NOX y Trema. Remember that the controller Trema is based in Ruby or C, the NOX controller is based on C / C + + and / or Python and Beacon and Master controllers are based on Java. This latter programming language in terms of speed is slower and consumes more resources (memory, CPU, ..) due to its characteristics than natively compiled languages such as C or C++.

5.2.5.3 Other Parameters that can affect the Performance Evaluation

There are different factors influencing the performance of an application, as explained in the previous Chapter 4. These factors are: **the code**, **the Controller design** (operating mode), **the programming language used to develop the controller** and **the operating system** on which various controllers are implemented.

The third parameter to evaluate after **the throughput** and **latency** is **the code**. Between the different Controllers tested, the code that has a greater extension is the NOX controller code, controller that is based on the C programming language. The learning-Switch application based on the different OpenFlow controllers have the following extension (see Figure 5.6). As you can be seen, between the 4 different codes, the Trema code is more compact and efficient because with only 88 lines implements the same application (learning Switch) that the NOX controller with 222 lines of code (see Figure 5.6).

OpenFlow Controller	Programming Language	Extension or Number of Lines
NOX Controller	Based on C	222 lines
Beacon OpenFlow Controller	Based on Java	172 lines
Maestro Controller	Based on Java	144 lines
Trema OpenFlow Controller	Based on C/Ruby	88 lines

Table 5.6: OpenFlow Controllers: Code Extension

Chapter 6

Conclusions

It was a difficult and complicated project, being a new technology unknown to me, but at the same time very interesting because this technology is the technology that is based on future networks. Therefore, initially it has been necessary situate within the context of the virtualization networks. It is for this reason that there has been a thorough and in-depth study on network virtualization as well as multiple virtualization techniques available today and finally about the OpenFlow technology and its ecosystem.

6.1 OpenFlow Impact

There are many potential benefits to using OpenFlow [41] but where will have the largest impact will be on future network generations. The following explains the impact of OpenFlow on society, economy and environment (sustainable development).

6.1.1 Social and ethical aspects

The deployment of OpenFlow technology offers many benefits to business and personal level. The benefits that businesses can achieve through an architecture based on OpenFlow SDN are: Centralized Control of multi-vendor environments, reduced complexity through automation, higher rate of innovation and more granular network control.

As a user, one of the benefits that provide the implementation of OpenFlow network by manufacturers and service providers is the ability to provide all users a network infrastructure that adapts to the needs of end users, centralizing control network and providing status information to higher level applications.

At present, the users must explicitly indicate that a specific configuration and the network may or may not support this configuration, which can cause a certain inconvenience to the end user, such as delays and interruptions, which degrade the user experience. With the development and implementation of OpenFlow technology, based on the Software Defined Networking, is the application itself that uses the network information to detect available bandwidth on the network in real time and automatically configures the application according to the network information.

6.1.2 Economic and environmental aspects

The Open Networking Foundation (ONF) is working to standardize OpenFlow technology and is promoting the concept. This organization has a great support, among them service providers such as Deutsche Telekom and Verizon, manufacturers such as Cisco, Hewlett Packard and many data center operators [42].

With the implementation of the OpenFlow network, the network equipment manufacturers and service providers have a new market opportunity being one of the main reasons to support OpenFlow community. From an economic point of view, they both like the idea that proposes OpenFlow to reduce network costs and simplify the introduction of new services, centralizing the intelligence of the routers or switches. Therefore, both are open to the idea of adding OpenFlow feature in network devices. The only requirement is that the implementation of OpenFlow networks involves modifying or replacing traditional devices Switches and Routers-that are part of the current network. Currently, this requirement is being implemented, because both network equipment manufacturers and suppliers of network equipment are actively adding OpenFlow function to their products, but are not a hardware change.

6.1.3 Ecologically sustainable development

At present it is producing an exponential increase in demand for network traffic that causes a rapid growth in data centers to satisfy this demand. As a result, data centers consume a large amount of energy and emit a lot of greenhouse gases causing great concern in the owners and administrators of data centers and many other people (politicians, environmentalists). It is possible to reduce energy consumption and consequently the emission of greenhouse gases in the network using OpenFlow [43]. Remember that the main objective of OpenFlow is to divide the data path from the control path and this division allows the creation of independent intelligent control mechanisms. Through these intelligent control mechanisms is possible to reduce energy consumption and consequently reduce the emission of greenhouse gases. One of the application areas for OpenFlow is its use in data centers. There are different projects in order to improve energy efficiency and reduce consumption and save energy - all these projects based on OpenFlow- in data centers.

