



Eder Leão Fernandes

Soft Switch 1.3: An experimenter-friendly OpenFlow Switch Implementation

Campinas

2015



UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Eder Leão Fernandes

Soft Switch 1.3: An experimenter-friendly OpenFlow Switch Implementation

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Engenharia de Computação.

Supervisor: Prof. Dr. Christian Rodolfo Esteve Rothenberg

Este exemplar corresponde à versão final da tese defendida pelo aluno Eder Leão Fernandes, e orientada pelo Prof. Dr. Christian Rodolfo Esteve Rothenberg

Campinas

2015

INCLUA AQUI O PDF COM A FICHA CATALOGRÁFICA FORNECIDA PELA BAE.

INCLUA AQUI A FOLHA DE ASSINATURAS.

Abstract

OpenFlow is the most prominent technology to enable Software Defined Networking. Designed as control interface between switches and controllers, the protocol might be considered an instruction set to program the network forwarding logic. The first OpenFlow version attracted attention from the industry and academy researchers interested in SDN promised benefits. Quickly, a toolset for OpenFlow 1.0 was available, which included switches, controllers, test and emulation software. When the protocol standardization process started by the Open Network Foundation, OpenFlow evolved fast and new specifications emerged in the last years. New features empowered the protocol and created enthusiasm, however projects of experimentation tools did not followed the OpenFlow fast pace. This work addresses one of these gaps, implementing an experimenter friendly OpenFlow 1.3 software switch. Driven by simplicity and basic performance requirements, the tool purpose is to be a functional and easy option for SDN developers that want to take advantage of the benefits brought by more recent OpenFlow versions. Overall, this project resulted in the open source release of the first OpenFlow 1.3 switch, allowing researchers from all around the globe to prototype and demonstrate solutions impossible in a prior moment.

Keywords: Computer Networks; Software Defined Networking; OpenFlow; Future Internet.

Resumo

OpenFlow é a mais proeminente tecnologia para a implementação de Redes Definidas por Software (RDS). Projetada como uma interface de controle entre switches e controladores, o protocolo pode ser visto como um conjunto de instruções para programar a lógica de encaminhamento em comutadores da rede. A primeira versão do *OpenFlow* atraiu a atenção de pesquisadores da indústria e universidades interessados nos potenciais benefícios prometidos por RDS. Rápidamente surgiram ferramentas para experimentação em *OpenFlow* 1.0, incluindo comutadores, controladores e software para testes e emulação. Após o início da padronização do protocolo pela *OpenNetworkFoundation*, o protocolo OpenFlow evoluiu rapidamente dando origem à novas especificações. As novas funcionalidades aumentaram as possibilidades de experimentos, gerando entusiasmo. Porém, o desenvolvimento das ferramentas de experimentação não acompanharam o mesmo ritmo do protocolo. Para preencher essa lacuna, esse projeto desenvolveu um comutador em *software* com suporte a *OpenFlow* 1.3. Guiado pelo objetivo de ser simples e básicos requerimentos de performance, a proposta da ferramenta é ser uma opção, fácil e funcional para desenvolvedores de aplicações RDS buscando utilizar as novas funcionalidades do *OpenFlow* 1.3. Em suma, o *software* desenvolvido nesse projeto foi o primeiro comutador *OpenFlow* 1.3 do mundo. Lançado como projeto de código aberto, possibilitou a pesquisadores de todo o mundo a prototipagem e demonstração de soluções não possíveis anteriormente.

Palavras-chaves: Redes de Computadores, Software Defined Networking, OpenFlow, Internet do Futuro.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objectives	2
1.3	Text Structure	4
2	Literature Review	5
2.1	OpenFlow	5
2.1.1	One Day in the Life of An OpenFlow 1.3 switch in 10 steps	7
2.2	OpenFlow software switches	9
2.2.1	OpenFlow reference switch	11
2.2.2	Open vSwitch	11
2.2.3	LINC	11
2.2.4	Trema	12
2.3	OpenFlow Controllers	12
2.3.1	NOX	12
2.3.2	POX	13
2.3.3	Floodlight	13
2.3.4	Ryu	13
2.4	OpenFlow test and emulation	14
2.4.1	OF-Test	14
2.4.2	Ryu Test Framework	14
2.4.3	OpenFlow packet dissectors	14
2.4.4	Mininet	15
3	Architecture	17
3.1	Software Switch Elements	18
3.1.1	Ports	18
3.1.2	Packet Parser	19
3.1.3	Flow Tables	20
3.1.4	Group Table	21
3.1.5	Meter Table	22
3.1.6	Marshaling/Unmarshaling library	22
3.1.7	Communication Channel	23
4	Development	25
4.1	Software switch implementation	25

4.1.1	Oflib	26
4.1.2	OpenFlow Extended Match	27
4.1.2.1	Packet Parser	28
4.1.2.2	Flow Match Prerequisites	30
4.1.2.3	Flow Matching	31
4.1.2.4	Extensible context expression in 'packet-in'	32
4.1.3	Set Field action	32
4.1.4	Per-flow Metering	33
4.1.5	Connection Features	34
4.1.5.1	Auxiliary Connections	35
4.1.5.2	Controller Role	36
4.1.5.3	Event Filtering	37
4.1.6	Other Changes	38
4.1.7	Dpctl	39
4.2	Open source development	39
4.2.1	Development workflow	40
4.2.2	Code maintenance	40
5	Evaluation	43
5.1	Feature Completeness	43
5.1.1	OFTest results	44
5.1.2	Ryu Certification results	45
5.2	Performance Benchmarks	45
5.2.1	Maximum Throughput	46
5.2.2	Throughput in function of flows and tables	48
5.2.3	Ping Round Trip Time	49
5.3	Portability	50
6	Conclusion	53
6.1	Results	53
6.2	Use Cases	55
6.3	Future Work	55
	Bibliography	57
	Annex	63
	ANNEX A NetPDL packet description example	65
	ANNEX B Publications	67

ANNEX C Full Ryu Certification test results	69
C.1 Action Tests	70
C.2 Match Tests	73
C.3 Group Tests	75
C.4 Meter Tests	75

The road for a dream is not a solitary path. During this arduous and long walk, some faces keep up by your side until the end, while others come and go in the middle of the way. Regardless the time spent with you, everyone leaves marks and contribute, for the good and for the bad, with your personal growth. For this reason, I would like to dedicate this work to everyone that at some point in my life, helped me to go through this process and reach my aspirations.

Acknowledgements

My thanks to my parents, for all the support, affection and belief. Also for raising me with enough freedom to chose my own path. I am grateful for my family too, for they pride on my achievements, always motivating me to go further.

Christian Esteve Rothenberg, my supervisor and friend.

My friends, Cássio, Daniel and Mônica, for the company, laughs and nonsense conversations that made things lighter.

Allan Vidal, for the long work partnership during all this years and due to the implementation of multiple connections in the software switch and all the help with Latex figures.

I am glad to have meet Kathrin. She showed me the light when everything was getting dark.

Marcos Rogério Salvador and Marcelo Ribeiro Nascimento, people who guided me through the SDN world and made me believe in the possibility to achieve international recognition.

Many thanks for CPqD, the Brazilian research center where the project development started; Ericsson Innovation Center Brazil, for sponsoring this work, and Ericsson Traffic Lab in Hungary, for technical support.

Among the collaborators I would like to thank and highlight two: Zoltán Lajos Kis, for the OpenFlow 1.1 software switch implementation and technical guidance; Jean Tourrilhes, for critical bug fixes and help with the git workflow.

Finally, my sincerely acknowledgments to people who collaborated with code, bug reports or suggestions.

“The path to OpenFlow is not a four lane highway of joy and freedom with a six pack and a girl in the seat next to you, it’s a bit more complex and a little hard to say how it will work out, but I’d be backing OpenFlow in my view”

Greg Ferro

List of Figures

Figure 1 – Traditional and SDN models	2
Figure 2 – OpenFlow switch minimal elements.	5
Figure 3 – Simple topology for the learning switch example.	8
Figure 4 – OpenFlow Software Switches: version support timeline.	10
Figure 5 – Software switch architecture	18
Figure 6 – Group Table internals	21
Figure 7 – Meter Table internals	22
Figure 8 – OXM field example	28
Figure 9 – Packet Parser components	29
Figure 10 – Token Bucket Algorithm illustration inside the meter band	35
Figure 11 – User space software switches throughput comparison	47
Figure 12 – Influence of the number of installed flows on the throughput.	48
Figure 13 – GitHub statistics.	54

List of Tables

Table 1 – Minimum bandwidth requirements for common internet applications	3
Table 2 – OpenFlow required match fields	6
Table 3 – Switch Flow Table state after controller connection	8
Table 4 – Switch Flow Table state after learning Host 1 address	9
Table 5 – Switch Flow Table final state	9
Table 6 – Comparison of OpenFlow software switches	10
Table 7 – Basic OpenFlow messages	44
Table 8 – Role request message results	44
Table 9 – Ryu Certification results comparison	45
Table 10 – Ping Round Trip Time comparison between software switches	50

Acronyms and Abbreviations List

API	Application Programming Interface
ARP	Address Resolution Protocol
ASIC	Application Specific Integrated Circuit
BOS	Bottom of the stack
CAM	Content Addressable Memory
CRC	Cyclic Redundancy Check
EH	Extension Header
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPv6	Internet Protocol Version 6
Kbps	Kilobits per second
Kpps	Kilo packets per second
MAC	Media Access Control
Mbps	Megabits per second
MPLS	Multiprotocol Label Switching
NetPDL	Network Protocol Description Language
ONF	Open Network Foundation
OVS	Open vSwitch
OXM	OpenFlow Extended Match
PBB	Provider Backbone Bridge Protocol
QoS	Quality of Service
RAM	Random Access Memory

SCTP Streaming Control Transfer Protocol

SDN Software Defined Networking

TCP Transmission Control Protocol

TDD Test Driven Development

TLS Transport Layer Security

TLV Type-Length-Value

XML Extensible Markup Language

1 Introduction

Traditional computer networks have been successful in their most basic goal: make packets originated from a source location to reach a destination (SHENKER, 1995). However, the exponential growth in Internet users, emerging use cases and applications have become a challenge for network carriers and administrators. These professionals should be able to handle and to master more and more complex scenarios and configurations (FELDMANN, 2007). Furthermore, network equipments are strictly closed or only offer a small set of options for users who want to add their own functionalities and applications. Consequently, innovation in computer networks is compromised, compelling companies to wait for new features on software updates or, worse, to buy a new network box.

In order to address these issues, inspired by older technologies that have followed similar concepts and evolved (FEAMSTER *et al.*, 2014), a new network model was needed. The Software Defined Networking (SDN) (KREUTZ *et al.*, 2014), is a network architecture in which the control plane of network switches is decoupled from the forwarding plane, as illustrated by Figure 1. The control plane is responsible for the management of one or more elements from the forwarding plane. The applications running on top of the control plane can program the data plane to execute determined actions according to the packet type received by an equipment or some network event. As a result of this flexibility to control the forwarding plane, network equipments may receive new functions and do not need to be replaced when the need for a new functionality arises. Moreover, the network resources can be fully exploited by some smart resource allocation, like network virtualization (SHERWOOD *et al.*, 2009) (AL-SHABIBI *et al.*, 2014), leveraging the network to its full potential.

The first and the most common standard southbound interface for SDN is the OpenFlow protocol (MCKEOWN *et al.*, 2008) (ONF, 2012b). The OpenFlow specification describes the interaction between an OpenFlow compliant switch and an OpenFlow controller. Basically the controller installs flow entries for the switch flow table, so that the switches apply instructions based on the network traffic.

1.1 Motivation

Among the reasons for the fast evolution of the OpenFlow protocol is the experimental work led by Stanford researchers, where the protocol was born. New features and capabilities were validated on a software switch implementation, allowing researchers to create and to

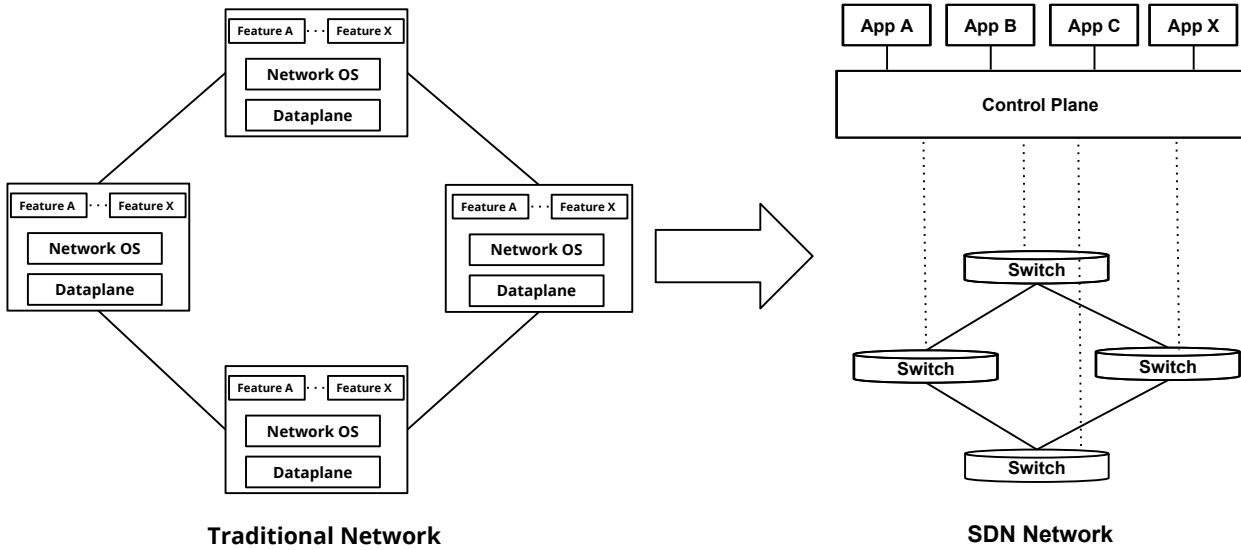


Figure 1 – Traditional and SDN models

try new control plane applications. Soon, advances and emerging use cases took the industry attention for SDN and OpenFlow, culminating in the creation of the Open Network Foundation (ONF), an organization composed by big network players and vendors, as by emerging startups. This organization became responsible for new OpenFlow versions since the version 1.2 and started working on new and enhanced features, resulting in the version 1.3 less than one year after 1.2.

On the other hand, OpenFlow controllers and switches did not follow the protocol advancements, notwithstanding the fast evolution, resulting in lack of alternatives to experiment new capabilities and anticipation of new applications that could benefit from new features.

With this scenario in mind we found the need to upgrade these tools, allowing fast experimentation and validation. By keeping up the pace with OpenFlow new versions, we expect to contribute to the future of the protocol, driving companies and researchers to develop applications in the state of the art and enabling future OpenFlow version to be build and tested upon our work.

1.2 Objectives

The objective of this work is the development of a programmable OpenFlow 1.3 software switch to enable fast, real and flexible experimentation for SDN research and education on OpenFlow networks. To achieve this objective, the software switch must meet the following requirements:

1. **OpenFlow protocol feature completeness.** All required and optional features shall be implemented, allowing a full OpenFlow experience, without limitations for SDN researchers and developers.
2. **Code simple, easy to prototype and extend.** The code must be simple enough to be modified by anyone with a basic level of programming and understanding of OpenFlow. For this reason, easy insertion of features should be favored in lieu of performance. This requisite meets research needs that goes beyond the OpenFlow specification (e.g: the addition of new messages, new algorithms for group processing, changes to the pipeline, etc). Also, it encourages and helps users to search and to fix bugs quickly, preventing work interruption while waiting for an official patch to correct the switch.
3. **Straight forward integration with experimentation environments and emulation tools.** The switch must integrate with both real and emulated environments, ensuring seamless communication with other switches and controllers, without huge modifications. Minor changes are acceptable due to specific platforms requirements: for instance, different processor architectures.
4. **Maximum throughput equal or higher than 100 Mbps.** High performance is not one of the project goals. However, there are features that play with the switch packet rate. Therefore, for a significant user experience, the switch must be able to support rates of at least 100 Mbps. This value was estimated based by the minimum bandwidth required by common Internet applications. Table 1 is a mix from values obtained by (CHEN *et al.*, 2004) and some popular Internet services. Considering the applications and the bandwidth usage, we found that 100 Mbps is a value to perform a reasonable number of different experiments.