One of these projects is ECODANE [44]. ECODANE optimizes energy consumption of network components by designing an intelligent network control system that dynamically adapts the set of active network components corresponding to the total traffic going through the data center. The optimizer module is accompanied by a load balancing routing module to guarantee the availability of a data center. Another project is ElasticTree [45]. This project is an energy manager of the entire network, which dynamically adjusts the set of active network elements — links and switches—to satisfy changing data center traffic loads. With ElasticTree is possible save up to 50 percent of network energy, while maintaining the ability to handle traffic surges. A third project is Response [46]. This project has also been developed to reduce power consumption. The main objective of response is to identify a few energy-critical paths off-line, install them into network elements, and use a simple online element to redirect the traffic in a way that enables large parts of the network to enter a low-power state.

6.2 Evaluation Design Application

Concerning the applications design, this process has been an enriching process and therefore has made a detailed investigation about OpenFlow and has made a study concerning the devices necessary to implement an OpenFlow network in order to design our application. In addition to know the OpenFlow technology, the design of these two applications has enabled, among other things, know the Ruby programming language, a new programming language for me on which he had never made any application.

It has obtained a logical result. In both controllers, Beacon and Trema OpenFLow Controller, you can seen that if you increase the number of switches in the system, this produces a throughput and latency increased , i.e. , an increasing of the requests per second number and the corresponding increase in the number of responses. This behavior is quite logical, a larger number of switches greater number of requests and consequently greater number of responses.

Throughput is a parameter to evaluate. If you compare the behavior of the performance evaluation of Trema Controller with the Beacon OpenFlow controller, you can see that the latter, the Beacon OpenFlow Controller, has a better performance because it is capable of supporting twice of requests that the Trema Controller (see Table 6.1 and Table 6.2).

Number of Switches	Responses per Second
6	59.000
12	60.000
18	62.000
24	63.000
30	64.000

Table 6.1: Beacon: Responses per Second

Number of Switches	Responses per Second
16	21.000
32	21.500
64	22.000

Table 6.2: Trema: Responses per Second

Latency is the second parameter to evaluate. The difference in terms of latency between Trema and Beacon Controller is that the Trema OpenFlow Controller has a lower latency compared to the Beacon Controller. This difference may be due to the programming language. Remember that while the Trema controller is based on Ruby or C, the Beacon Controller is based on Java. The latter, in terms of speed is slower and consumes more resources (memory, CPU, ..) due to its characteristics compared with native compiled languages such as C or C++.

Also, there is a big difference in terms of latency depending on which tool is used to evaluate the Trema controller. There are 2 options: one option is to use the Cbench tool that provides Trema and the second option is through the Cbench external tool. If **the designed application** is evaluated by **Cbench external tool**, this evaluation shows the expected behaviour: increasing the number of devices, this will produce a load increase in the controllers, causing a latency increase.

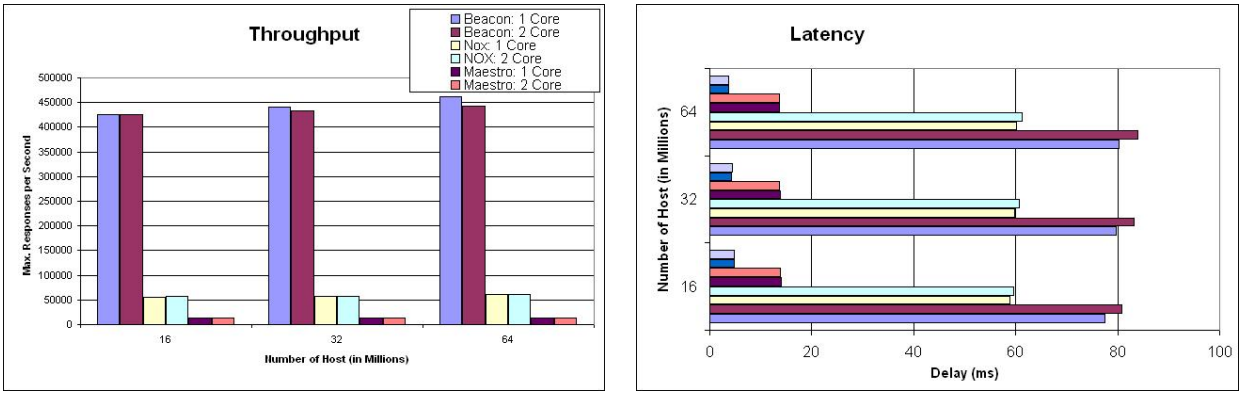
We found many and various problems when deploying the designed application, for example, the ignorance of the Ruby programming language in which the Trema OpenFlow Controller is based or the problems with the SNAC OpenFlow Controller due to the inabil-

ity to install the requirements required to operate (reason that it was decided not to make their implementation). Finally, with the Beacon OpenFlow controller had several problems, between them a problem with a particular plug-in needed to export the code. This last problem, for example, was solved by downloading a file (Delta Pack) and including it in the project.