Table 1 – Minimum bandwidth requirements for common internet applications

Application	Bandwidth (Mbps)
Web Browsing	0.038
Email	0.01
Telnet	< 0.001
Audio Broadcasting	0.08 to 0.375 ¹
Video Broadcasting	0.5 to 60 ²

¹ Spotify - <https://support.spotify.com/us/learn-more/faq/#!/article/What-bitrate-does-Spotify-use-for-streaming>

² Netflix - <https://help.netflix.com/en/node/306>

1.3 Text Structure

In this Introduction we explained the motivational aspects that justify this work. Also, we give a clear explanation for the objectives of this project.

In Chapter 2 we present a Literature Review. Related OpenFlow software switches current functionalities are discussed from the point of view of our implementation requisites. Furthermore, we introduce other tools which are important parts of the OpenFlow ecosystem.

In Chapter 3 we take a look at the architecture of the software switch which is compliant with OpenFlow 1.3. We explain the modules relationship and roles within the OpenFlow pipeline.

In Chapter 4 we highlight implementation details of OpenFlow 1.3 features in our architecture.

In Chapter 5 we evaluate the software switch in terms of common OpenFlow benchmarks and compare with related work.

Finally, in Chapter 6 we give our conclusion remarks. This chapter highlights results, presents known use cases and discusses possible improvements in future works.

2 Literature Review

OpenFlow is the central theme of this project and deep understanding of the protocol is required. Therefore, the OpenFlow 1.3 specification (ONF, 2012a) is the main document of our bibliographic base. We also have studied and compared available implementations of other OpenFlow software switches. Moreover, some tools to evaluate our work are worth mentioning because of the necessary adaptations made to overcome the lack of support for the most recent OpenFlow versions. We investigated controllers, test frameworks and packet dissectors, all candidates to compose our OpenFlow test environment. The next sections will give an overview about OpenFlow and relevant tools related to this work.

2.1 OpenFlow

OpenFlow is an open standard communication interface between switches and controllers, allowing centralized control and programmability in the network. The basic OpenFlow switch is composed by one or more flow tables, a group table and one or more OpenFlow channels for communication with OpenFlow controllers. Figure 2 is a logical view of the minimal elements required by a switch.

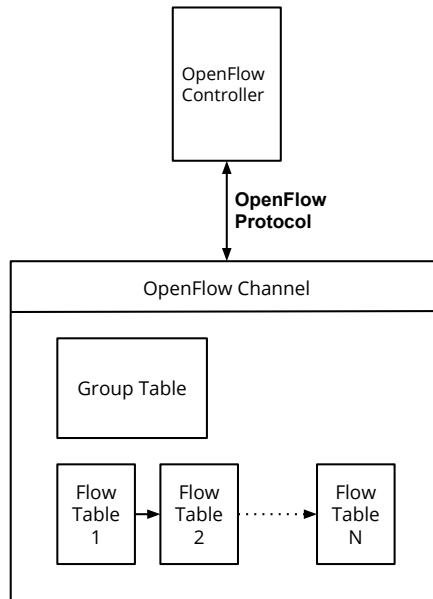


Figure 2 – OpenFlow switch minimal elements.

OpenFlow controllers can install flows into the switch flow table. A flow consists of match fields, counters and instructions applied to matching packets. A packet matches a flow if the protocol header fields values, are the same as those specified in flow match fields. The most recent version requires 13 match fields, shown by Table 2, but has optional support for more than 30 protocols fields from layers 2, 3 and 4 of TCP/IP network stack.

The OpenFlow pipeline starts at the first flow table and continues to additional tables. Flows are matched in order of priority and the associated instructions are executed. Only two instructions are required for OpenFlow switches: Write-Actions, in which actions are executed at the end of the pipeline, and Goto-Table, to jump to tables with an id greater than the table where the instruction is executed. Optional instructions are Meter, to direct the packet to some meter for QoS, Apply-Actions, for immediate action application, Clear-Actions, to clear all actions written by a Write-Actions instruction, and Write-Metadata, to write metadata information to be carried across the tables.

Table 2 – OpenFlow required match fields

Field	Description
Inport	Ingress Port
Eth Dst	Ethernet destination address
Eth Src	Ethernet source address
Eth Type	Type of the packet, after VLAN tags
IP Proto	IPv4 or IPv6 next protocol number
IPv4 Src	IPv4 source address
IPv4 Dst	IPv4 destination address
IPv6 Src	IPv6 source address
IPv6 Dst	IPv6 destination address
TCP Src	TCP source port
TCP Dst	TCP destination port
UDP Src	UDP source port
UDP Dst	UDP destination port

Actions can perform modifications on packets, discard or send them to the group table or simply output to some specific port. The only required actions are Output, to send the packet through a port, and Drop. The packet modification actions are all optional, but their implementation is recommended as they give more power and options to OpenFlow networks. The optional actions are Group, to process the packet through a specific group; Push-Tag/Pop Tag, for addition and deletion of VLAN, MPLS and PBB tags; Set-Field, to modify packets header fields; Change TTL, an action to modify MPLS and IP TTL; Set-Queue, to determine which queue is attached to a port and will be used for scheduling in packet forwarding.

OpenFlow groups are a way to perform more complex forwarding actions. When a

packet is sent to a group it is cloned and executed by sets of action buckets. This abstraction enables flooding, multipath, link aggregation and other techniques that demand transmission of packets through more than one port.

The last essential block is the OpenFlow channel. The principal connection between controller and switch is done by one of the following transport protocols: TCP or TLS, where the second is recommended because enables data encryption of the transmitted data. Auxiliary connections are also allowed and it is possible to have UDP connections for transmission of less sensitive OpenFlow messages.

An optional element of the OpenFlow switch is the Meter Table. This table comprises different types of Meter Bands, which have a speed limit and apply a determined QoS action in the case of a packet flow exceeding the determined limit. There are two types of bands covered by the OpenFlow 1.3 specification: Drop, to discard packets; and DSCP Remark, to decrease the drop precedence of the DSCP field of the IP header.

2.1.1 One Day in the Life of An OpenFlow 1.3 switch in 10 steps

To illustrate the operation of an OpenFlow switch we will give a common and simple example. A learning switch is a layer 2 network equipment that learns the port to which a host is connected. Learning happens when a packet from a host arrives the switch for the first time. The switch then obtains and stores the host Media Access Control (MAC) address associated with the port number. Next time, when another host sends a packet to the previous learned address, the switch will forward it directly, instead of flooding to all ports.

In legacy devices, the control software is embedded into the switch hardware and the MAC address are stored in a Content Addressable Memory (CAM) table. In an OpenFlow scenario the learning happens inside the controller and the forwarding rules are stored in the Flow Table. We will describe the steps of learning and forwarding process of an OpenFlow switch controlled by a simple learning switch application, considering the topology in Figure 3.

In the initial state the switch Flow table is empty, *Host 1* and *Host 2* do not know anything about each other and the *Controller* is about to connect with the *Switch*.

1. The *Controller* establishes a connection with the *Switch*. As packets sent to an empty Flow Table are dropped according the OpenFlow 1.3 specification, the *Controller* installs a low priority flow to direct every non matching packet to him. Table 3 shows the Flow table state after install the first flow.

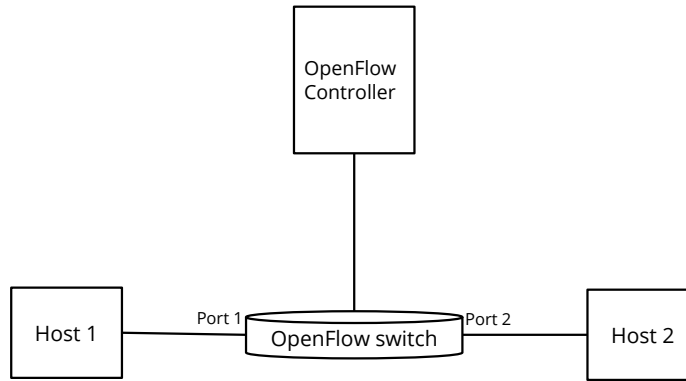


Figure 3 – Simple topology for the learning switch example.

Table 3 – Switch Flow Table state after controller connection

Match	Priority	Instruction
all	0	apply actions -> output:controller

2. *Host 1* wants to transmit a file to *Host 2*. As it does not know about *Host 2* MAC address, it sends an ARP (PLUMMER, 1982) request to the network.
3. The ARP request packet enters the *Switch* and matches the unique flow installed in the Flow Table. The action is applied, sending the packet to the *Controller* in an OpenFlow *Packet In* message.
4. The *Controller* receives the *Packet In*. The message contains information about the packet input port and headers. The *Controller* application learns and stores the *Host 1* input port and MAC address. After that, it sends a *Packet Out* message back to the *Switch*, with an action to flood the packet in every *Switch* port.
5. The *Packet Out* message arrives the *Switch* and the packet is flooded to every port, except the port where it came from.
6. The ARP request is delivered to *Host 2*. Then, an ARP reply is sent back with the *Host 2* MAC address required by *Host 1*.
7. The ARP reply arrives the *Switch*. It matches the only present flow and is sent in a *Packet In* message to the *Controller*.
8. Now, the *Controller* checks if the packet MAC address is known. As the address is not present, it stores the input port and the MAC. Now it checks if the destination address was stored previously. The destination to *Host 1* is already know. The *Controller* installs a new flow into the *Switch* Flow Table. The flow illustrated in the second row of Table 4, matches every packet destined to *Host 1* MAC address and output the packet into

port 1. After installing the flow, it sends the ARP reply to the *Switch*, encapsulated in a *Packet Out* message.

Table 4 – Switch Flow Table state after learning Host 1 address

Match	Priority	Instruction
all	0	apply actions -> output:controller
eth dst:HOST 1 MAC	100	apply actions -> output:1

9. This time The ARP reply, is not flooded to all ports. As the *Controller* knew about the destination, it is sent directly to *Host 1*. After receiving the ARP reply, it starts transmitting the file to *Host 2*.
10. Again, the first packet of the file transference to *Host 2* is sent to the *Controller*. Now it knows about *Host 2* MAC and origin port. It installs another flow, shown in the third row of Table 5, and send the *Packet Out* message to the *Switch*. The *Switch* sends the packet directly to *Host 2*. From now on, the packets are not sent for the *Controller* anymore because the installed flows are matched.

Table 5 – Switch Flow Table final state

Match	Priority	Instruction
all	0	apply actions -> output:controller
eth dst:HOST 1 MAC	100	apply actions -> output:1
eth dst:HOST 2 MAC	100	apply actions -> output:2

2.2 OpenFlow software switches

OpenFlow software switches play an important role in the protocol evolution. At first, they were a low cost solution to create and experiment your own SDN applications in a controlled and smaller environment, before the deployment in a production environment. With the new SDN approaches for network virtualization (TSENG *et al.*, 2011) (DRUTSKOY *et al.*, 2012), software switches are receiving a lot of attention from the industry (VMWARE, 2012) and academy (EMMERICH *et al.*, 2014). SDN virtual switches interconnect virtual machines from data center tenants and help to scale a plethora of traffic engineering applications. For instance, tenant isolation is easily achieved using an OpenFlow software switch to connect virtual machines. Usually it is implemented using VLAN segregation, which is not scalable for more than 4096 tenants (the total number of possible VLAN identifiers). In turn, OpenFlow switches offers more granular options to segregate the network hosts, due to the number of

fields available for flow matching, eliminating the need to segregate hosts using only a layer 2 domain.

Table 6 – Comparison of OpenFlow software switches

Switch	Language	Emulation tool integration	Mode	OF-Config
Reference	C	Yes	userspace	No
Open vSwitch	C	Yes	userspace/kernel	No
LINC	Erlang	No	userspace	Yes
Trema	C/Ruby	No	userspace	No

Before the implementation, we investigated four OpenFlow software switches that supported OpenFlow 1.0, since there was no sense implementing basic OpenFlow features from scratch. Table 6 is a comparison of the examined software switches and reflects their state in the year of 2012. The columns are related to the objectives defined in chapter 1. Popular programming languages have the power to reach a bigger audience. The integration with emulation tools eases experimentation and speed up testing. The mode column is related to switch performance, since kernel space implementations tend to be more efficient than a user space implementation. In a kernel implementation packets are processed directly in the kernel space, eliminating the overhead of traversing packets between the kernel and user space.

In the mean time of this work, most of the analysed software switch projects started to add support for newer OpenFlow versions. Figure 4 shows the switches timeline for the implementation of most recent OpenFlow versions. Colored intervals mean that these OpenFlow versions are still in progress or do not include all features described on the specification as we will show in section 5.1.

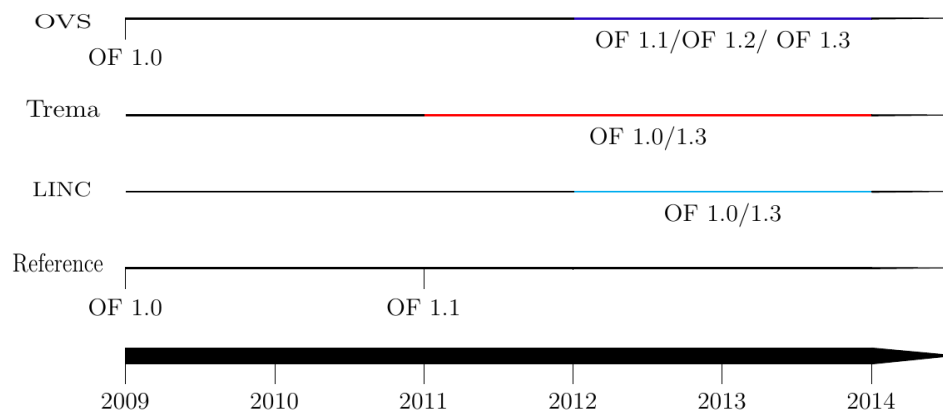


Figure 4 – OpenFlow Software Switches: version support timeline.

The next subsections will provide a short individual description of the software switches investigated: OpenFlow reference switch, Open vSwitch, LINC and Trema.

2.2.1 OpenFlow reference switch

The first OpenFlow switch is known as reference switch because it was implemented by Stanford researchers directly involved with the OpenFlow protocol creation and the first specification releases. The code is written using the C language and its simplicity is one of the reasons for the implementation to other platforms. Two important ports are: NetFPGA boards (STANFORD, 2010a), eliminating the disadvantage of the user space implementation; and OpenWrt (STANFORD, 2010b) for wireless routers. These efforts enabled low cost options to test OpenFlow on real hardware during earlier protocol stages.

The last OpenFlow version implemented by Stanford researchers was 1.0, but after the version 1.1 release there was an update, based on the reference switch, released by Ericsson Traffic Lab, called of11softswitch (KIS, 2011). To conform changes like multiple tables, the switch forwarding plane was rewritten, but still retained all of the base software switch characteristics.

2.2.2 Open vSwitch

Considered the *de facto* switch for virtual networks, Open vSwitch (PFAFF *et al.*, 2009) (OVS) is a mature and constantly evolving open source project. The efficiency provided by the kernel module and the functionalities beyond OpenFlow, turns OVS into a great solution to replace the original Linux bridge. In addition to the basic connectivity provided by traditional bridges, OVS offers a flexible option to manage and program the packet forwarding behavior. OVS uses another protocol for switch management. The Open vSwitch Database Management Protocol (OVSDB) is one of the most praised OVS features and is designed to manage all running switch instances, permitting the control of distributed virtual network nodes. With OVSDB a network engineer can create, configure and delete OVS ports and tunnels from a centralized location.

2.2.3 LINC

LINC (FORWARDING, 2012) is an userspace software switch written in the Erlang programming language and has different support levels, considering the number of working features, for OpenFlow versions from 1.2 to 1.4. The main advantage of this switch is the

support to OF-CONFIG (FOUNDATION, 2011), the OpenFlow switch configuration protocol. As a userspace implementation, efficiency is not one of its strong points. However, it promises flexibility, fast development and testing of new OpenFlow features. The Erlang language is not a disadvantage per se, but can be considered a blocking point for developers who want to add their own features. Erlang developers are growing, but their number are still very far from languages like C and Java.

2.2.4 Trema

Trema is a name most known by the OpenFlow controller, but it goes beyond. It offers a full OpenFlow framework with the tools needed to develop OpenFlow applications (NEC, 2013), including an OpenFlow software switch. Also, the framework has its own emulation tool for OpenFlow networks and end-hosts. The main repository of this project features switch with only OpenFlow 1.0. However, there is a repository known as *trema-edge* where the work for OpenFlow 1.3 is on progress.