In general, all the problems found regarding the design and implementation of the application has been solved thanks to information available Internet and with the support from the creators and designers of the different controllers.This last is for me the most serious problem that I have found. In some controllers, there is very limited support from some developers and designers. Thus, for example, in the SNAC controller, the support is limited. Compared with the support that provides the SNAC controller, the Beacon controller has a platform where users can ask your questions and these questions are resolved clearly and concisely in a short time by the creators and designers. In my opinion, between all controllers studied in this project, Beacon is one that offers better support.

6.3 Evaluation Learning Switch Application

The learning-Switch application evaluation has been expected being the Beacon Controller, the Controller that has better performance and higher latency, this last because it's implemented with Java and this type of programming language, in terms of speed, is slower and consumes more resources (memory, CPU, ..) due to its characteristics compared with native compiled languages such as C or C++. These results obtained are similar to the results of the evaluation that provides OpenFlow, the "Controller Performance Comparisons" [47]. Thus, comparing the "Controller Performance Comparisons" [47] evaluation with the evaluation made in this project, is observed a similar performance, in which the controller that has a better performance is the Beacon Controller (see Figure 6.1(a)). Furthermore, as has been seen in latency evaluations made in this project, the Beacon controller is the controller that has a higher latency (see Figure 6.1(b)). Otherwise, the Trema Controller has a lower latency.



(a) Throughput Performance Test

(b) Latency Performance Test

Figure 6.1: Final Performance Test

Among the many problems found with the installation and implementation of learning Switch application, highlight the problem that has occurred during the NOX controller instal-

lation. This controller requires a minimum of memory that the device where it was to install and deploy did not have. Therefore, it was necessary to find a new device with better features (specifically memory) to install and implement the controller.

6.4 Future work

Looking to the future, perform the evaluation of various controllers in order to obtain better results with a computer with a processor that has multiple cores and each core has several threads. With this device you could implement an evaluation with for example 4 cores and 8 threads in each core. This test would provide us better results, more consistent with those provided on the "Controller Performance Comparison" [47].

In addition, run the controller and the Cbench tool with the utility taskset [29] that Ubuntu provides. This tool "taskset", according with their definition [30], is used to set or retrieve the CPU affinity of a running process given its PID or to launch a new COMMAND with a given CPU affinity. CPU affinity is a scheduler property that "bonds" a process to a given set of CPUs on the system. The Linux scheduler will honor the given CPU affinity and the process will not run on any other CPUs.

Finally, along with the different types of controllers evaluated in this project, today there are many other types of controllers. Therefore, could do a study on these new controllers and their corresponding evaluation. The following table 6.3 shows the different types of controllers available (along with these controllers are the controllers evaluated in this project and which has made a study. These controllers are: NOX, Trema, Beacon and Maestro). For more information about other types of controllers [48].

Controller Platforms	Description
POX [49]	(Python) Pox as a general SDN controller that supports OpenFlow. It has a high-level SDN API including a queriable topology graph and support for virtualization.
Jaxon [50]	(Java) Jaxon is a NOX-dependent Java-based OpenFlow Controller.
Floodlight [51]	(Java) The Floodlight controller is Java-based OpenFlow Controller. It was forked from the Beacon controller, originally developed by David Erickson at Stanford.
NDDI -OESS [52]	OESS is an application to configure and control OpenFlow Enabled switches through a very simple and user friendly User Interface.
Ryu [53]	(Python) Ryu is an open-sourced Network Operating System (NOS) that supports OpenFlow.
NodeFlow [54]	(JavaScript) NodeFlow is an OpenFlow controller written in pure JavaScript for Node.JS.
Ovs-controller [55]	(C) Trivial reference controller packaged with Open vSwitch.

Table 6.3: Controller Platforms

Bibliography

[1] Software-Defined Networking: The New Norm for Networks. Open Networking Foundation. April 13, 2012

[2] Open Networking Foundation. Available at: https://www.opennetworking.org/

[3] Beacon Web-Site: https://openflow.stanford.edu/display/Beacon/Home

[4] Maestro Web-Site: http://code.google.com/p/maestro-platform/

[5] Trema Web-Site: http://trema.github.com/trema/

[6] NOX Web-Site: http://www.noxrepo.org/

[7] Designing and Building a Datacenter Network: An Alternative Approach with OpenFlow. Sponsored by: NEC Corporation. Rohit Mehra. January 2012

[8] White Paper Virtual Networks

[9] Virtual LAN: Applications and Technology.

[10] MAC address-based VLANs. Virtual Networks, SysKonnect

[11] A Survey of Network Virtualization. N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. 2008

[12] Types of Virtual Private Network. Available at: http://www.alliancedatacom.com/how-vpn-works.asp

[13] Devices in a Virtual Private Network. Available at:http://ptgmedia.pearsoncmg.com/images/1587051796/samplechapter/1587051796content.pdf

[14] Layer 3 MPLS VPN Enterprise Consumer Guide Version 2

[15] Network Virtualization: The new Building Blocks of Network Design. Nicholas John Lippis III. 2007

[16] Logical Network partitions on a shared network infrastructure. Available at: http://www.cisco.com/en/US/docs/solutions/Enterprise/Network_Virtualization/PathIsol.html

[17] Architecture of Network Virtualization. Available at: http://www.cisco.com/en/US/docs/solutions/Enterprise/Network_Virtualization/PathIsol.html

[18] Introduction to OpenFlow: Bringing Experimental Protocols to a Network Near You. Chris Tracy, Network Engineer. ESnet Engineering Group.