2.3 OpenFlow Controllers

Controllers are considered the brain of an OpenFlow network. Every configuration and forwarding rules are defined by applications running on top of a controller. They are sent in form of OpenFlow messages, which needed to conform with the format defined by the specification. For this reason, we need to validate our work testing interoperability between the software switch and a compliant controller. We reviewed the main open source projects, looking for an OpenFlow 1.3 controller or an easy alternative to implement some level of support for OpenFlow 1.3, required for our tests.

2.3.1 NOX

One of the first open source OpenFlow controllers, NOX (GUDE *et al.*, 2008) was very famous during the first years of OpenFlow. The popularity was due a combination of factors, from the C++ implementation and a Python binding, which helped to speed the prototyping, to a quite simple interface and good number of example applications. The last official release supported OpenFlow 1.0 version. After some enhancements on the controller speed (TOOTOONCHIAN *et al.*, 2012), no efforts were made to upgrade it for newer OpenFlow versions.

2.3.2 POX

Pox is a controller implemented in Python and can be considered a NOX sibling (POX, 2014), born to address the lack of speed when prototyping with NOX. Its main goal is to become the archetypal of a modern SDN controller, featuring some desired SDN capabilities like debugging, new programming models and network virtualization.

Designed with research in mind, under constant development and a typical controller used for SDN education, POX was a great candidate to be the controller to test the software switch, but the OpenFlow support still not surpasses the version 1.0

2.3.3 Floodlight

The Floodlight controller is a popular OpenFlow controller backed by Big Switch Networks, one of the most prominent SDN startups (FLOODLIGHT, 2011a). It is developed in Java and was designed for high performance and is the core of Big Switch Networks commercial solution. One of the greatest features of Floodlight is the module loading system that makes it very extensible, allowing to enable and disable application at run time. Another great feature is the Open Stack integration, a cloud orchestration platform. Floodlight instances control the virtual switches linking virtual machines orchestrated by Open Stack. Regarding OpenFlow version support, it currently supports OpenFlow 1.0 and 1.3.

2.3.4 Ryu

Ryu is considered an SDN network framework (NTT, 2013b), an abstraction that provides code software components, with generic functionality, easing and speeding development of SDN applications. It is implemented in Python, like POX, but has a different architecture. Designed as software components, developers can create OpenFlow applications like modules. Furthermore, the controller also supports management protocols like OF-Config and Netconf.

Very well documented, with a large number of examples and featuring all OpenFlow versions, Ryu is currently a great choice to test the software switch, but when we started this work OpenFlow support was limited to OpenFlow 1.0. OpenFlow 1.3 support was only available in the middle of our switch implementation.

2.4 OpenFlow test and emulation

The last pieces of a minimal OpenFlow environment are test frameworks and emulation tools. For our development we are more interested in the first, on the other hand, a modular emulation software compatible with our software switch may benefit users looking for a complete, controlled and easy to setup testbed.

2.4.1 OF-Test

OF-Test was the first OpenFlow test framework. Developed by the same team working on Floodlight, OF-Test (FLOODLIGHT, 2011b) tests basic functionality for OpenFlow 1.0 and 1.1, with 1.2 and 1.3 currently in development. The simpler architecture and Python implementation turns it into a very easy and fast platform to create and run tests. It works connecting the OF-test server to the switch control and the data plane. The server is responsible by monitoring OpenFlow messages and packets sent through and along the planes. If these messages and packets are according to expected results, the test returns OK, otherwise, a failure is reported.

2.4.2 Ryu Test Framework

This test framework is part of the Ryu controller and implements tests to cover all, required and optional, OpenFlow 1.3 and 1.4, actions, instructions and match fields, with more comprehensive test than OF-Test. The test cases are written in JSON, so there is no need to code to create a test, enhancing the speed of test creation.

There is an online certification which continuously tests OpenFlow software and hardware switches, including our work (NTT, 2013a). Results from this certification will be presented in chapter 5.

2.4.3 OpenFlow packet dissectors

Packet dissectors are important tools to test if message packets are correct or to check if an specific packet was sent or modified by an OpenFlow output and set-field actions, respectively. Wireshark (WIRESHARK, 2014) is the most famous program to dissect packets. Although a wide range of protocols are officially supported, OpenFlow support started as an unofficial plug in for OpenFlow 1.0 and 1.1. For a long time this plugin was the only option to

analyze OpenFlow traffic. Recently due to the growth in the number of users requesting for official support, OpenFlow is now developed in the dissector main repository (WIRESHARK, 2014).

2.4.4 Mininet

Mininet (LANTZ *et al.*, 2010) is a tool for network emulation. In a single machine it runs switches, links and hosts just like a complete virtual network. It is possible to log into the hosts and use programs like Iperf (NLNR/DAST..., Accessed 2007) and Ping, to measure throughput and check connectivity; specify link parameters as speed and delay; and instantiate a network topology composed of software switches. The capacity to create and to destroy virtual networks allows easy and fast experimentation.

3 Architecture

Architectural design is strongly tied down by OpenFlow 1.3 required and optional elements. With the architecture description, we do not intend to repeat the specification. The elements described in the next sections are a conceptual point of view of each software component.

The OpenFlow switch implementation is a fork from the OpenFlow 1.1 reference switch. We choose this implementation as a starting point for our work because of the simpler code base, when compared to other switch implementations discussed in section 2. Although simple, the only documentation available to understand the reference switch was the OpenFlow specification. For this reason, the definition of the architectural design started from the extraction of the previous software architecture.

Firstly, through a simple reverse software engineering process (SOMMERVILLE, 2001), we analyzed the code and listed the switch core components. Next, we identified missing components from the OpenFlow 1.3 specification. After these steps, we found that block structures suggests the application of a bottom-up design (MAYRHAUSER, 1990). In this approach, the basic set of foundational modules and their interrelationships are the foundation for the final architecture. Following these concepts we came up with the final design for the OpenFlow 1.3 software switch.

In the Figure 5 we show the software switch architecture. The most important block is the Datapath. It consists of OpenFlow internal elements such as Flow, Meter and Group Tables, and a Packet Parser as well. The other three blocks operate on different levels along with the Datapath. From the top level, the blocks Datapath, Marshaling/Unmarshaling library and Communication Channel are part of the OpenFlow message layer. Below, Ports and Datapath form the network packets layer, where packets arrive, are processed and usually sent back for the network.¹ Except for the special case of the *Packet In* message, in which packets can be sent for the controller, these two layers do not interact. In Figure 5, dotted lines illustrate some possible paths a network packet can travel between the Ports and the Datapath. Solid lines denote the OpenFlow messages traveling in the OpenFlow message layer. Arrows mean the direction packets and OpenFlow messages can take across the switch components.

¹ Some instructions, actions and even an empty table may cause a packet drop in the Datapath.

There is a set of responsibilities associated with the OpenFlow protocol and the pipeline. Functionalities are:

- OpenFlow enables some level of control over a port behavior. A port modification message permits the configuration of the port state. Ports can be administratively set to drop all received or forwarded packets, forbid the generation of *Packet In* messages from arriving packets, and brought down. Ports elements should handle these messages and change the port behavior according to the configuration sent.
- OpenFlow Ports keep the current state of the physical link. This information is not configurable by an OpenFlow controller, but the switch should inform the control plane about link state changes. Ports monitor the state of the port link and update the information according to changes.
- Packets encapsulated in *Packet In* usually have only the header sent within the message. A buffer stores the packet, while waiting for the controller decision after the *Packet In*. Ports elements store these packets and resend them for further processing.
- An OpenFlow controller can ask the switch about a port description. The software switch element retrieves information like current and max operating speed from the interfaces of the machine and stores it. On a port description request, the element handles the message and sends the required information to the control plane.
- Queues creation are not part of the OpenFlow protocol. However, OpenFlow can configure port queues, created by whichever mechanism, to be associated with a switch port. Ports are responsible for handling queue association and configuration.
- Ports must update port and queue packet counters.

3.1.2 Packet Parser

Before entering the software switch OpenFlow pipeline, packet protocol fields are extracted by the Packet Parser element. Parsing packets was a formally defined task until OpenFlow 1.1. The main reason to define how packets should be parsed is to guarantee parsing consistency, but it limits switch designers and demands algorithm updates for each new protocol addition. For this reason, further specifications removed how packets should be parsed and match fields are now defined only logically.

A Packet Parser element converts extracted protocol fields of a packet to an internal flow entry format. Two scenarios may trigger this function:

- A network packet enters the switch through one of its ports.
- If the packet was modified by an action and is resubmitted for the pipeline, or sent to a table ahead by a *Go To Table* instruction, packet revalidation is required. Hence the packet is processed by the Packet Parser again. In Figure 5, after passing by the Flow Tables, there is an arrow representing packet return to the Packet Parser.

Further OpenFlow extensions, supporting new protocols, directly affect the packet parsing. Modifications are required in order to add new match fields for the Packet Parser. Therefore, a flexible and extensible Packet Parser element is desirable.

3.1.3 Flow Tables

Flow Tables are the heart of an OpenFlow switch architecture. They are the elements where flow entries are stored and the OpenFlow pipeline starts. Although the use of multiple flow tables is optional - the specification mandates at least one table - its implementation is recommended, as even simple applications can not scale in switches with only one Flow Table (PEPELNJAK, 2013).

Flow Tables roles in the software switch are listed below.

- In case of nonexistence of a table miss flow entry, Flow Tables have to implement some default action for not matched packets. Currently, the default action is drop the packets.
- Handle *Flow Mod* messages sent by the controller. These messages may add or delete flow entries, or change the instruction set from currently installed flows.
- Flow Tables must be able to have their capabilities reconfigured by a controller. These table features can express the table supported properties. The instructions type and the match fields allowed in the table are examples of properties. Also, some fields show relevant information for an OpenFlow application. For instance, the table identifier value is an information required to add a new flow, and the max number of flow entries should be considered to avoid scalability problems.
- A Packet look up must be performed upon receiving of a packet. The operation looks for a flow table entry that matches the packet. In the case of a match, the switch executes the instruction set associated with the flow entry. This is the most common activity in Flow Tables.

- Keep table statistics about the number of active flow entries, number of look ups and matched packets.

3.1.4 Group Table

Group Table empowers OpenFlow forwarding options. Packets reach the Group Table after matching a flow entry containing a group action, in one of the Flow Tables.

Group entries are stored into the Group Table. Each group entry contains an identifier, a type, counters and action buckets. Action buckets are an ordered list of action sets to be executed according to the group type. Figure 6 represents a group table filled with groups of All, Indirect and Fast Failover types. The layout of the specific group types is important, because it defines Group Table attributions, as shown by the responsibilities listed here.

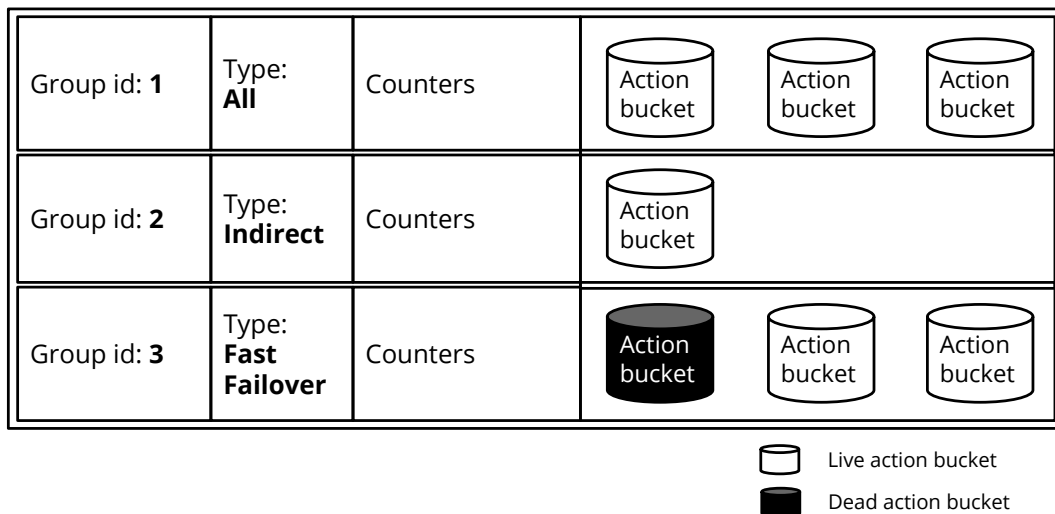


Figure 6 – Group Table internals

- The Group Table have to guarantee group type restrictions. For instance, indirect groups support only one action bucket.
- The Group Table must handle modification messages and perform consistency checks in the case of group chaining. Chained groups point to another groups and may cause loops that should be avoided by the element.
- Fast failover groups require monitoring switch ports and group buckets for state changes. For this reason the Group Table is responsible for checking bucket liveness when choosing the first live bucket.

- A Group Table that supports the select group type, has to implement a schedule discipline algorithm to choose which bucket will be applied to the packet.

3.1.5 Meter Table

The Meter Table is an element to perform simple QoS operations. Per-flow meters are attached to flow entries through the *Meter* instruction. A meter entry is composed by a meter id, counters and meter bands. The QoS operations to apply are defined by the meter bands. A meter band must have a type and rate value, which is the boundary to apply the action determined by the type. Figure 7 illustrates the internals of a Meter Table, with two meter band types.

Meter id: 1	Counters	Meter Band			
		Type: DSCP Remark	Rate: 100 kbps	Precedence Level: 1	Counters
Meter id: 2	Counters	Meter Band			
		Type: Drop	Rate: 100 kbps	Counters	

Figure 7 – Meter Table internals

Meter Table responsibilities include:

- Creation, destruction and modification of meter entries.
- Matched packets, from flows pointing to the Meter Table, rate measurement.
- Keep and update counters for statistics of packets processed by an entry.
- Process the packets according to band operation. A Drop meter band type discards the packets and the DSCP remark changes the IP packet drop precedence.

3.1.6 Marshaling/Unmarshaling library

OpenFlow messages defined by the specification follow a proper format for transmission in the network. Messages are 8-byte aligned, so there may be insertion of padding fields to follow this alignment rule. Another requirement for the message format is the byte order. The preferred format for packets sent through the network is the network byte order (REYNOLDS; POSTEL, 1994). As OpenFlow messages are sent over IP networks, their messages should be assembled following the Big-Endian format.

The architectures of machine's processor may operate on different byte-endianness. For instance, Intel processors use the Little-Endian byte order (VERTS, 1996). So, in order to handle and assemble OpenFlow messages, conversion is required for non Big-Endian architectures.

For the mentioned reasons, a library which abstracts byte-endianness and adds any required bytes to ensure right message format is required. A Marshaling/Unmarshaling library is not an element defined by OpenFlow specification. Its main function is the translation of OpenFlow messages from the network format to an internal format and vice-versa. The library responsibilities are the following:

- Every OpenFlow message must have a function that packs and unpacks it. Pack is the function which converts internal structures into network format. While unpack, turns received messages into an internal structure.
 - When packing, the library has to add any necessary padding bytes.
 - On packing, the message should be assembled in network byte-order.
 - On unpacking, the library must translate the message fields to the switch host architecture byte-order.
- Some handling of OpenFlow message errors is done in this level. The library must raise errors for messages with wrong length or bad arguments.

3.1.7 Communication Channel

The OpenFlow software switch communicates with controllers through the Communication Channel. This element connects with the Datapath and the controller and acts as proxy between them. This element exists because the implementation of the communication channel is not defined by the specification. Since the message format is respected, implementation are free to choose the connection protocol. For instance, when security for the channel is a requirement, a protocol like TLS should be used to encrypt the messages.

In the software switch, the Communication Channel roles are:

- The Communication Channel must establish a TCP connection with the switch and the controller.
- Connection setup is a Communication Channel responsibility. After a TCP connection, the switch negotiates the protocol version with the controller. This process, known as handshake, is managed by the Communication Channel.

- The channel may use multiple connections with a single controller at the same time. These connections can be used to send OpenFlow in parallel or to create specific channels for some message types.
- A Communication Channel is responsible for opening connections to enable switch communication with more than one OpenFlow controller.

4 Development

This chapter covers the software switch development. In section 4.1, we give details about the implementation of the most important OpenFlow 1.3 features, in section 4.2, we describe the open source model adopted for the software switch development.