[19] How the emergence of OpenFlow and Software-Defined Networking (SDN) will change the networking landscape. Brocade. 2012

[20] Nick McKeown. Available at: http://yuba.stanford.edu/

[21] Open Networking Foundation. Available at: https://www.opennetworking.org/

[22] The OpenFlow Switch Specification. Available at: http://OpenFlowSwitch.org.

[23] OpenFlow Switch. Available at: http://yuba.stanford.edu/cs244wiki/index.php/Overview

[24] Introduction to OpenFlow: Bringing Experimental Protocols to a Network Near You!. ESnet Engineering Group. Chris Tracy, Network Engineer.

[25] OpenFlow: Enabling Innovation in Campus Networks. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner. March 14, 2008

[26] OpenFlow in Service Provider Networks. Rob Sherwood, Saurav Das, Yiannis Yiakoumis. October 2010

[27] An Experimenter's Guide to OpenFlow. Rob Sherwood. GENI Engineering Workshop June 2010

[28] MiniNet Web-Site: http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet

[29] Ubuntu: Taskset utility . Available at: http://manpages.ubuntu.com/manpages/natty/man1/taskset.1.html

[30] The Taskset Tool . Available at: http://linuxcommand.org/man_pages/taskset1.html

[31] Cbench: Controller Benchmarker. Available at: http://www.openflow.org/wk/index.php/Oflops

[32] Trema: A Brief Introduction and Tutorial. National Institute of Information and Communications Technology. Shuji Ishii, Eiji Kawai. 2011

[33] Trema: Open Source OpenFlow Controller Platform. System Platforms Research Labs., NEC Corp. 2011

[34] Trema Basic Operations. Yasunobu Chiba. 2011

[35] The C programming Language. Brian W. Kernighan and Dennis M. Ritchie. 1988

[36] Ruby in a Nutshell. A Desktop Quick Reference. Yukihiro Matsumoto. 2001

[37] Beacon: Debugging and Exporting Screencasts Web-Site: https://openflow.stanford.edu/display/Beacon/2011/09/02/Debugging+and+Exporting+Screencasts

[38] Characteristics of Java. Supplement for Introduction to Java Programming. Y. Daniel Liang

[39] OpenFlow Controller: SNAC (Simple Network Access Control). Stanford University and Big Switch Networks

[40] Maestro: A System for Scalable OpenFlow Control. Department of Computer Science, Rice University. Zheng Cai Alan L. Cox T. S. Eugene Ng

[41] How the emergence of OpenFlow and SDN will change the networking landscape. Brocade

[42] OpenFlow: The nextgeneration in networkinginteroperability. IBM. May 2011

[43] An OpenFlow-Based Energy-Efficient Data Center Approach. Michael Jarschel, Rastin Pries. University of Würzburg, Institute of Computer Science, Würzburg, Germany

[44] ECODANE - Reducing Energy Consumption in Data Center Networks based on Traffic Engineering. Truong Thu Huong, Daniel Schlossery, Pham Ngoc Nam, Michael Jarschely, Nguyen Huu Thanh, Rastin Priesy. University of Science and Technology, School of Electronics and Telecommunications, Hanoi, Vietnam.

[45] ElasticTree: Saving Energy in Data Center Networks. Brandon Heller, Srini Seetharaman†, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, Nick McKeown. Stanford University, Palo Alto, CA USA.

[46] Identifying and Using Energy-Critical Paths. Nedeljko Vasic, Dejan Novakovic, Marco Canini, Dejan Kostic, Satyam Shekhar, Prateek Bhurat. EPFL, Switzerland and IIT Guwahati, India.

[47] Controller Performance Comparisons Web-Site: http://www.openflow.org/wk/index.php/Controller_Performance_Comparisons

[48] List of OpenFlow Software Projects Web-Site: http://yuba.stanford.edu/~casado/of-sw.html

[49] Pox as a general SDN controller that supports OpenFlow. Web-Site: https://github.com/noxrepo/pox

[50] Jaxon is a NOX-dependent Java-based OpenFlow Controller. Web-Site: http://jaxon.onuos.org/

[51] The Floodlight controller is Java-based OpenFlow Controller. Web-Site: http://floodlight.openflowhub.org/

[52] NDDI -OESS: Application to configure and control OpenFlow Enabled switches Web-Site: http://code.google.com/p/nddi/

[53] Ryu: Open-sourced Network Operating System that supports OpenFlow. Web-Site: http://www.osrg.net/ryu/using_with_openstack.html

[54] NodeFlow: OpenFlow controller written in pure JavaScript. Web-Site: https://github.com/gaberger/NodeFLow

[55] Simple OpenFlow Controller reference implementation Web-Site: http://openvswitch.org/cgi-bin/ovsman.cgi?page=utilities%2Fovs-controller.8

Appendix A

Requirements

This chapter specifies and details the necessary requirements to implement the scenario, on which are designed and implemented the two applications or controllers made, one based on Java (Beacon Controller) and other applications based on Ruby (Trema OpenFlow Controller). Also on this same scenario evaluates the application learning-switch, of the following controllers: NOX, Trema, Beacon y Maestro.

Initially it has to choose the operating system and the version to use. Then used regardless of operating system the virtualization tool VirtualBox. The following table A.1 summarizes the different options available:

OS Type	OS Version	V. Software	X Server Terminal	Terminal
Windows	7 or XP	VirtualBox	Xming	Putty
MAC	10.6 Snow Leopard 10.4 Tiger 10.3 Panther 10.5 Leopard	VirtualBox	X11 or XQuartz	Terminal.app
Linux	Ubuntu 10.04	VirtualBox	X server	gnome terminal + SSH

Table A.1: Summary of the Operating System Options

It has chosen to work the Linux operating system with the following characteristics A.2.

OS Type	OS Version	V. Software	X Server Terminal	Terminal
Linux	Ubuntu 10.04	VirtualBox	X server	gnome terminal + SSH

Table A.2: Operating System

A.1 Operating System Version: Ubuntu 11.04

Ubuntu is a computer operating system based on the Debian Linux distribution and distributed as free and open source software.

It is composed of many software packages, the vast majority of which are distributed under a free software license. The main license used is the GNU General Public License (GNU GPL) which, along with the GNU Lesser General Public License (GNU LGPL), explicitly declares that users are free to run, copy, distribute, study, change, develop and improve the software. Ubuntu focuses on usability, security and stability. Ubuntu also emphasizes accessibility and internationalization to reach as many people as possible. Ubuntu includes Ubuntu Desktop that is a graphical desktop environment.

A.2 Software Virtualization: VirtualBox

VirtualBox is a general-purpose full virtualized for x86 hardware, targeted at server, desktop and embedded use. VirtualBox requires an existing operating system to be installed and it can run alongside existing applications on that host. VirtualBox is identical in terms of functionally for all of the host platforms, and the same file and image formats are used. This allows you to run virtual machines created on one host on another host with a different host operating system; you can create a virtual machine on Windows and then run it under Linux.

A.2.1 Main Features

Below, explains briefly the main features of VirtualBox's. One of these features is the portability because VirtualBox runs on a large number of operating systems. Another characteristic is that no hardware virtualization required. This means that for many scenarios, VirtualBox does not require the processor features. As opposed to many other virtualization solutions, you can therefore use VirtualBox even on older hardware where these features are not present. The characteristic third is the Guest Additions. The VirtualBox Guest Additions are software packages which can be installed inside of supported guest systems to improve their performance and to provide additional integration and communication with the host system. After installing the Guest Additions, a virtual machine will support automatic adjustment of video resolutions, seamless windows, accelerated 3D graphics, shared folders and more. Finally, the last feature: modularity and architecture. VirtualBox has an extremely modular design with well-defined internal programming interfaces and a clean separation of client and server code. This makes it easy to control it from several interfaces at once. Due to its modular architecture, VirtualBox can also expose its full functionality and configurability through a comprehensive software development kit (SDK), which allows for integrating every aspect of VirtualBox with other software systems.