4.1 Software switch implementation

Implementation of an OpenFlow switch depends on the platform for which it is designed. OpenFlow hardware implementation on traditional Application Specific Integrated Circuit (ASIC) chips, usually suffer from limitations, like small capacity in the total number of flows and not real support for multiple tables. Unlike hardware, software implementations offer greater flexibility in the implementation of OpenFlow features. In environments where high throughput is not the biggest concern, software switches running on commodity servers can be a low cost replacement option for traditional network switches.

The OpenFlow specification describes OpenFlow switches pipeline and the required and optional building blocks. It does not gives low level details about how these components should be implemented. As long as it works how the specification dictates, switch designers are free to use any data structures and algorithms in order to implement OpenFlow. When defining implementation details, we explored the software implementation freedom to meet the requisites defined on section 1.2. At the same time, we came up with innovative design decisions towards future extensions of the OpenFlow match field support.

In this section we discuss how we implemented the OpenFlow 1.3 software switch adding several changes to the base switch - using C ¹, the switch native programming language, and C++ - in order to support all features and keep it as simple as possible. The next subsections describe this new functionalities in the context of the architecture of the software switch presented in chapter 3.

¹ In this chapter there will be two common words: struct and structure. While struct is a C language keyword and structure is a more generic word for a collection of data variables, both will be used to denote a C struct

4.1.1 Oflib

The software switch architecture Marshaling/Unmarshaling library, presented in section 3.1.6, is called Oflib. Although already present in the software switch base code, the library underwent several modifications in old messages and grew with the addition of OpenFlow 1.3 messages.

Every OpenFlow message represented by the Oflib has a common header. This header struct contains only one member, which is the message type information. Using the same initial struct for every message struct allows the implementation of two general functions that abstract marshaling and unmarshaling. In the Listing 4.1, we show the definition of these functions. Marshaling, also known as packing, is done by *ofl_msg_pack*. By passing a pointer to the struct *ofl_msg_header* for the function, we can check the message type and apply the message respective marshaling function. Unmarshaling, also known as unpacking, is performed by *ofl_msg_unpack*. In this function, the first bytes of the Openflow messages, the *buf* parameter, reveal their types. With this information the function calls the proper function to convert the message for the Oflib format.

```

1 int ofl_msg_pack(struct ofl_msg_header *msg, uint32_t xid, uint8_t **buf,
    size_t *buf_len, struct ofl_exp *exp);
2
3 ofl_err ofl_msg_unpack(uint8_t *buf, size_t buf_len, struct ofl_msg_header **
    msg, uint32_t *xid, struct ofl_exp *exp);

```

Listing 4.1 – Oflib: message pack and unpack base functions

Another Oflib task, discussed in the section 3.1.6, is message error handling. Checks for bad requests from the controller, like messages with unknown type and wrong size are performed by every unpacking function. In case of error, the function returns the OpenFlow error code for the Datapath, which creates an error message and sends it for the controller, through the Communication Channel.

Addition of new OpenFlow messages in the Oflib is a trivial task. Firstly, the developer needs to define a C struct, with *struct ofl_msg_header* as the first member. Then, write a pack and unpack function. Finally, add the new message type for *ofl_msg_pack* and *ofl_msg_unpack*. Listing 4.2 illustrates the OpenFlow 1.3 *Role Request*, implemented during our work.

```

1 struct ofl_msg_role_request {
2     struct ofl_msg_header header; /* Type OFPT_ROLE_REQUEST/OFPT_ROLE_REPLY. */
3     uint32_t role;                /* One of OFPCR_ROLE*. */
4     uint64_t generation_id;       /* Master Election Generation Id */
5 };
6
7 static ofl_err
8 ofl_msg_unpack_role_request(struct ofp_header *src, size_t *len, struct
9                             ofl_msg_header **msg)
10
11 static int
12 ofl_msg_pack_role_request(struct ofl_msg_role_request *msg, uint8_t **buf,
13                           size_t *buf_len)
14 };

```

Listing 4.2 – Oflib message Role request struct and function definition

Additionally, the Oflib also has printing functions. This is helpful for logging and debugging in the software switch.

4.1.2 OpenFlow Extended Match

When compared to OpenFlow 1.1, in the number of supported match fields, the version 1.3 of the OpenFlow protocol supports nearly twice as much fields as the former version. The growth was only possible due to changes in the match structure specification. A match structure from OpenFlow 1.1 was a fixed number of fields, carrying 88 bits of information in every message carrying a new flow. Match fields not set in the message were sent, adding unnecessary space overhead. In order to keep the protocol evolution and to support more fields, the OpenFlow Extended Match (OXM) was introduced by the OpenFlow 1.2 specification. The OXM format is Type-Length-Value (TLV) based and replaces the old fixed match structure. A less restricted definition of the match structure adds more flexibility for the insertion of new match fields. Figure 8 shows an example of how a field is defined by an OXM field and the TLV respective sizes in bits. The Type of a match field is formed by the OXM Class and OXM Field. An OXM class represents a vendor number, where 0x8000 is the basic class for the specification of the match fields. OXM Field defines the match field. In the example, the field number 15 represents the UDP protocol in OpenFlow. The last bit of the Type is left for the Has Mask field, which indicates if the match is masked or not. Finally, the Length field is the value size.

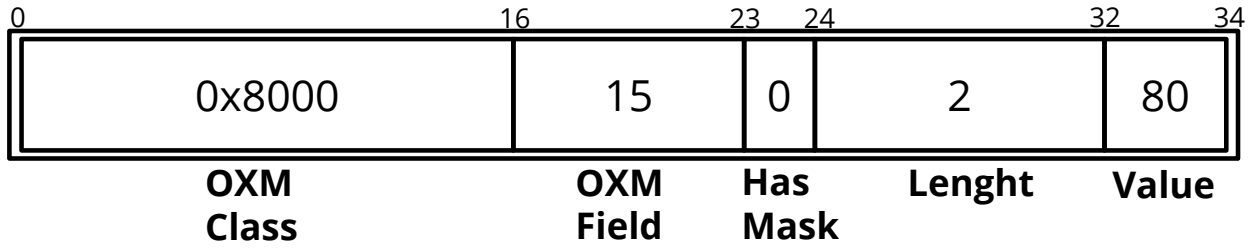


Figure 8 – OXM field example

Some challenges arise with the OXM introduction. Whereas extension of match support for messages is solved, there is nothing concerning the packet parsing in the Datapath. The next subsections discuss how our implementation deals with protocol fields extensibility in the software switch.

4.1.2.1 Packet Parser

Each new protocol added for the OpenFlow specification demands the addition of specific code to extract the new fields. Distinct protocols may have singular and complex parsing methods. For instance, variable fields such as IP options can require cumbersome deep packet inspection. For this reason, the Packet Parser implementation needs to be flexible and easy to extend. Also, the idea of simple insertion of new match fields meets the ease of extension requisite.

As a means to achieve a Packet Parser implementation featuring the mentioned characteristics, we have come up with a design which uses a packet description language to assist the parsing. Figure 9 shows the Packet Parser model implemented on the switch Datapath. Each module is described as follows:

- **NetPDL.** The Network Protocol Description Language (NetPDL)(RISSO; BALDI, 2006) is the packet description language. It is a XML-based language and has a large number of protocols and specified encapsulations. In addition, the simple language definition, allows easy and fast addition of non available protocol description. An example of how the UDP protocol is described using NetPDL can be found in the Annex A. In the Figure 9 the NetPDL module feeds the parsing library with the description of the OpenFlow 1.3 supported match fields.
- **NetBee library Parser.** Netbee is as library for packet processing (NBEE, 2012). It is composed by several modules for different types of network application, such as packet filtering and sniffing. For our Packet Parser implementation, we use the Netbee

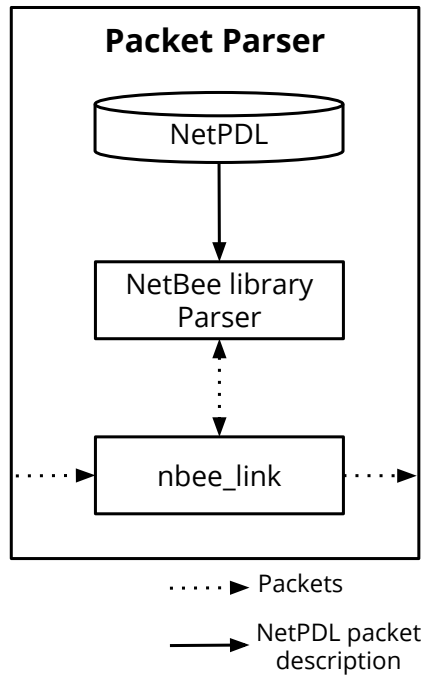


Figure 9 – Packet Parser components

library decoding objects. These objects come from a C++ set of classes and methods that ease packet decoding. To accomplish this, firstly Netbee loads the NetPDL protocols specification into the machine Random Access memory (RAM) and on a packet. Then, received packets are decoded according to the NetPDL description and the extracted information are stored in a protocol tree. Finally, packet field values can be retrieved from the tree using specific methods of the library.

- **nbee_link.** This module is where packets are converted in the flow match structure. Arriving packets are sent for the Netbee library for decoding. From the protocols tree generated by Netbee, the nbee_link module extracts the field values and builds the packet match structure that will be sent to the Flow Table look up. The code to extract a protocol is shown by Listing 4.3. Using the Netbee method *GetPDMLField*, we get the three ethernet protocol supported fields in OpenFlow 1.3. The second argument of *GetPDMLField* reflects the field name defined in the NetPDL specification. The function *nblink_extract_proto_fields* receives the extracted field value and type and inserts this into the match structure. Another important piece of code is present in the third line. For possible further processing, for instance, the application of a *set field* action, a reference to the protocol position needs to be stored.

```

1      if (protocol_Name.compare("ethernet") == 0 && pkt_proto->eth == NULL)
2      {

```

```

3      pkt_proto->eth = (struct eth_header *) ( (uint8_t*) pktin->data +
          proto->Position);
4      PDMLReader->GetPDMLField(proto->Name, (char*) "dst", proto->
          FirstField, &field);
5      nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_ETH_DST)
          ;
6      PDMLReader->GetPDMLField(proto->Name, (char*) "src", proto->
          FirstField, &field);
7      nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_ETH_SRC)
          ;
8      PDMLReader->GetPDMLField(proto->Name, (char*) "type", proto->
          FirstField, &field);
9      nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_ETH_TYPE
          );
10     }

```

Listing 4.3 – Ethernet parsing in the nbee_link module

An example of how helpful is a flexible design for the Packer Parser is on the support for IPv6 Extension Headers (EH) (DEERING; HINDEN, 1998). EHs parsing execution is not a trivial task, as there are different types and formats. What is more, IPv6 packets may present complex combinations of headers. In OpenFlow 1.3 support for IPv6 EHs is not based in values, but in a special bitmap that matches in the presence of EHs. Besides, a bit field to matches an IPv6 packet only if their EHs are in the recommended order. All of these details would result in a large ammount of code to parse EHs correctly, however, it is done in few lines, due to our extensible implementation and the NetPDL language.

4.1.2.2 Flow Match Prerequisites

Another change brought by OXMs is the introduction of flow match fields prerequisites. In order to obtain flow match consistency, some match fields require the presence of other fields. For example, matching any ARP protocol field requires the ethertype field having the correct value for an ARP packet. Thereby, inconsistent flows are denied by the Flow Table.

To map OXM fields prerequisites, a file ², with several C Preprocessor macros, was created. The macros map each field with their respective network layer 2, layer 3 or upper level requisite. In addition there is also a field that tells if a field is maskable or not. Listing 4.4 shows prerequisites and fields macros definition. Also, it gives an example of a field created

² This file was inspired by the old way that OVS handled the Nicira Extended Match (NXM) format. NXM is the format that gave origin to OXM.

by the *DEFINE_FIELD* macro.

```

1
2 #define OXM_DL_NONE      (0 , 0)
3 #define OXM_DL_ARP       (ETH_TYPE_ARP, 0)
4 #define OXM_DL_PBB       (ETH_TYPE_PBB,0)
5 #define OXM_DL_IP        (ETH_TYPE_IP, 0)
6 #define OXM_DL_MPLS      (ETH_TYPE_MPLS, ETH_TYPE_MPLS_MCAST)
7 #define OXM_DL_IPV6      (ETH_TYPE_IPV6, 0)
8 #define OXM_DL_IP_ANY    (ETH_TYPE_IP, ETH_TYPE_IPV6)
9
10 #define DEFINE_FIELD_M(HEADER, DL_TYPES, NW_PROTO, MASKABLE) \
11     DEFINE_FIELD(HEADER, DL_TYPES, NW_PROTO, MASKABLE) \
12     DEFINE_FIELD(HEADER##_W, DL_TYPES, NW_PROTO, true)
13
14 DEFINE_FIELD      (OF_TCP_SRC,          OXM_DL_IP_ANY,    IPPROTO_TCP,    false)

```

Listing 4.4 – Ethernet parsing in the `nbee_link` module

OXM matches definitions are loaded by the `Oflib`, and used in the function `oxm_pull_match`, which is called during the match unpack. Among the tests performed to detect invalid OXM fields are: bad prerequisite, duplicate fields, wrong masked and nonexistent field.

4.1.2.3 Flow Matching

In the pursuit for the best way to perform flow matching inside the flow table, developers might want to try different algorithms and data structures. For this reason, the switch implements a flexible and easy interface to change the way packets are matched.

Match fields are part of the software switch *flow_entry* struct. Instead of defining a fixed match as one of the *flow_entry* member, a pointer to `Oflib struct ofl_match_header` is left as a reference for the entry match fields. Therefore, if a developer wants to experiment his own match structure, there is only the need to make it start with an *ofl_match_header*.

This work presents a default flow matching using the `Oflib` match structure called *ofl_match*. Besides the match header, the struct includes a Hash Map structure to store OXM TLVs. Each OXM entries in the Hash Map has an exclusive key, created by the combination of the field Type and Length information. Storing only flow specified fields saves memory space, at a small cost of the pointers created to maintain the data structure. Another advantage in the Hash Map use in the match structure is the constant access time for the OXMs. Fast element access is very important for two of the most common operations:

- **Check packet matching.** Packet fields are extracted and matched against the flows. Matching is performed by look ups of the packet fields in the Hash Map.
- **Check flow collision.** Flows collide when a new flow is installed and the Flow Table contains a flow with the same match fields and priority. In this case the old one is replaced by the new one. The Hash Map allows a direct comparison of fields.

Another detail about flow matching in the software switch is about the linear behavior of Flow Table look up. The Flow Table stores flows in a list ordered by priority. When a packet is sent to the flow match it loops through the flow list until finding a matching rule or reach its end. This is the most simple approach for the flow match and was chosen for its simplicity. Developers who might want to modify the behaviour of Flow Table look up just need to add their own code for the function *flow_table_lookup*.

4.1.2.4 Extensible context expression in 'packet-in'

Former *packet-in* message contained little information about the packet parsed in the Datapath. The only match field present was the switch input port. In order to get the other packet fields a controller needs to parse the packet header, included in the end of *packet-in*. This causes an unnecessary parsing repetition in the control plane. With the OXM introduction, OpenFlow 1.3 solves this problem sending the extracted packet fields in the form of OXMs, making it easier for the control plane to retrieve the packet fields.

While an standard switch implementation requires only context information, which are input port, metadata and *tunnel_id*, our implementation follows the option to add all parsed fields in a *packet-in* message.

4.1.3 Set Field action

Support for rewriting packet fields exists since the first OpenFlow version. However, it was limited to a small set of fields. In OpenFlow 1.3, with the OXM introduction, a *flow mod* message can carry a *set field* action with any of the OXMs defined by the specification. It is up for the switch designers to decide which fields are allowed for overwrite.

Implementation of *set field* is a little bit tricky, as the consistence is achieved through the match fields. For instance, a flow with a *set field* action to rewrite the IP source address, needs to present in the match fields the same ethertype - 0x800 in hexadecimal - of the IP protocol. How the pack and unpack of match fields and actions is performed by different functions, needs to be checked in the Datapath. When handling a new *flow mod* message, the

Flow Table calls the function `dp_actions_check_set_field_req`. This function uses an `Oflib` function to check if the prerequisites are ok and validate the action.