Appendix B

Code Design Application

B.1 Trema Design Application

```
# Simple learning switch application in Ruby
# Author: Yasuhito Takamiya <yasuhito@gmail.com>
# Copyright (C) 2008-2011 NEC Corporation

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License, version 2, as
# published by the Free Software Foundation.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License along
# with this program; if not, write to the Free Software Foundation, Inc.,
# 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

require "fdb"

class LearningSwitch < Trema::Controller
  include Timers
  add_timer_event :age_fdb, 5, :periodic

  def start
    @fdb = FDB.new
  end

  def packet_in datapath_id, message
    port_no = @fdb.port_no_of( message.macda )
    machost2 = Mac.new("00:00:00:00:00:02")
    machost3 = Mac.new("00:00:00:00:00:03")
    machost4 = Mac.new("00:00:00:00:00:04")
    iphost2 = IP.new("10.0.0.2")
    iphost3 = IP.new("10.0.0.3")
    iphost4 = IP.new("10.0.0.4")
    porthost2 = 1
```

```
  porthost3 = 2
  porthost4 = 3
  if message.macsa.eql?(machost3) and message.macda.eql?(machost2)
      flow_mod datapath_id, message, machost4, iphost4, porthost4
      flood datapath_id, message
      fw_mod datapath_id, message, machost2, iphost2, porthost2, machost3,
          machost4, iphost3, iphost4, porthost4
  elsif message.macsa.eql?(machost2) and message.macda.eql?(machost3)
      flow_mod datapath_id, message, machost4, iphost4, porthost4
      flood datapath_id, message
      f_mod datapath_id, message, machost3, iphost3, porthost3, machost2,
          machost4, iphost2, iphost4, porthost4
  else

  end
end

def age_fdb
  @fdb.age
end

def flow_mod datapath_id, message, machost4, iphost4, port4
  send_flow_mod_add(
    datapath_id,
    :match => Match.from( message, [:dl_dst, :nw_dst] ),
    :actions => [Trema::ActionSetDlDst.new( :dl_dst => machost4 ), Trema::
        ActionSetNwDst.new( :nw_dst => iphost4 ), Trema::ActionOutput.new( :
        port => port4 )]
  )
end

def fw_mod datapath_id, message, machost2, iphost2, porthost3, machost3,
    machost4, iphost3, iphost4, porthost4
  send_flow_mod_add(
    1,
    :match => Trema::Match.new( :in_port => porthost4 , :dl_src => machost4 ,
        :dl_dst => machost3 , :nw_src => iphost4 , :nw_dst => iphost3 ),
    :actions => [Trema::ActionSetDlSrc.new( :dl_src => machost2), Trema::
        ActionSetNwSrc.new( :nw_src => iphost2 ), Trema::ActionOutput.new( :
        port => 2 )]
  )
end

def f_mod datapath_id, message, machost3, iphost3, porthost3, machost2,
    machost4, iphost2, iphost4, porthost4
  send_flow_mod_add(
    1,
    :match => Trema::Match.new( :in_port => porthost4 , :dl_src => machost4 ,
        :dl_dst => machost2 , :nw_src => iphost4 , :nw_dst => iphost2 ),
    :actions => [Trema::ActionSetDlSrc.new( :dl_src => machost3), Trema::
        ActionSetNwSrc.new( :nw_src => iphost3 ), Trema::ActionOutput.new( :
        port => 1 )]
  )
end

def packet_out datapath_id, message, port_no
```

```
    send_packet_out(
      datapath_id,
      :packet_in => message,
      :actions => Trema::ActionOutput.new( :port => port_no )
    )
  end

  def flood datapath_id, message
    packet_out datapath_id, message, OFPP_TABLE
  end
end

### 88 lines
### Local variables:
### mode: Ruby
### coding: utf-8
### indent-tabs-mode: nil
### End:
```