Another frequent task caused by rewriting fields is protocol CheckSum recalculation. Fields like the IP source and destination, in the case of change, requires recalculation of IP and TCP CheckSum values. Fortunately these protocols CheckSum calculation is very simple (BRADEN *et al.*, 1988). This is not the case for the SCTP protocol (STONE *et al.*, 2002). SCTP CheckSum is calculated using a Cyclic Redundancy Check (CRC). In order to recalculate the SCTP CheckSum value we used a Python program named `pycrc`³. The program takes as input the CRC polynomial and generates all the functions necessary for the calculations. Listings 4.5 shows the code to rewrite the SCTP destination port. In the packet field rewriting we attribute a pointer to the protocol struct representation to the packet position obtained by the Packet Parser. Doing so, we can easily change the current value of the action value.

```

1 case OXM_OF_SCTP_DST: {
2     crc_t crc;
3     struct sctp_header *sctp = pkt->handle_std->proto->sctp;
4     size_t len = ((uint8_t*) ofpbuf_tail(pkt->handle_std->pkt->
5         buffer)) - (uint8_t *) sctp;
6     uint16_t v = htons(*(uint16_t*) act->field->value);
7     sctp->sctp_csum = 0;
8     memcpy(&sctp->sctp_dst, &v, OXM_LENGTH(act->field->header));
9     crc = crc_init();
10    crc = crc_update(crc, (unsigned char*)sctp, len);
11    crc = crc_finalize(crc);
12    sctp->sctp_csum = crc;
13    break;
14 }
```

Listing 4.5 – Ethernet parsing in the `nbee_link` module

4.1.4 Per-flow Metering

The Meter Table implementation follows the architectural details and responsibilities of the element described on section 3.1.5. Firstly we defined a structure for the Meter Table. The main components are the table features, such as the max number of entries and supported band types, and a Hash Map of meter entries. Other members include a reference pointer to

³ `pycrc` v0.8.2, Available at <http://www.tty1.net/pycrc/>

the Datapath, allowing a Meter Table to call the function to send OpenFlow messages; and two counters: one for the number of meter entries and another one for the quantity of bands. Secondly, we implemented a set of functions: init and destruction of the Meter Table; *meter mod* and *meter features* messages handlers; find and apply a meter entry.

Structures of meter entries are composed of a configuration, which contains information about the meter id and meter bands, and a struct for recording statistics. In addition, the meter entry has pointers to the Datapath and the Meter Table, similar to what is done in the Meter Table struct. Finally, it has a list of flow references. If the meter entry is deleted, all flows that send packets to the meter entry are deleted.

Meter entry bands are chosen accordingly to a configured rate - in Kilo packets (Kpps) per second or Kilobits per second (Kbps). Thus, it is necessary to measure the flow matched packets in function of one of the specified unities. The first idea to implement rate measuring scheme, considered the use of matched flow counters, divided by the number of matched bytes by some time interval. Although easy to implement, this approach proved inaccurate after some attempts to limit the bandwidth between two hosts connected to the switch.

After a better analysis of the task and some literature research we found and implemented a simple and efficient algorithm used for rate policy: the Token Bucket (TANENBAUM, 2002). Figure 10 illustrates how the Token Bucket works within a meter band. Basically each meter band has a bucket attached to it. At every second the bucket is refilled with a numbers of tokens equal to the meter rate. When a packet is sent to the Meter Table, it goes through each band bucket belonging to the meter entry. Inside the bucket, packets consume a number of tokens equal to their size. If there are enough tokens, the OpenFlow pipeline continues processing the packet, otherwise, the meter band is chosen and executed.

4.1.5 Connection Features

Network control protocols must be designed with scalability and high availability on mind. Node failures and high traffic loads may cause frustration for early adopters of new technologies, as these two important points usually are not considered by initial versions. Previous OpenFlow versions fall into this category of protocols, criticized by the lack of mechanisms to handle control plane issues.

More recent OpenFlow versions try to address control plane scalability and high availability with the addition of new features for OpenFlow connections. Auxiliary connections allow higher scalability for message exchanging, while controller roles try to enable fast failover for OpenFlow controllers. Event filtering, in turn, may be seen as a mechanism that sum up

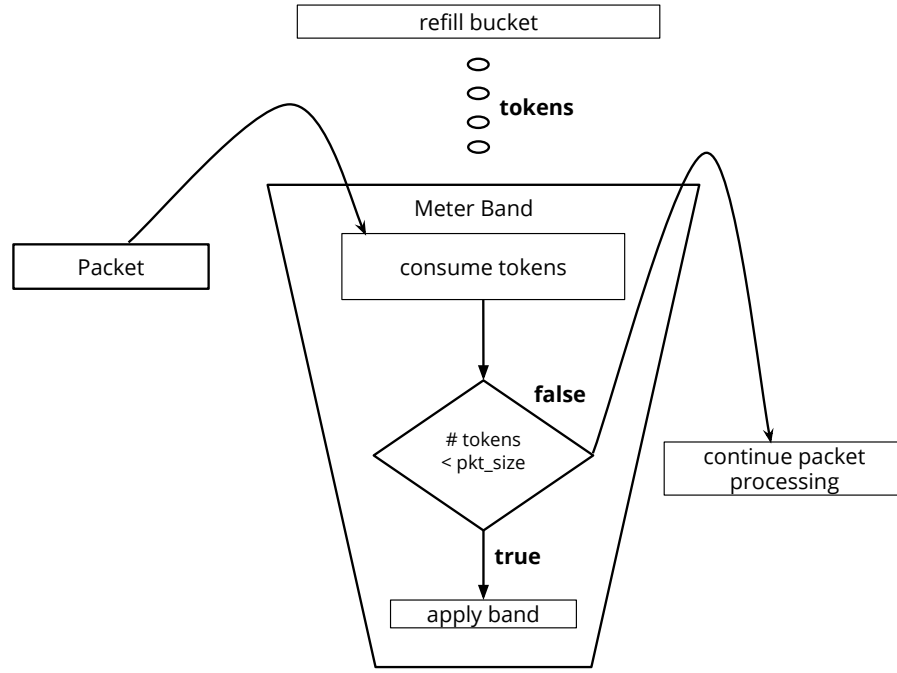


Figure 10 – Token Bucket Algorithm illustration inside the meter band

on these two topics. In the next subsections we will see a more detailed description of each feature and how they have been implemented in our software switch.

4.1.5.1 Auxiliary Connections

Auxiliary connections allow a controller to create more than one Communication Channel with a single switch. These connections add the possibility to exploit message parallelism and create a channel for specific types of messages. For instance, a controller can use one connection only for *packet-in* messages.

As a proof-of-concept we have implemented basic support for auxiliary connections. In our implementation, there is support for only one additional channel and it only carries *packet-in* messages. The following items show steps added in the switch code to handle auxiliary connection.

- The software switch sends OpenFlow messages, for the Communication Channel, encapsulated into a struct called *obuf*. The struct is a buffer that holds information such as the pointer for the first allocated byte and the size of bytes in use. In order to identify which connection is being used to receive or send the message, we have added a new member called *conn_id*. The possible values for *conn_id* are `MAIN_CONNECTION` and `PTIN_CONNECTION`.

- A new connection listener was added to the Datapath. If an auxiliary connection is specified when running the datapath, the auxiliary listener is opened after the main connection listener.
- When the Datapath talks to remotes, it searches for the auxiliary connection. If the connection exists, it processes messages received by the connection.
- On sending OpenFlow messages, the switch by default maps to the MAIN_CONNECTION. If the message is a reply from a sent message of a sender connection, the connection id is set to the same id used by the sender. In the last case, if the message type is a *packet-in*, the switch uses PTIN_CONNECTION for the connection id.

The start an auxiliary connection from one controller is disabled by default in our software switch standard program execution. To enable auxiliary connections a user should specify the *multiconn* option in the command line option

4.1.5.2 Controller Role

Controller Role is a mechanism to permit connection of multiple controllers with different duties. One of the possible use cases of roles is for fast failover, in which when the main controller goes down, a backup controller assumes the switch command. There are three possible roles for controllers: Master, Slave and Equal. A master controller has permissions to send and receive any type of OpenFlow messages. Slaves have very strict default permissions, allowed only to receive a specific set of messages. The last role, Equal, is the default role when a controller connects to the switch and the other controllers connected do not have a defined role.

Role election is totally driven by the control plane, however some additional code is required for the switch. In order to implement controller role support in our software switch we first filtered asynchronous messages received by Slave controllers. Then we restricted slaves to send only read state messages, for example, *flow stats* and *table stats*. The last insertion is the algorithm defined by the specification to handle the *role request* generation_id. Messages with a generation id smaller than previous generation ids seen by the switch are discarded. Listing 4.6 presents the function that implements the algorithm.

```

1 static ofl_err
2 dp_check_generation_id(struct datapath *dp, uint64_t new_gen_id){
3
4     if(dp->generation_id >= 0 && ((uint64_t)(new_gen_id - dp->generation_id)
5         < 0) )
6         return ofl_error(OFPET_ROLE_REQUEST_FAILED, OFPRRFC_STALE);
7     else dp->generation_id = new_gen_id;
8     return 0;
9 }

```

Listing 4.6 – Ethernet parsing in the nbee_link module

4.1.5.3 Event Filtering

Event Filtering enables controllers to filter undesired asynchronous messages, sent by the switch. Filtering of asynchronous messages is possible for three types: *port status*, *packet-in* and *flow removed*. In addition, a controller can also choose to not receive these message types for the generation reason. For example, a *packet-in* can be generated by an action to output the packet for the controller. This feature, along with auxiliary connections, gives power for controllers to create exclusive message channels.

Message filtering is handled by the Datapath. On a *set async request*, the Datapath sets the controller remote channel with bitmaps values sent in the message defined by the OpenFlow 1.3 specification, shown on Listing 4.7. Each bit set in the bitmap represents a message type and a reason. For instance, a bit with value 4 in *flow_removed_mask[0]*, determines if the controller will receive *flow removed messages* with reason OFPRR_DELETE when the role is Master.

Filtering happens before the sending of an OpenFlow message. The Datapath function to send an OpenFlow buffer through the Communication Channel checks the remote configuration and the type of the message to be sent. If it is one of the three asynchronous messages and the reason and the controller roles matches the remote filtering configuration, the message is dropped.

```

1
2 /* Asynchronous message configuration. */
3 struct ofp_async_config {
4     struct ofp_header header;
5     /* OFPT_GET_ASYNC_REPLY or OFPT_SET_ASYNC. */
6     uint32_t packet_in_mask[2];
7     /* Bitmasks of OFPR_* values. */

```

```

8     uint32_t port_status_mask[2]; /* Bitmasks of OFPPR_* values. */
9     uint32_t flow_removed_mask[2]; /* Bitmasks of OFPRR_* values. */
10 };
11 OFP_ASSERT(sizeof(struct ofp_async_config) == 32);

```

Listing 4.7 – Ethernet parsing in the nbee_link module

4.1.6 Other Changes

There is a complete list of other changes we have made to upgrade the base software switch from OpenFlow 1.1 to OpenFlow 1.3. Whereas these other changes are important the implementation demanded less effort than changes described in the previous subsections. For this reason, we will list and comment them briefly:

- **Table Miss.** Previous behavior of OpenFlow switches on non matching packets were defined by configuration flags. OpenFlow 1.3 removes these flags and defines table-miss flow entry. This entry is an all field matching with the lowest possible priority. Table miss support implementation required the remotion of code to handle the old behavior. In addition, in the case of a table-miss entry with an action to output the packet for the controller, the switch sends the *packet-in* message with a reason of type *OFPR_NO_MATCH*.
- **Rework Tag Order.** The order of the supported protocols tags, pushed by an OpenFlow action, was dictated by the specification. In OpenFlow 1.3, tags do not have a right order and should be pushed in the outermost possible position. In the code, these features were reflected by deletion of old restrictions and right adjustment of the tag order.
- **Addition of MPLS BOS and PBB fields.** The MPLS Bottom of the Stack (BOS) field and support for the Provider Backbone Bridge Protocol (PBB) was quite easy to implement thanks to the Packet Parser design, described in the subsection 4.1.2.1. PBB support also included actions to push and to pop tags in a packet. These actions are similiar to MPLS and VLAN push and pop, thus implementation for PBB followed a workflow similar to the mentioned protocols.
- **Duration for stats.** Old versions included the time of existence only for flow entries statistics. OpenFlow 1.3 introduces a duration field for meter, port, queue and group statistics and they are supported by the software switch.

4.1.7 Dpctl

Dpctl is a useful tool for management and debugging of the tables of a single OpenFlow. By using Dpctl, it is possible to avoid addition of debugging code in a controller application, for example, query the current Flow Table state. Available since the OpenFlow 1.0 reference implementation, we find its upgrade quite necessary to help during our switch development.

To connect with Dpctl, the software switch keeps a passive listening port. Unlike the switch active sockets, which looks for a controller to connect, the passive port waits for a connection. Thereby, Dpctl must initiate an active connection in order to establish contact with the switch. The switch port number for incoming connections is 6634.

Several changes were required in order to upgrade Dpctl for OpenFlow 1.3. New commands had to be implemented for meter, table stats and features, along with new arguments for existing commands, like flow mod, such as instructions and recently modified or added match fields. As each command sends a different message for the switch, the independence of OFlib comes in handy for this tasks.

Dpctl uses OFlib to create and receive switch messages and also to print their contents. After command parsing, the arguments are stored in the respective OFlib structures, for example, a *meter-mod* command creates the *struct ofl_msg_meter_mod*. The struct is packed using the OFlib function and then sent for the switch. Commands like *stats-flow* always generate an answer, which on receipt are unpacked by OFlib and the results are displayed on the screen. Check for message delivery on Dpctl accomplished through the use of OpenFlow barrier messages. After every message a barrier is sent for the switch to confirm the receipt.

4.2 Open source development

An important aspect about the software switch development is the project open source nature. The code is distributed under the BSD 3-clause license, a permissive license with few restrictions about software redistribution. This type of license fits with our main motivation, because it gives freedom to use and encourages collaboration.

The code is available on GitHub ⁴, a well known web site for projects that adopt git (GIT, 2005) as the source control management tool. Git is ideal for distributed development, as developers keep a local copy of the current code state and usually make changes to local branches. These changes are tested, reviewed and then merged into main, typically named master branch.

⁴ Project Page: <https://github.com/CPqD/ofsoftswitch13>

In this section we will show how this model works in our development process and show how it helps in the process of code maintenance and support.

4.2.1 Development workflow

GitHub is a very social platform and leverages code collaboration for open source projects. For this reason, code development follows a very simple and common workflow for public git projects hosted at GitHub:

- Developers fork the code and create local branches of new features or fixes.
- Changes are made in the branch and committed.
- A Pull Request is sent to the main repository.
- Code is reviewed and tested.
- Changes are merged into master.

This steps enabled collaboration of developers from all around the world. Also, this work model adds transparency to the development, allowing anyone from the community to track changes, enhancements and bugs. The last benefit is fairness, as every git commit is signed with the author name, credits are guaranteed to be given and shown for all contributors.

4.2.2 Code maintenance

In traditional Waterfall model, software life cycle maintenance is an independent phase and comes after implementation and testing steps (RUPARELIA, 2010). While this development model is well suited for large and well defined projects, it lacks flexibility and the dynamism required for an open source project that demands fast reactions to bugs and inquires for enhancements. Therefore, maintenance and support are processes that walk side by side with implementation and test.

The most common of maintenance tasks are usually triggered by users requests on GitHub issue tracker. Users are free to open tickets asking for enhancements or to report bugs. The issue tracker is also a common place for questions about the software switch code or execution. Thus, maintenance and support are very close in the switch development.

Another important aspect of code maintenance are regression tests. After every new feature implemented or issue fixed, we run tests frameworks presented in section 2.4, ensuring

that no change caused a break in functional code. Furthermore, Ryu tests run automatically after a new commit, since their developers keep an infrastructure to detect changes, execute the tests and publish results on a public web page ⁵.

The idea of feature testing gives light for possible evolution in our development model. A Test Driven Development (TDD) (NAGAPPAN *et al.*, 2008), a development approach where tests are written first, before the software functionalities. A TDD based methodology would force continuous maintenance in the code, because developers, when designing tests, are already thinking about the system behavior and how it going to be used and. Moreover, they might preview further changes. Thus, leading to implementation of more maintainable code.

One possible parallel with TDD, is the ONF Extension Working Group. This group is responsible for suggestion and validation of new OpenFlow features. Functionality approval process goes through proposal and test implementation in some of the available OpenFlow software switches. This can be seen as the test design phase of TDD. After validation, i.e running the test and confirm it is working, the feature is ready to be written in the specification, just like code ready for production.