B.2 Beacon Design Application

```
package net.beaconcontroller.tutorial;

import java.io.IOException;
import java.math.BigInteger;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.openflow.protocol.OFFlowMod;
import org.openflow.protocol.OFMatch;
import org.openflow.protocol.OFMessage;
import org.openflow.protocol.OFPacketIn;
import org.openflow.protocol.OFPacketOut;
import org.openflow.protocol.OFPort;
import org.openflow.protocol.OFSwitchConfig;
import org.openflow.protocol.OFType;
import org.openflow.protocol.action.OFAction;
import org.openflow.protocol.action.OFActionDataLayerDestination;
import org.openflow.protocol.action.OFActionDataLayerSource;
import org.openflow.protocol.action.OFActionNetworkLayerDestination;
import org.openflow.protocol.action.OFActionNetworkLayerSource;
import org.openflow.protocol.action.OFActionOutput;
import org.openflow.protocol.action.OFActionTransportLayerSource;
import org.openflow.util.HexString;
import org.openflow.util.U16;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import net.beaconcontroller.core.IBeaconProvider;
import net.beaconcontroller.core.IOFMessageListener;
import net.beaconcontroller.core.IOFSwitch;
import net.beaconcontroller.core.IOFSwitchListener;
import net.beaconcontroller.core.IOFMessageListener.Command;
import net.beaconcontroller.packet.Ethernet;
import net.beaconcontroller.util.LongShortHopscotchHashMap;

public class Tutorial implements IOFMessageListener, IOFSwitchListener {
    protected static Logger logger = LoggerFactory.getLogger(Tutorial.class);
    protected IBeaconProvider beaconProvider;
    long Coockie = 0;

    public IBeaconProvider getBeaconProvider() {
        return beaconProvider;
    }

    public void setBeaconProvider(IBeaconProvider beaconProvider) {
```

```
        this.beaconProvider = beaconProvider;
    }

    public void startUp() {
        beaconProvider.addOFMessageListener(OFType.PACKET_IN, this);
        beaconProvider.addOFSwitchListener(this);
    }

    public void shutDown() {
        beaconProvider.removeOFMessageListener(OFType.PACKET_IN, this);
        beaconProvider.removeOFSwitchListener(this);
    }

    public String getName() {
        return "tutorial";
    }

    public void addedSwitch(IOFSwitch sw) {
        logger.info("Registering switch with DPID {}", sw.getId());
    }

    public void removedSwitch(IOFSwitch sw) {
        logger.info("Unregistering switch with DPID {}", sw.getId());
    }

    protected void sendPacketOut(IOFSwitch sw, OFPacketIn pi)
    {
        OFPacketOut po = new OFPacketOut();

        short poLength = (short)OFPacketOut.MINIMUM_LENGTH; // starting length
        po.setBufferId(pi.getBufferId()); //OFPacketOut.BUFFER_ID_NONE
        po.setInPort(pi.getInPort());

        // set actions
        OFActionOutput action = new OFActionOutput();
        action.setMaxLength((short) 0);
        action.setPort(OFPort.OFPP_TABLE.getValue());
        poLength += (short) (OFActionOutput.MINIMUM_LENGTH);

        List<OFAction> actions = new ArrayList<OFAction>();
        actions.add(action);
        po.setActions(actions);
        po.setActionsLength((short) (OFActionOutput.MINIMUM_LENGTH));

        // set data - only if buffer_id == -1
        if (pi.getBufferId() == OFPacketOut.BUFFER_ID_NONE)
        {
            byte[] packetData = pi.getPacketData();
            po.setPacketData(packetData);
            poLength += (short)packetData.length;
        }

        // finally, set the total length
        po.setLength(poLength);

        // and write it out
```

```
        try {
            sw.getOutputStream().write(po);
        } catch (IOException e) {
            logger.error("could not write packet out to switch");
        }
    }

private void flowMod (IOFSwitch sw, OFMatch match, OFMatch two_match, short
    destPort)
{
    OFFlowMod fm = new OFFlowMod();

    match.setWildcards(OFMatch.OFPFW_ALL & ~OFMatch.OFPFW_DL_DST & ~OFMatch.
        OFPFW_DL_SRC  & ~OFMatch.OFPFW_NW_DST_MASK);

    fm.setBufferId(-1);
    fm.setOutPort(OFPort.OFPP_NONE.getValue());
    fm.setType(OFType.FLOW_MOD);
    fm.setMatch(match);
    fm.setCommand(OFFlowMod.OFPFC_MODIFY);
    fm.setCookie(Coockie++);

    OFActionDataLayerSource dataLayerSource = new OFActionDataLayerSource();
    dataLayerSource.setDataLayerAddress(match.getDataLayerSource());
    OFActionDataLayerDestination dataLayerDestination = new
        OFActionDataLayerDestination();
    dataLayerDestination.setDataLayerAddress(two_match.getDataLayerDestination
        ());
    OFActionNetworkLayerDestination networkLayerDestination = new
        OFActionNetworkLayerDestination();
    networkLayerDestination.setNetworkAddress(two_match.getNetworkDestination()
        );

    OFActionOutput action = new OFActionOutput();
    action.setMaxLength((short) 0);
    action.setPort(destPort);

    List<OFAction> actions = new ArrayList<OFAction>();
    actions.add(dataLayerDestination);
    actions.add(networkLayerDestination);
    actions.add(action);

    fm.setActions(actions);
    fm.setLength(U16.t(OFFlowMod.MINIMUM_LENGTH+OFActionOutput.MINIMUM_LENGTH+
        OFActionNetworkLayerDestination.MINIMUM_LENGTH+
        OFActionDataLayerDestination.MINIMUM_LENGTH));


    try {
        sw.getOutputStream().write(fm);
    } catch (IOException e) {
        logger.error("could not write flow into switch");
    }
}
```

```
private void flowModR (IOFSwitch sw, OFMatch _match_, OFMatch finally_match,
    short port)
{
    OFFlowMod fm = new OFFlowMod();