⁵ Ryu Certification - <http://osrg.github.io/ryu/certification.html>

5 Evaluation

In this chapter we evaluate our work in terms of the requisites presented on section 1.2. The first section shows in numbers how many features are covered by the software switch. Following, in the next section, we present the results of performance benchmarks tests. The last chapter section is a qualitative evaluation about the code ease to change. We will highlight some known examples that demonstrate the software switch code simplicity.

The software switch evaluated version dates from the last commit pushed to GitHub. The box below shows the dates and last code changes description.

- **commit** cb740bd2565ac7e5d61ebe30ee75160a5452a033
- **Commit:** Eder Leão Fernandes <ederleaofernandes@gmail.com>
- **CommitDate:** Mon Feb 23 18:42:49 2015 -0300

Add flags member to ofp_flow_stats.

Fix missing flags field in the response of a flow stats request.

5.1 Feature Completeness

Evaluate the proper operation of the OpenFlow switch features is not a trivial task because of multiple and rich configurations allowed by the specification. For example, test all flow match fields combination would require creation of a large number of flows and packets, making manual tests very time consuming. For this reason, automatic tests frameworks, discussed on section 2.4, are the best options to test the switch functionality in order to evaluate features completeness.

OFTest and Ryu Certification are the two test frameworks used for switch validation. As mentioned in chapter 4.2.2, both are important tools for the software switch development. While Ryu certification has a strong focus on validation of the Datapath, OFTest offers a nice set of test cases for control and data plane message exchange. In the next sections we present a resume of the results obtained.

5.1.1 OFTest results

Testing in OFTest is simple as it provides scripts in Python to run the switch and the test cases. Each test case starts a controller which connects with a running switch, executes the test instructions and checks the switch answers.

Some controller to switch, like a *flow stats* request, and symmetric messages demand an answer from the switch. Thus, the main purpose of the framework usage with the software switch is for message handling validation. Although, OFTest has capabilities to evaluate the pipeline processing - for instance, checking if a packet was correctly forwarded by a flow - we found in Ryu a more comprehensive test set for this task.

Table 7 shows test results for basic OpenFlow messages. The major type of messages of the the test set are messages to query information about the state of manifold switch elements, such as *GroupFeatureStats* and *MeterStats*. Also, there are some configuration messages, like the *PortConfigMod*. In all tests the switch returned the right answer for the control plane.

Table 7 – Basic OpenFlow messages

Message	Result	Message	Result
AggregateStats	ok	GroupFeaturesStats	ok
AsyncConfigGet	ok	GroupStats	ok
DescStats	ok	MeterConfigStats	ok
Echo	ok	MeterFeaturesStats	ok
EchoWithData	ok	MeterStats	ok
FeaturesRequest	ok	QueueStats	ok
FlowStats	ok	PortConfigMod	ok
FlowRemoveAll	ok	PortDescStats	ok
GroupDescStats	ok	TableStats	ok

Controller roles test results are shown in table 8. This tests checks if the software switch changes correctly controller roles and if the respective permission police is respected. As in the previous message tests, all role tests were sucessful.

Table 8 – Role request message results

Role Request Tests	Results
RoleRequestEqualToSlave	ok
RoleRequestSlaveToMaster	ok
RolePermissions	ok
RoleRequestEqualToMaster	ok
RoleRequestNochange	ok
SlaveNoPacketIn	ok

5.1.2 Ryu Certification results

The Ryu Certification tests are divided into five categories: Action, Set-Field, Match, Group and Meter. The test sets of each category are very comprehensive, with tests for different packet types.

Table 9 is a resume of test results - the complete list of test cases can be found on Annex C. - for this work compared to the other three switches presented on chapter 2. White cells give the number of tests passed, while grey cells show the number of test cases that returned an error. Tests are divided by each category, with the last two columns giving the total sum of working and non working features. The first row presents the results for this work.¹

Table 9 – Ryu Certification results comparison

Switch	Action		Set-Field		Match		Group		Meter		Total	
ofsoftswitch13	50	6	159	7	708	6	15	0	30	6	848	25
Open vSwitch	34	22	96	74	534	180	0	36	0	36	670	321
LINC	24	32	68	102	428	286	3	12	0	24	523	456
Trema	50	6	159	11	708	6	15	0	34	2	852	25

Results show that the software switch has a higher number of working features than Open vSwitch and LINC. With only 25 errors, it is tied with Trema in the number of supported features. There is a small difference between ofsoftswitch13 and Trema, in the total number of tests passing. This happens because Ryu Certification does not execute four tests due to old switch restrictions.

The values presented in this section are from the official certification site². Some failed test results presented on the site work in our internal test setup. For instance, matching on *PBB ISID* value. However, we chose to show the official results as we could not identify the causes for different results. In addition, some test may never pass. For example, *IP proto* modification causes packet malformation, because the IP proto packet field will not conform with the next layer protocol, which leads to a test failure.

5.2 Performance Benchmarks

One of the software switch requirements listed on chapter 1 is to reach a maximum throughput of at least 100 Mb/s. For this reason we evaluated the switch performance in terms

¹ ofsoftswitch13 is the software switch repository

² <http://osrg.github.io/ryu-certification/switch/config/ofsoftswitch13.html>

of network metrics. In this section we show how the switch performs for different packet sizes in comparison with other userspace software switches. Also, we investigated how performance is affected by the number of flows and by the number of tables traversed to match a packet.

The machine configuration used to perform measurement tests are listed in the box below.

- **Processor:** 8x Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz
- **Memory:** 6003MB
- **Operating:** System Ubuntu 14.04.2 LTS

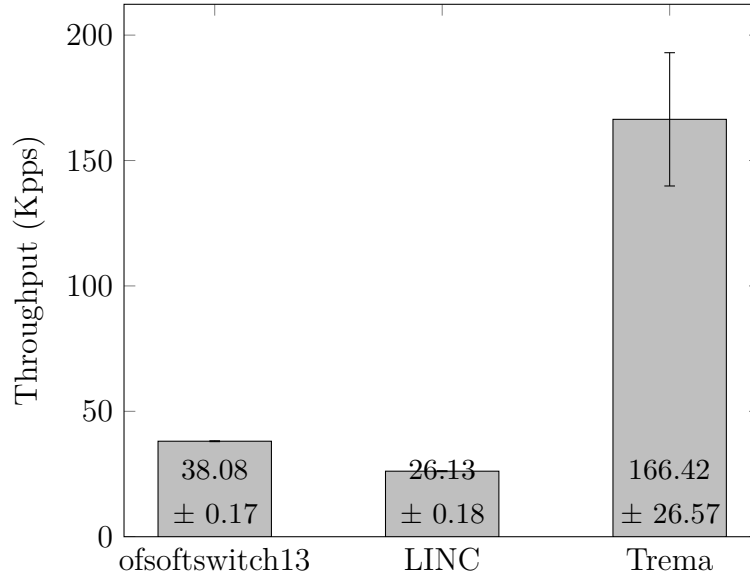
5.2.1 Maximum Throughput

This test evaluates the maximum forwarding rate the software switch can reach in comparison to other userspace implementations.

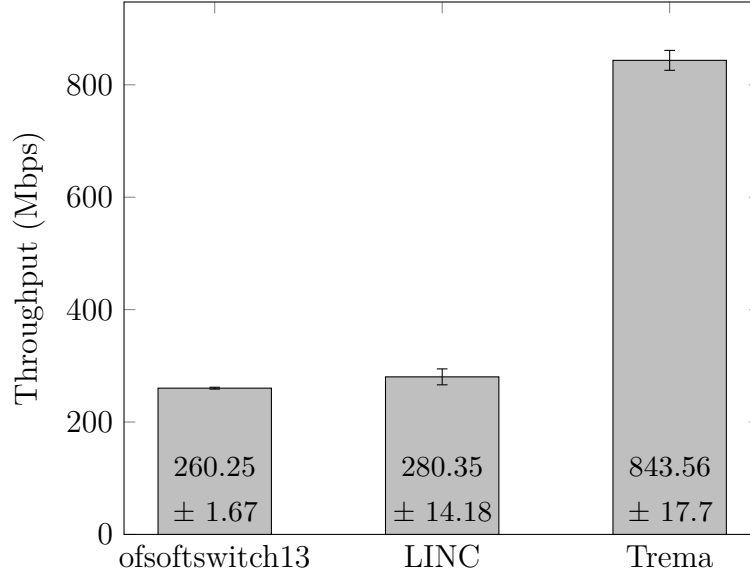
The setup for maximum throughput evaluation is the following:

- A running instance of the software switch with two virtual interfaces - Port 1 and Port 2 - attached.
- One flow installed in the flow table to match a packet sent to Port 1 with destiny ethernet 00:00:00:00:01. The action is output packet to Port 2.
- A packet traffic generator. We used a simple program named packeth (ISKRATEL, 2003).
- A script running in the linux terminal checking the current packet rate on Port 2.

The test starts by installing the flow in the switch Flow Table. After, we inject packets, using the traffic generator, directly into Port 1. The bandwidth results of Port 2, reported by the script, are used to calculate the average rate and the standard deviation. Two transmission measurements were made: for small packets of 64Kb and bigger packets of 1500Kb. Figure 11 shows results for both experiments.



(a) Comparison using packets of 64 bytes



(b) Comparison using packets of 1500 bytes

Figure 11 – User space software switches throughput comparison

Switch forwarding performance for small packets is evaluated in Kilo packets per second (Kpps). Figure 11a shows that ofsoftswitch13 can handle 38.08 Kpps. This result is very far from Trema and approximately 32% more efficient than LINC. Bigger packets are measured in Mega bits per second. Results presented in Figure 11b show that ofsoftswitch13 and LINC, with rates of 260.25 Mbps 280.35 Mbps respectively, are slower than Trema with a bandwidth of 843.56 Mbps.

Although Trema overcomes our work in performance, due to optimization like multiple

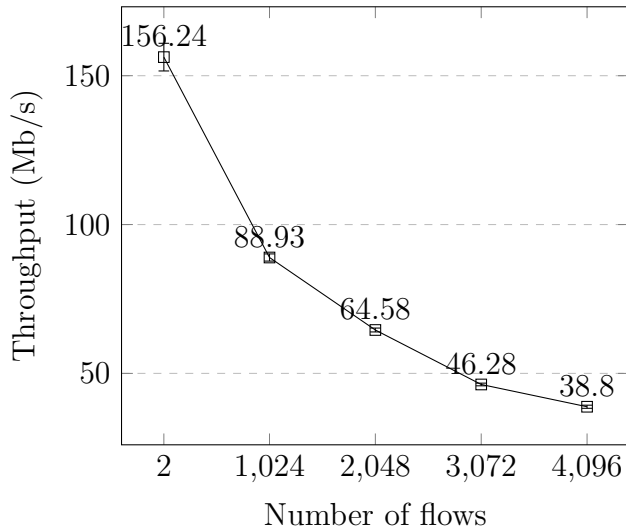
threads, the most important result of this experiment is the software switch maximum throughput found. This value is higher than the value established by the software switch requirements.

5.2.2 Throughput in function of flows and tables

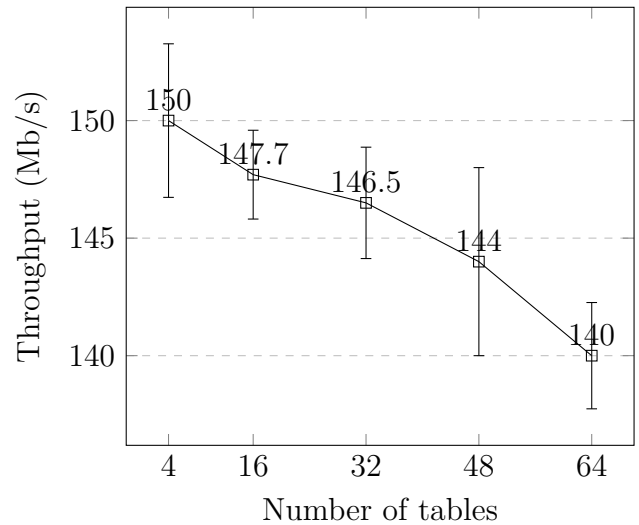
This experiment measured two factors that may affect the software switch performance. One is the number of flows installed in one table. The second is the number of tables traversed until the match.

For this test we used Mininet with the software switch connecting two hosts. Bandwidth is measured through the Iperf session established between the two hosts. Flow Table setup for the two cases are the following:

- (A) **Number of flows.** Install a certain amount of flows, with the same priority, that will not match packets. In the end, two flows, with the same or lesser priority than the previous are installed to forward packets between the two hosts.
- (B) **Number of tables.** Flows to send the packet to next table are installed until the penultimate table. Then, in the last table install two flows to forward the traffic between the hosts.



(a) Throughput per number of flows in one table



(b) Throughput per number of tables

Figure 12 – Influence of the number of installed flows on the throughput.

The graphs in the Figure 12 shows that both cases have a strong influence over the switch performance. The most sensitive case is for one table shown in Figure 12a, as the number of flows increases the throughput decreases linearly. The increase in the number of tables, shown by the graph in the Figure 12b, also causes a linear decrease in the packet rate, however it is smaller than in the first case. These results were already expected, because the software switch implements linear matching. Thus, this experiments were important to verify one improvement point for the software switch.

5.2.3 Ping Round Trip Time

Round Trip Time (RTT) is the time between a data request and answer. Several factors might affect the total RTT and influence network latency. Two examples are the number of nodes between two communicating hosts and the transmission medium. The time a packet takes to enter and leave a switch is also considered for the RTT. Thus, it is important to measure how much the software switch affects the RTT.

In order to measure the RTT between two hosts connected by our software switch and also LINC and Trema, the following steps are executed:

1. Creation of two linux containers (LXC) - Host 1 and Host 2 - with a pair of virtual interfaces *veth0* and *veth1*. LXC is an operating system lightweight virtualization technology, in which is possible to run multiple isolated linux instances as containers. With LXC, we run two containers to serve as the network hosts.
2. Execution of a software switch instance of a software switch with the container virtual ports attached on switch interfaces.
3. Installation of two flows in the switch Flow Table to forward the traffic between the two hosts.
4. Configuration of Host 1 and Host 2 with IP addresses in the same network. In our test Host 1 is configured with the IP address 192.168.0.1 and host 2 as 192.168.0.2.
5. Execution of the *ping* program in Host 1 to ping the address 192.168.0.2. Ping is a program to send and measure the time between an Internet Control Message Protocol (ICMP) "Echo request" and the ICMP "Echo Reply". The number of Echo requests sent is 100 and the packet sizes are 64Kb.

Switch results comparison is shown in Table 10. This tests give a good approximation for the software switch impact over the network delay, because its connected directly to the

hosts. As expected, because of the previous results, Trema is the most efficient among the userspace software switches. The ofsoftswitch13 obtain a low minimum RTT, with 0.304 ms, however the average is approximately 1ms. LINC has a very high RTT, with more than a half second to complete. This is not a surprise, because the throughput tests, shown in section 5.2.1, revealed that LINC does not handle small packets very fast.

An acceptable RTT value depends on the application running over the network. Latency sensitive programs, like multiplayer online games, benefit from a low RTT. Considering a small network, with not many hops, the RTT in our software switch is acceptable.

Table 10 – Ping Round Trip Time comparison between software switches

Software Switch	Minimum	Average	Maximum	Standard Deviation
ofsoftswitch13	0.304	1.077	1.821	0.313
LINC	303.897	554.763	821.482	253.034
Trema	0.120	0.400	0.473	0.040

5.3 Portability

Software portability is the ability to compile and run a program in different hardware architectures. For a network environment, more specifically OpenFlow, portability allows the richer testbeds. Proposed as a friendly experimentation tool for multiple environments the software switch implementation enables portability with few platform dependant modifications. Based on build scripts to install the OpenFlow 1.0 software switch on an OpenWRT (OpenWRT, 2004) operating system image, (YIAKOUMIS *et al.*, 2011), we demonstrated portability building our OpenFlow 1.3 software switch for OpenWRT and running in a home wireless router.

The wireless router model for the software switch port is the TP-LINK TL-WR1043ND. This router already comes with a default OpenWRT image, however it is necessary to build a new image containing the software switch installed as an operational system package.

- **Enhanced portability for different architectures.** Previously, the implementation only considered Intel based - i686 and x86_64, architectures. Byte order conversions were necessary because Intel processors byte-order are Little-endian, while the network follows the the Big-endian order. The MIPS processor of the wireless router model, follows the same byte-order from the network. Standard Linux byte-order functions, from the library *netinet/in.h*, does not check the system architecture and change the

byte-order whatever the type of conversion called. For instance, if we call the function `htons`, to change the byte-order to network from host, and the value is already in the correct order, data is changed anyway. Thus, to avoid wrong values, we implemented byte-order conversions functions that check the system architecture before calling the standard procedures from *netinet/in.h*.

- **Netbee Remotion.** After the first execution we realized that the switch was consuming too much memory of the router limited amount of RAM. From 32Mb of memory, the software switch was consuming 30Mb. After a memory profiling, we found that Netbee was the switch component most memory consuming. While it is not a problem for a server or a machine with higher capacity, it is not true for a small embedded system like the wireless router. Therefore, we removed the Netbee library, reducing the average memory usage to less than 1Mb. Since the code is well structured, the new parsing implementation was trivial. A merely redefinition of the parsing function in the packet handler interface did the job. This situation also demonstrated the code friendliness.

The implementation of an OpenFlow 1.3 switch for a wireless router opens a myriad of opportunities in the area of Software Defined Wireless Networking. Experimenters might take advantage of the new features implemented by our software switch. Flow metering, for example, is a simple yet powerful mechanism to provide bandwidth control in home environments. Also, creation of firewall blocking rules is made easy by OpenFlow, since field matching is a natural operation for an OpenFlow switch.

6 Conclusion

Six years ago SDN and OpenFlow caused a stir in the world of computer networks. Although data and control plane separation is not a new idea, the flexibility and programmability enabled by OpenFlow started a wave industry efforts to support the protocol in their products. Several OpenFlow 1.0 switches, controllers and test frameworks emerged from this movement confirming the growing interest in the technology. Large networks operators, such as Google and Facebook, embraced OpenFlow interested on its potential. An organization, named Open Network Foundation, was created to speed up the OpenFlow development and adoption. Quickly, new versions of the protocol were released. This time, however, implementations did not aroused at the same time.

To keep up with the pace of the technology and enable research that leverages the new functionalities, we found the need to implement an OpenFlow 1.3 software switch. More than a full compliant implementation requirements included minimal performance and ease of experimentation. This effort lead to the open source software implementation of the first OpenFlow 1.3 switch.

Today, the software switch is a well known open source project and a cheap and friendly option to experiment OpenFlow 1.3. Although new software switches supporting OpenFlow 1.3 are now available, this work is still a solid and relevant option to prototype and develop new OpenFlow applications. In the next two sections, we present some relevant obtained results and notorious use cases. Finally, we conclude this chapter discussing future areas for research and improvement in the software switch.

6.1 Results

In this section we list positive results achieved on the dissemination of our work:

- **Development of an Open source community.** GitHub choice to host the code proved to be a great way to reach a high number of users. Figure 13, shows information that confirm the tool popularity. In a 14 days the interval, the software switch repository had 3125 accesses, with 796 unique visitors and cloned 97 times.

With respect to code contributions, there are 15 users listed in GitHub that submitted pull requests. Also, there is a number of contributors that send patches through other means.

This result is very important because the creation of a community around open source code gives visibility, helps to spread the software and speed reports of detected bugs.

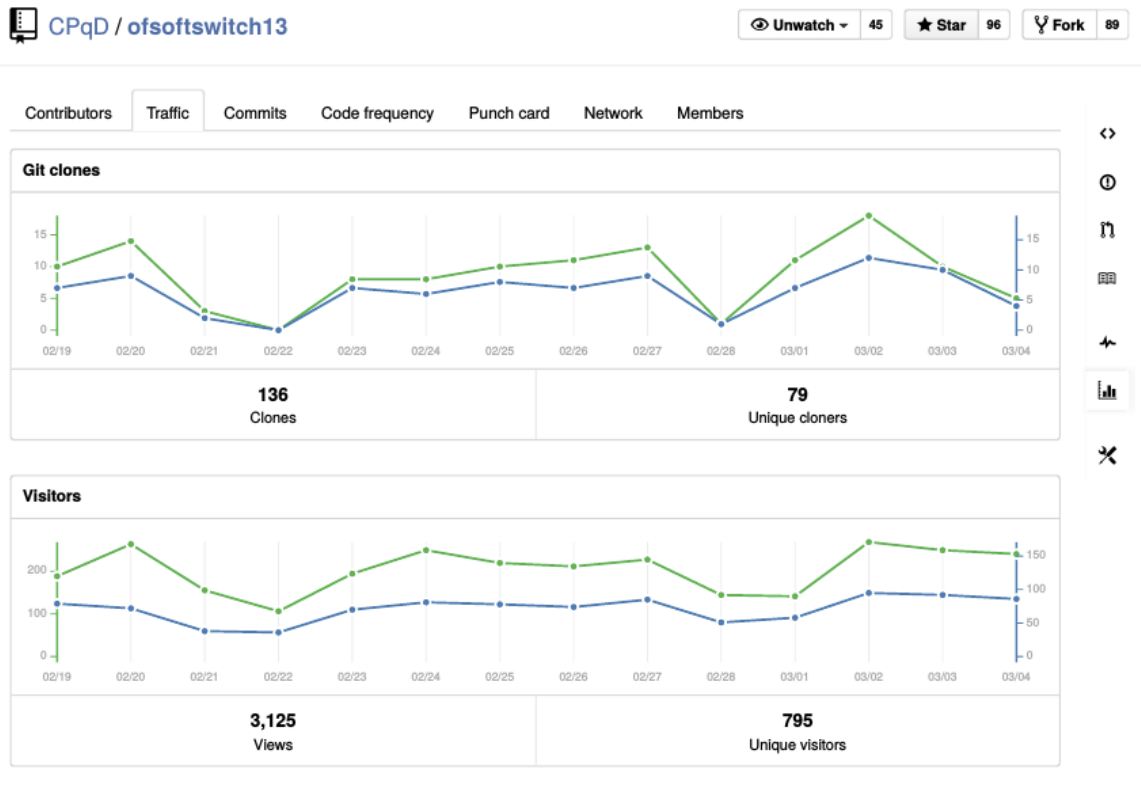


Figure 13 – GitHub statistics.

- **Part of Mininet installation options.** The software switch is included among the installation options of Mininet. It makes easier for users to start experimenting with OpenFlow 1.3. In addition, the range of people trying
- **Publications.** This work gave origin to three publications. The first paper is about IPv6 support on OpenFlow. It was the first. The second is an invited paper that gives perspectives on SDN for home network. These ideas were inspired by the software switch port to OpenWRT. Finally, the last paper is an overall presentation of this project, highlighting architectural and implementation details. These publications are listed on Annex B.

6.2 Use Cases

Known use cases show that the software is an important tool in the advance of the state of art on SDN research and development. As there is a large number of projects who are using or have made use of our work, we will list some notorious examples:

- **Base for new OpenFlow features implementation.** The ONF group responsible by the addition of new OpenFlow features decided to publish new features only if properly implemented and tested. As OpenFlow 1.3 basic functionalities is required for 1.4, the Extensions Work Group chose our work as one of the base software switches used for new functionality prototyping (TOURRILHES, 2013).
- **Academic.** The software switch has found good adoption by the academic community. For instance, works published in renowned conferences (REITBLATT *et al.*, 2013) (BIANCHI *et al.*, 2014) and master dissertations (ARORA, 2013) (SHOURMASTI, 2013) cite our software as the OpenFlow 1.3 switch chosen for their experiments .
- **Industry.** Industrial development is harder to track because this development are usually closed. However, one successful case is in the development of an application for the Open Network Operating System (ONOS). Built by two companies teams, Dell and ON.Lab, the Segment Routing implementation used the software switch for its simplicity (ONOS, 2014).

6.3 Future Work

Each architectural component from the software switch has space for improvements. New algorithms and data structures are objects of study for the Flow Table matching. More complex and precise algorithms for rate limiting might be considered for Meter Table better performance. As for groups, new bucket select types may be a subject for academic research.

While there are open ideas for further research and development, some optimizations and features are planned for the software switch in the medium-term. These major improvements are listed below:

- **Support for OpenFlow 1.4.** OpenFlow 1.4 is an expansion of OpenFlow 1.3 and it would good to keep the pace with the OpenFlow evolution. Some OpenFlow 1.4 features are already implemented, as stated in section 6.2, however we would like to have both

versions supported in a single switch running instance, without need to split the code in two different programs.

- **Hash based match.** Results found in experiments presented on section 5.2.2 show a huge loss in performance due to linear matching. To solve this problem, flows entries might be represented as hash value into the Flow Table. Then, packet fields could also be turned into a hash and looked up in the Flow Table. This would give constant performance for the Flow Table look up. However, there are some questions to answer:
 - How to handle flow priority? Since flows should be matched in order of priority, how to ensure the first hash value for a flow is the one with higher priority?
 - How to deal with field masking? Some flow match fields may have a mask, so they should be considered in the hash calculation. The question is how to efficiently search and apply these masks to the packet hash calculation.

The search for an answer for these questions opens space for new research in OpenFlow and SDN, because this questions are not only related to the software switch.

- **New packet parsing engine.** The software switch relies on Netbee library to parse packets. While Netbee adds flexibility and extensibility for the parsing and ease the addition of new protocols to OpenFlow, its code is not frequently updated, not following dependencies upgrades. This breaks the software switch compilation in more recent Linux versions, because of more recent versions of libraries required by Netbee. After Due to the number of compilation issues related to netbee, a new packet parsing module must replace the current third party library.

Bibliography

AL-SHABIBI, A.; LEENHEER, M. D.; GEROLA, M.; KOSHIBE, A.; PARULKAR, G.; SALVADORI, E.; SNOW, B. Openvirtex: Make your virtual sdns programmable. In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2014. (HotSDN '14), p. 25–30. ISBN 978-1-4503-2989-7. Disponível em: <<http://doi.acm.org/10.1145/2620728.2620741>>. Cited on page 1.

ARORA, D. *Proactive Routing in Scalable Data Centers with PARIS*. 2013. Cited on page 55.

BIANCHI, G.; BONOLA, M.; CAPONE, A.; CASCONI, C. Openstate: Programming platform-independent stateful openflow applications inside the switch. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 44, n. 2, p. 44–51, abr. 2014. ISSN 0146-4833. Disponível em: <<http://doi.acm.org/10.1145/2602204.2602211>>. Cited on page 55.

BRADEN, R.; BORMAN, D.; PARTRIDGE, C. *Computing the Internet checksum*. IETF, 1988. RFC 1071. (Request for Comments, 1071). Updated by RFC 1141. Disponível em: <<http://www.ietf.org/rfc/rfc1071.txt>>. Cited on page 33.

CHEN, Y.; FARLEY, T.; YE, N. Qos requirements of network applications on the internet. *Inf. Knowl. Syst. Manag.*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 4, n. 1, p. 55–76, jan. 2004. ISSN 1389-1995. Disponível em: <<http://dl.acm.org/citation.cfm?id=1234242.1234243>>. Cited on page 3.

DEERING, S.; HINDEN, R. *Internet Protocol, Version 6 (IPv6) Specification*. IETF, 1998. RFC 2460 (Draft Standard). (Request for Comments, 2460). Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. Disponível em: <<http://www.ietf.org/rfc/rfc2460.txt>>. Cited on page 30.

DRUTSKOY, D.; KELLER, E.; REXFORD, J. Scalable network virtualization in software-defined networks. *IEEE Internet Computing: Special*, 2012. Cited on page 9.

EMMERICH, P.; RAUMER, D.; WOHLFART, F.; CARLE, G. Performance characteristics of virtual switching. In: *3rd IEEE International Conference on Cloud Networking, CloudNet 2014, Luxembourg, Luxembourg, October 8-10, 2014*. [s.n.], 2014. p. 120–125. Disponível em: <<http://dx.doi.org/10.1109/CloudNet.2014.6968979>>. Cited on page 9.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: An intellectual history of programmable networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 44, n. 2, p. 87–98, abr. 2014. ISSN 0146-4833. Disponível em: <<http://doi.acm.org/10.1145/2602204.2602219>>. Cited on page 1.

FELDMANN, A. Internet clean-slate design: What and why? *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 37, n. 3, p. 59–64, jul. 2007. ISSN 0146-4833. Disponível em: <<http://doi.acm.org/10.1145/1273445.1273453>>. Cited on page 1.

FLOODLIGHT, P. *Floodlight*. 2011. <<http://www.projectfloodlight.org/floodlight/>>. [Access: 27-Oct-2014]. Cited on page 13.

FLOODLIGHT, P. *OF-Test*. 2011. <<http://www.projectfloodlight.org/oftest/>>. [Access: 27-Oct-2014]. Cited on page 14.

FORWARDING, F. *LINC-Switch*. 2012. <<https://github.com/FlowForwarding/LINC-Switch>>. [Access: 22-Oct-2014]. Cited on page 11.

FOUNDATION, O. N. *OF-CONFIG 1.2 - OpenFlow Management and Configuration Protocol*. 2011. <<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf>>. [Access: 22-Oct-2014]. Cited on page 12.

GIT. *git -distributed-even-if-your-workflow-isnt*. 2005. <<http://git-scm.com/>>. [Access: 23-Fev-2015]. Cited on page 39.

GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M.; MCKEOWN, N.; SHENKER, S. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 38, n. 3, p. 105–110, 2008. Cited on page 12.

ISKRATEL, M. J. *packeth - Ethernet packet generator*. 2003. [Access: 01-Mar-2015]. Disponível em: <<http://packeth.sourceforge.net/packeth/Home.html>>. Cited on page 46.

KIS, Z. L. *of11softswitch*. 2011. <<https://github.com/TrafficLab/of11softswitch>>. [Access: 01-Nov-2014]. Cited on page 11.

KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *CoRR*, abs/1406.0440, 2014. Disponível em: <<http://arxiv.org/abs/1406.0440>>. Cited on page 1.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: Rapid prototyping for software-defined networks. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2. Disponível em: <<http://doi.acm.org/10.1145/1868447.1868466>>. Cited on page 15.

MAYRHAUSER, A. von. *Software Engineering: Methods and Management*. San Diego, CA, USA: Academic Press Professional, Inc., 1990. ISBN 0-12-727320-4. Cited on page 17.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <<http://doi.acm.org/10.1145/1355734.1355746>>. Cited on page 1.

NAGAPPAN, N.; MAXIMILIEN, E. M.; BHAT, T.; WILLIAMS, L. Realizing quality improvement through test driven development: Results and experiences of four industrial teams. *Empirical Softw. Engg.*, Kluwer Academic Publishers, Hingham,

MA, USA, v. 13, n. 3, p. 289–302, jun. 2008. ISSN 1382-3256. Disponível em: <<http://dx.doi.org/10.1007/s10664-008-9062-z>>. Cited on page 41.

NBEE. *NetBee Library*. 2012. <<http://www.nbee.org/>>. [Access: 11-Nov-2014]. Cited on page 28.

NEC. *Trema switch*. 2013. <<https://github.com/FlowForwarding/LINC-Switch>>. [Access: 23-Oct-2014]. Cited on page 12.

NLANR/DAST : Iperf - The TCP/UDP Bandwidth Measurement Tool. Accessed 2007. <Http://dast.nlanr.net/Projects/Iperf/>. Disponível em: <<http://dast.nlanr.net/Projects/Iperf/>>. Cited on page 15.

NTT. *Ryu Certification*. 2013. <<http://osrg.github.io/ryu/certification.html>>. [Access: 27-Oct-2014]. Cited on page 14.

NTT. *Ryu SDN Framework*. 2013. <<http://osrg.github.io/ryu/>>. [Access: 27-Oct-2014]. Cited on page 13.

ONF. *OpenFlow 1.3 Specification*. 2012. <<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>>. [Access: 9-Nov-2014]. Cited on page 5.

ONF. *SDN Architecture*. [S.l.], 2012. Disponível em: <<http://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>>. Cited on page 1.

ONOS. *ONOS Segment Routing*. 2014. <<https://wiki.onosproject.org/display/ONOS/Installation+Guide>>. [Access: 03-Mar-2015]. Cited on page 55.

OpenWRT. 2004. Disponível em: <<http://www.openwrt.org>>. Cited on page 50.