    _match_.setWildcards(OFMatch.OFPFW_ALL & ~OFMatch.OFPFW_DL_SRC & ~OFMatch.
        OFPFW_DL_DST & ~OFMatch.OFPFW_NW_SRC_MASK);

    fm.setBufferId(-1);
    fm.setOutPort(OFPort.OFPP_NONE.getValue());
    fm.setType(OFType.FLOW_MOD);
    fm.setMatch(_match_);
    fm.setCommand(OFFlowMod.OFPFC_MODIFY);
    fm.setCookie(Coockie++);

    OFActionDataLayerDestination dataLayerDestination = new
        OFActionDataLayerDestination();
    dataLayerDestination.setDataLayerAddress(_match_.getDataLayerSource());
    OFActionDataLayerSource dataLayerSource = new OFActionDataLayerSource();
    dataLayerSource.setDataLayerAddress(finally_match.getDataLayerSource());
    OFActionNetworkLayerSource networkLayerSource = new
        OFActionNetworkLayerSource();
    networkLayerSource.setNetworkAddress(finally_match.getNetworkSource());

    OFActionOutput action = new OFActionOutput();
    action.setMaxLength((short) 0);
    action.setPort(port);

    List<OFAction> actions = new ArrayList<OFAction>();
    actions.add(dataLayerSource);
    actions.add(networkLayerSource);
    actions.add(action);


    fm.setActions(actions);
    fm.setLength(U16.t(OFFlowMod.MINIMUM_LENGTH+OFActionOutput.MINIMUM_LENGTH+
        OFActionNetworkLayerSource.MINIMUM_LENGTH+OFActionDataLayerSource.
        MINIMUM_LENGTH));

    try {
        sw.getOutputStream().write(fm);
    } catch (IOException e) {
        logger.error("could not write flow into switch");
    }
}
private void deleteFlowMod (IOFSwitch sw, OFMatch match)
{
    match.setWildcards(OFMatch.OFPFW_ALL);

    OFMessage fm = ((OFFlowMod) sw.getInputStream().getMessageFactory()

        .getMessage(OFType.FLOW_MOD))

        .setMatch(match)

        .setCommand(OFFlowMod.OFPFC_DELETE)
```

```
        .setOutPort(OFPort.OFPP_NONE)

        .setLength(U16.t(OFFlowMod.MINIMUM_LENGTH));

    try {
        sw.getOutputStream().write(fm);
    } catch (IOException e) {
        e.printStackTrace();
    }

}
private void fMod (IOFSwitch sw, OFMatch match, short destPort)
{
    OFFlowMod fm = new OFFlowMod();

    fm.setBufferId(-1);
    fm.setOutPort(OFPort.OFPP_NONE.getValue());
    fm.setType(OFType.FLOW_MOD);
    fm.setMatch(match);
    fm.setCommand(OFFlowMod.OFPFC_ADD);
    fm.setCookie(Coockie++);

    match.setWildcards(OFMatch.OFPFW_ALL & ~OFMatch.OFPFW_DL_SRC & ~OFMatch.
        OFPFW_DL_DST);

    OFActionOutput action = new OFActionOutput();
    action.setMaxLength((short) 0);
    action.setPort(destPort);

    List<OFAction> actions = new ArrayList<OFAction>();
    actions.add(action);

    fm.setActions(actions);
    fm.setLength(U16.t(OFFlowMod.MINIMUM_LENGTH+OFActionOutput.MINIMUM_LENGTH))
        ;


    try {
        sw.getOutputStream().write(fm);
    } catch (IOException e) {
        logger.error("could not write flow into switch");
    }
}

    private void initialize (OFMatch _match_,String dataLayerSource, String
        dataLayerDestination, int networkSource, int networkDestination,int
        inputPort,int icmpType)
    {
        _match_.setDataLayerSource(dataLayerSource);
        _match_.setDataLayerDestination(dataLayerDestination);
        _match_.setNetworkSource(networkSource);
        _match_.setNetworkDestination(networkDestination);
        _match_.setInputPort((short)inputPort);
        _match_.setTransportSource((short)icmpType);
    }
```

```
    private void configure (OFMatch two_match,String dataLayerDestination, int
        networkDestination,OFMatch finally_match,String dataLayerSource, int
        networkSource)
    {
        two_match.setDataLayerDestination(dataLayerDestination);
        two_match.setNetworkDestination(networkDestination);

        finally_match.setDataLayerSource(dataLayerSource);
        finally_match.setNetworkSource(networkSource);
    }

    public Command receive(IOFSwitch sw, OFMessage msg) {

        OFPacketIn pi = (OFPacketIn) msg;
        OFMatch m = new OFMatch();
        OFMatch _match_ = new OFMatch();
        OFMatch two_match = new OFMatch();
        OFMatch finally_match = new OFMatch();

        m.loadFromPacket(pi.getPacketData(), pi.getInPort());

            String DataLayerType = HexString.toHexString(m.getDataLayerType());

            if ((DataLayerType.equals("00:00:00:00:00:00:08:00"))) {
                    if (m.getNetworkSource()==167772162 & m.
                        getNetworkDestination()==167772163) {

                        initialize(_match_,"00:00:00:00:00:04","
                            00:00:00:00:00:02",167772164,167772162,3,0);
                        configure (two_match,"00:00:00:00:00:04",167772164,
                            finally_match,"00:00:00:00:00:03",167772163);

                        flowMod(sw, m,two_match, (short) 3);
                        flowModR(sw, _match_,finally_match, (short)1);
                        sendPacketOut(sw,pi);
                    }
                    if (m.getNetworkSource()==167772163 & m.
                        getNetworkDestination()==167772162) {

                        initialize(_match_,"00:00:00:00:00:04","
                            00:00:00:00:00:03",167772164,167772163,3,0);
                        configure (two_match,"00:00:00:00:00:04",167772164,
                            finally_match,"00:00:00:00:00:02",167772162);

                        flowMod(sw, m,two_match, (short) 3);
                        flowModR(sw, _match_,finally_match, (short)2);
                        sendPacketOut(sw,pi);
                    }
        }
        return Command.CONTINUE;
    }
}
```