PEPELNJAK, I. *FLOW TABLE EXPLOSION WITH OPENFLOW 1.0 (AND WHY WE NEED OPENFLOW 1.3)*. 2013. <<http://blog.ipSPACE.net/2013/10/flow-table-explosion-with-openflow-10.html>>. [Access: 7-Fev-2015]. Cited on page 20.

PFAFF, B.; PETTIT, J.; KOPONEN, T.; AMIDON, K.; CASADO, M.; SHENKER, S. e.a.: Extending networking into the virtualization layer. In: *In: 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII). New York City, NY (October 2009)*. [S.l.: s.n.], 2009. Cited on page 11.

PLUMMER, D. *Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. IETF, 1982. RFC 826 (INTERNET STANDARD). (Request for Comments, 826). Updated by RFCs 5227, 5494. Disponível em: <<http://www.ietf.org/rfc/rfc826.txt>>. Cited on page 8.

POX. *About POX*. 2014. <<http://www.noxrepo.org/pox/about-pox/>>. [Access: 27-Oct-2014]. Cited on page 13.

- REITBLATT, M.; CANINI, M.; GUHA, A.; FOSTER, N. Fattire: Declarative fault tolerance for software-defined networks. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 109–114. ISBN 978-1-4503-2178-5. Disponível em: <<http://doi.acm.org/10.1145/2491185.2491187>>. Cited on page 55.
- REYNOLDS, J.; POSTEL, J. *Assigned Numbers*. IETF, 1994. RFC 1700 (Historic). (Request for Comments, 1700). Obsoleted by RFC 3232. Disponível em: <<http://www.ietf.org/rfc/rfc1700.txt>>. Cited on page 22.
- RISSO, F.; BALDI, M. Netpdl: An extensible xml-based language for packet header description. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 50, n. 5, p. 688–706, abr. 2006. ISSN 1389-1286. Disponível em: <<http://dx.doi.org/10.1016/j.comnet.2005.05.029>>. Cited on page 28.
- RUPARELIA, N. B. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 35, n. 3, p. 8–13, maio 2010. ISSN 0163-5948. Disponível em: <<http://doi.acm.org/10.1145/1764810.1764814>>. Cited on page 40.
- SHENKER, S. Fundamental design issues for the future internet. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, v. 13, p. 1176–1188, 1995. Cited on page 1.
- SHERWOOD, R.; GIBB, G.; YAP'S, K.-K.; CASADO, M.; MCKEOWN, N.; PARULKAR, G. *FlowVisor: A Network Virtualization Layer*. [S.l.], 2009. Disponível em: <<http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf>>. Cited on page 1.
- SHOURMASTI, K. S. *Stochastic Switching Using OpenFlow*. [S.l.]: Institutt for telematikk, 2013. 70 p. Cited on page 55.
- SOMMERVILLE, I. *Software Engineering (6th Ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN 0-201-39815-X. Cited on page 17.
- STANFORD. *NetFPGA 10G OpenFlow Switch*. 2010. <<https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G-OpenFlow-Switch>>. [Access: 21-Oct-2014]. Cited on page 11.
- STANFORD. *Pantou : OpenFlow 1.0 for OpenWRT*. 2010. <http://archive.openflow.org/wk/index.php/Pantou_:__OpenFlow_1.0_for_OpenWRT>. [Access: 21-Oct-2014]. Cited on page 11.
- STONE, J.; STEWART, R.; OTIS, D. *Stream Control Transmission Protocol (SCTP) Checksum Change*. IETF, 2002. RFC 3309 (Proposed Standard). (Request for Comments, 3309). Obsoleted by RFC 4960. Disponível em: <<http://www.ietf.org/rfc/rfc3309.txt>>. Cited on page 33.
- TANENBAUM, A. Computer networks. In: _____. 4th. ed. [S.l.]: Prentice Hall Professional Technical Reference, 2002. p. 401. ISBN 0130661023. Cited on page 34.

TOOTOONCHIAN, A.; GORBUNOV, S.; GANJALI, Y.; CASADO, M.; SHERWOOD, R. On controller performance in software-defined networks. In: *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. Berkeley, CA: USENIX, 2012. Disponível em: <<https://www.usenix.org/conference/hot-ice12-0/controller-performance-software-defined-networks>>. Cited on page 12.

TOURRILHES, J. *OpenFlow prototyping*. 2013. <<https://github.com/jean2/ofsoftswitch13>>. [Access: 11-Nov-2014]. Cited on page 55.

TSENG, H.-M.; LEE, H.-L.; HU, J.-W.; LIU, T.-L.; CHANG, J.-G.; HUANG, W.-C. Network virtualization with cloud virtual switch. In: *Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems*. Washington, DC, USA: IEEE Computer Society, 2011. (ICPADS '11), p. 998–1003. ISBN 978-0-7695-4576-9. Disponível em: <<http://dx.doi.org/10.1109/ICPADS.2011.159>>. Cited on page 9.

VERTS, W. T. *An Essay on Endian Orders*. 1996. <<http://people.cs.umass.edu/~verts/cs32/endian.html>>. Cited on page 23.

VMWARE. *The VMware NSX Network Virtualization Platform*. [S.l.], 2012. Disponível em: <<http://www.vmware.com/files/pdf/products/nsx/VMware-NSX-Network-Virtualization-Platform-WP.pdf>>. Cited on page 9.

WIRESHARK. *Wireshark OpenFlow*. 2014. <<http://wiki.wireshark.org/OpenFlow>>. [Access: 27-Oct-2014]. Cited 2 times on page(s) 14 and 15.

YIAKOUMIS, Y.; SCHULZ-ZANDER, J.; ZHU, J. *Pantou : OpenFlow 1.0 for OpenWRT*. 2011. Disponível em: <http://www.openflow.org/wk/index.php/OpenFlow_1.0_for_OpenWRT>. Cited on page 50.

Annex

ANNEX A – NetPDL packet description example

```

1 <protocol name="udp" longname="UDP (User Datagram protocol)"
  showsumtemplate="udp">
2 <format>
3   <fields>
4     <field type="fixed" name="sport" longname="{0x8000 15}" size="2"
      showtemplate="FieldDec"/>
5     <field type="fixed" name="dport" longname="{0x8000 16}" size="2"
      showtemplate="FieldDec"/>
6     <field type="fixed" name="len" longname="Payload length" size="2"
      showtemplate="FieldDec"/>
7     <field type="fixed" name="crc" longname="Checksum" size="2" showtemplate
      ="FieldHex"/>
8   </fields>
9 </format>
10 <visualization>
11   <showsumtemplate name="udp">
12     <section name="next"/>
13     <text value="UDP: port "/>
14     <protofield name="sport" showdata="showvalue"/>
15     <text value=" => "/>
16     <protofield name="dport" showdata="showvalue"/>
17   </showsumtemplate>
18 </visualization>
19 </protocol>

```


ANNEX B – Publications

Three papers were published during this work and are listed below.

- Eder Leão Fernandes, Christian Esteve Rothenberg. "OpenFlow 1.3 Software Switch". In Salão de Ferramentas XXXII Simpósio Brasileiro de Redes de Computadores - SBRC'2014, Florianópolis, 5 a 9 de Maio de 2014.
- E. L. Fernandes, C. Esteve Rothenberg and M. R. Salvador. "Software Defined Home Networking: Research Challenges and Innovation Opportunities."(invited paper), In International Workshop on Telecommunications (IWT'13), Santa Rita do Sapucaí, Brazil, 6-9 May 2013
- Rodrigo R. Denicol, Eder L. Fernandes, Christian E. Rothenberg, Zoltán Lajos Kis, "On IPv6 support in OpenFlow via Flexible Match Structures". OFELIA/CHANGE Summer School SummerSchool, Poster session, Berlin, Germany, November 7-11 November 2011.

ANNEX C – Full Ryu Certification test results

Ryu Certification Resume		
ofsoftswitch13	OK	ERROR
Action	50	6
(Required)	(3)	(0)
(Optional)	(47)	(6)
set_field	159	7
(Optional)	(159)	(7)
Match	708	6
(Required)	(108)	(0)
(Optional)	(600)	(6)
Group	15	0
(Required)	(3)	(0)
(Optional)	(12)	(0)
Meter	30	6
(Optional)	(30)	(6)
Total	962	25
(Required)	(114)	(0)
(Optional)	(848)	(25)

C.1 Action Tests

Action	Required	IPv4	IPv6	ARP
OUTPUT	x	OK	OK	OK
PUSH_VLAN	-	OK	OK	OK
PUSH_MPLS	-	OK	OK	OK
PUSH_PBB	-	OK	OK	OK
PUSH_VLAN (multiple)	-	ERROR	ERROR	ERROR
POP_VLAN	-	OK	OK	OK
COPY_TTL_OUT	-	OK	OK	[(#069a36adbdd0739563365540be6e9b28)]
COPY_TTL_IN	-	OK	OK	[(#4f5d77f1fc49b1b854e116048c24058d)]
SET_MPLS_TTL	-	OK	OK	OK
DEC_MPLS_TTL	-	OK	OK	OK
PUSH_MPLS (multiple)	-	OK	OK	OK
POP_MPLS	-	OK	OK	OK
PUSH_PBB (multiple)	-	OK	OK	OK
POP_PBB	-	ERROR	ERROR	ERROR

Decrease TTL	Required	ether	vlan	mpls	pbb
SET_NW_TTL (IPv4)	-	OK	OK	OK	OK
DEC_NW_TTL (IPv4)	-	OK	OK	OK	OK
SET_NW_TTL (IPv6)	-	OK	OK	OK	OK
DEC_NW_TTL (IPv6)	-	OK	OK	OK	OK

set_field	Required	IPv4	IPv6	ARP
ETH_DST	-	OK	OK	OK
ETH_SRC	-	OK	OK	OK
ETH_TYPE	-	OK	OK	OK
TUNNEL_ID	-	OK	OK	OK
VLAN_VID	-	OK	OK	OK
VLAN_PCP	-	OK	OK	OK
MPLS_LABEL	-	OK	OK	OK
MPLS_TC	-	OK	OK	OK
MPLS_BOS	-	OK	OK	OK
PBB_ISID	-	ERROR	ERROR	ERROR

set_field	Required	ether	vlan	mpls	pbb
IP_DSCP (IPv4)	-	OK	OK	OK	OK
IP_ECN (IPv4)	-	OK	OK	OK	OK
IP_PROTO (IPv4)	-	ERROR	ERROR	ERROR	ERROR
IPV4_SRC	-	OK	OK	OK	OK
IPV4_DST	-	OK	OK	OK	OK
TCP_SRC (IPv4)	-	OK	OK	OK	OK
TCP_DST (IPv4)	-	OK	OK	OK	OK
UDP_SRC (IPv4)	-	OK	OK	OK	OK
UDP_DST (IPv4)	-	OK	OK	OK	OK
SCTP_SRC (IPv4)	-	OK	OK	OK	OK
SCTP_DST (IPv4)	-	OK	OK	OK	OK
ICMPV4_TYPE	-	OK	OK	OK	OK
ICMPV4_CODE	-	OK	OK	OK	OK
IP_DSCP (IPv6)	-	OK	OK	OK	OK
IP_ECN (IPv6)	-	OK	OK	OK	OK
TCP_SRC (IPv6)	-	OK	OK	OK	OK
TCP_DST (IPv6)	-	OK	OK	OK	OK
UDP_SRC (IPv6)	-	OK	OK	OK	OK
UDP_DST (IPv6)	-	OK	OK	OK	OK
SCTP_SRC (IPv6)	-	OK	OK	OK	OK
SCTP_DST (IPv6)	-	OK	OK	OK	OK
IPV6_SRC	-	OK	OK	OK	OK
IPV6_DST	-	OK	OK	OK	OK
IPV6_FLABEL	-	OK	OK	OK	OK
ICMPV6_TYPE	-	OK	OK	OK	OK
ICMPV6_CODE	-	OK	OK	OK	OK
IPV6_ND_TARGET	-	OK	OK	OK	OK
IPV6_ND_SLL	-	OK	OK	OK	OK
IPV6_ND_TLL	-	OK	OK	OK	OK
ARP_OP	-	OK	OK	OK	OK
ARP_SPA	-	OK	OK	OK	OK
ARP_TPA	-	OK	OK	OK	OK
ARP_SHA	-	OK	OK	OK	OK
ARP_THA	-	OK	OK	OK	OK

C.2 Match Tests

Match	Req	ether	vlan	mpls	pbb
IP_DSCP (IPv4)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IP_ECN (IPv4)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IP_PROTO (IPv4)	x	OK/OK /OK	OK/OK /OK	OK / OK / OK	OK/OK /OK
IPV4_SRC	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV4_SRC (Mask)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV4_DST	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV4_DST (Mask)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
TCP_SRC (IPv4)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
TCP_DST (IPv4)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
UDP_SRC (IPv4)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
UDP_DST (IPv4)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
SCTP_SRC (IPv4)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
SCTP_DST (IPv4)	-	OK / OK / OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ICMPV4_TYPE	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ICMPV4_CODE	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IP_DSCP (IPv6)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IP_ECN (IPv6)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IP_PROTO (IPv6)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
TCP_SRC (IPv6)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
TCP_DST (IPv6)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
UDP_SRC (IPv6)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
UDP_DST (IPv6)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
SCTP_SRC (IPv6)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
SCTP_DST (IPv6)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_SRC	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_SRC (Mask)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_DST	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_DST (Mask)	x	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_FLABEL	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_FLABEL(Mask)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ICMPV6_TYPE	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ICMPV6_CODE	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_ND_TARGET	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_ND_SLL	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_ND_TLL	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_EXTHDR	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
IPV6_EXTHDR(Mask)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK

Match	Req	ether	vlan	mpls	pbb
ARP_OP	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ARP_SPA	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ARP_SPA (Mask)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ARP_TPA	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ARP_TPA (Mask)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ARP_SHA	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK / OK / OK
ARP_SHA (Mask)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ARP_THA	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK
ARP_THA (Mask)	-	OK/OK /OK	OK/OK /OK	OK/OK /OK	OK/OK /OK

Match	Required	IPv4	IPv6	ARP
IN_PORT	x	OK / OK / OK	OK / OK / OK	OK / OK / OK
METADATA	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
METADATA (Mask)	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
ETH_DST	x	OK / OK / OK	OK / OK / OK	OK / OK / OK
ETH_DST (Mask)	x	OK / OK / OK	OK / OK / OK	OK / OK / OK
ETH_SRC	x	OK / OK / OK	OK / OK / OK	OK / OK / OK
ETH_SRC (Mask)	x	OK / OK / OK	OK / OK / OK	OK / OK / OK
ETH_TYPE	x	OK / OK / OK	OK / OK / OK	OK / OK / OK
TUNNEL_ID	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
TUNNEL_ID (Mask)	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
VLAN_VID	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
VLAN_VID (Mask)	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
VLAN_PCP	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
MPLS_LABEL	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
MPLS_TC	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
MPLS_BOS	-	OK / OK / OK	OK / OK / OK	OK / OK / OK
PBB_ISID	-	ERROR / OK / OK	ERROR / OK / OK	ERROR / OK / OK
PBB_ISID (Mask)	-	ERROR / OK / OK	ERROR / OK / OK	ERROR / OK / OK

C.3 Group Tests

Group	Required	IPv4	IPv6	ARP
ALL	x	OK	OK	OK
SELECT_Ether	-	OK	OK	OK
SELECT_IP	-	OK	OK	OK
SELECT_Weight_Ether	-	OK	OK	OK
SELECT_Weight_IP	-	OK	OK	OK

C.4 Meter Tests

Meter	Required	IPv4	IPv6	ARP
DROP_00_KBPS_00_1M	-	OK	OK	OK
DROP_00_KBPS_01_10M	-	OK	OK	OK
DROP_00_KBPS_02_100M	-	OK	OK	OK
DROP_01_PKTPS_00_100	-	OK	OK	OK
DROP_01_PKTPS_01_1000	-	OK	OK	OK
DROP_01_PKTPS_02_10000	-	OK	OK	OK
DSCP_REMARK_00_KBPS_00_1M	-	OK	OK	OK
DSCP_REMARK_00_KBPS_01_10M	-	OK	OK	OK
DSCP_REMARK_00_KBPS_02_100M	-	ERROR	ERROR	ERROR
DSCP_REMARK_01_PKTPS_00_100	-	OK	OK	OK
DSCP_REMARK_01_PKTPS_01_1000	-	OK	OK	OK
DSCP_REMARK_01_PKTPS_02_10000	-	ERROR	ERROR	ERROR