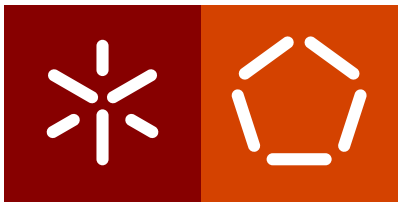
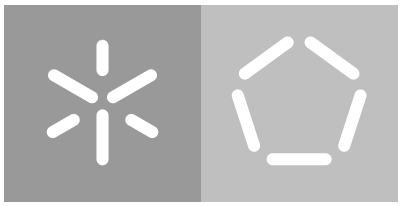


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Catarina Isabel Pires da Silva

Using Software Defined Networking for Flexible Network Measurements

March 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Catarina Isabel Pires da Silva

Using Software Defined Networking for Flexible Network Measurements

Master dissertation
Master Degree in Informatics Engineering

Dissertation supervised by
Maria Solange Pires Ferreira Rito Lima
João Marco Cardoso da Silva

March 2017

Universidade do Minho

Declaração RepositóriUM: Dissertação de Mestrado

Nome: Catarina Isabel Pires da Silva

Nº Cartão Cidadão /BI: 13491302 Tel./Telem.: 915420361

Correio eletrónico: kat.pps.16@gmail.com

Curso: Mestrado em Engenharia Informática Ano de conclusão da dissertação: 2017

Área de Especialização: Informática

Escola de Engenharia, Departamento/Centro: Departamento de Informática

TÍTULO DISSERTAÇÃO/TRABALHO DE PROJECTO:

Título em PT : Medições Flexíveis de Rede Utilizando Redes Definidas por Software

Título em EN : Using Software Defined Networking for Flexible Network Measurements

Orientadores: Maria Solange Pires Ferreira Rito Lima , João Marco Cardoso da Silva

Declaro sob compromisso de honra que a dissertação/trabalho de projeto agora entregue corresponde à que foi aprovada pelo júri constituído pela Universidade do Minho.

Declaro que concedo à Universidade do Minho e aos seus agentes uma licença não-exclusiva para arquivar e tornar acessível, nomeadamente através do seu repositório institucional, nas condições abaixo indicadas, a minha dissertação/trabalho de projeto, em suporte digital.

Concordo que a minha dissertação/trabalho de projeto seja colocada no repositório da Universidade do Minho com o seguinte estatuto (assinale um):

1. ☐ Disponibilização imediata do trabalho para acesso universal;
2. ☒ Disponibilização do trabalho para acesso exclusivo na Universidade do Minho durante o período de
   ☐ 1 ano, ☐ 2 anos ou ☒ 3 anos, sendo que após o tempo assinalado autorizo o acesso universal.
3. ☐ Disponibilização do trabalho de acordo com o **Despacho RT-98/2010 c)** (embargo___ anos)

Braga/Guimarães, 30 /01 /2017

Assinatura: Catarina Isabel Pires da Silva

ACKNOWLEDGEMENTS

Para a minha mãe, Isabel, que me apoiou incondicionalmente e acreditou sempre em mim. Obrigada mãe.

Quero também agradecer à minha tia Ana e à minha avó Belmira, pelo encorajamento que sempre me deram ao longo da minha vida e por estarem sempre presentes, inclusive nos momentos difíceis. Deixo também uma palavra de carinho ao meu irmão João, pela ajuda e amizade.

Quero deixar um agradecimento especial aos meus orientadores, Professora Solange e Professor João Marco, pela supervisão, disponibilidade, confiança e transmissão de conhecimentos, essenciais para a elaboração e conclusão deste trabalho.

Ao meu pai, Mário, que ao longo da vida me foi transmitindo as suas paixões, que são agora também minhas, e quero agradecer o apoio quem me tem dado para seguir os meus sonhos.

Gostaria também de mencionar os meus amigos de Braga e de Vila-Real: Rafaela, Sérgio Ricardo, Ana, Lucas e Nuno. Obrigada pelo vosso apoio, carinho e amizade ao longo de todos estes anos. À Raquel, pelo apoio em todos os momentos, força e carinho, principalmente na reta final.

Não posso também deixar de agradecer à minha amiga canina Tita, pelo carinho e atenção diários.

Finalmente, gostaria de agradecer os meus avôs João e Arlindo e a minha avó Irene, pelo amor e carinho que me deram, e valores que me incutiram. Apesar de já não estarem fisicamente presentes, estão presentes em tudo o que faço.

ABSTRACT

Network management evolved in a way where implementing complex, high level network policies, implies dealing with some attributes that depend on low-level specific configuration. This reflects on a difficulty of changing the underlying infrastructure. SDN (Software-Defined Networking) concept opens a road for new developments due to the centralized non vendor-specific control of the network, most of it related with the separation of data and control planes. Since collecting actual data to create information is important at the time of taking decisions, network operators need to understand the dynamic of their network through monitoring and sampling. An SDN approach offers different possibilities to solve network managing problems, raising new points of view on how networks can operate and, consequently, how they can be managed and monitored. This study is mainly focused on exploring the SDN architecture, and its elements, for applying sampling techniques through flexible network measurements. To pursue this, SDN elements will be presented and explained, alongside with existing monitoring solutions. These solutions, after explored and analysed, will lead to a new approach on applying and configuring flexible sampling techniques on SDN.

RESUMO

A gestão de redes evoluiu de forma a que, para implementar políticas de rede de alto nível complexas, é comum lidar com alguns atributos que dependem de configuração específica de baixo nível. Isto reflete-se numa dificuldade: mudar a infra-estrutura subjacente. O conceito SDN (Software-Defined Networking) abre caminho para novos desenvolvimentos devido ao controlo de rede centralizado que não depende do fornecedor de equipamentos de rede, em grande parte devido à separação das camadas de dados e controlo. Para que os operadores de rede possam compreender a dinâmica da rede, recorrem às tarefas de monitorização e amostragem, uma vez que a captura de dados reais da rede, com finalidade de criar informação, torna-se importante no momento de tomar decisões. Uma abordagem SDN oferece diversas possibilidades para resolver problemas de rede, apresentando novos pontos de vista sobre como as redes podem operar e, consequentemente, como podem ser geridas e monitorizadas. Este estudo está principalmente focado na exploração da arquitetura SDN, e os seus elementos, para a aplicação de técnicas de amostragem através de medições de rede flexíveis. Para alcançar o objetivo final, os elementos de uma SDN serão apresentados e explicados, juntamente com as soluções de monitorização existentes. Essas soluções, depois de exploradas e analisadas, irão guiar o trabalho para uma nova abordagem na aplicação de técnicas de amostragem flexível em SDN.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface.
ARP	Address Resolution Protocol.
CE	Control Element.
CPqD	Centro de Pesquisa e Desenvolvimento em Telecomunicações.
CRUD	Create, Read, Update and Delete.
DPCTL	Datapath Control.
DSCP	Differentiated Services Code Point.
FE	Forwarding Element.
ForCES	FORwarding & Control Element Separation.
IETF	Internet Engineering Task Force.
IP	Internet Protocol.
MAC	Media Access Control.
MPLS	Multiprotocol Label Switching.
NaaS	Network-as-a-Service.
NE	Network Element.
NFV	Network Function Virtualization.
NOS	Network Operating System.
NS-3	Network Simulator - 3.
ONF	Open Networking Foundation.
OXM	OpenFlow Extensible Match.
QoS	Quality of Service.

REST	Representational State Transfer.
SDN	Software-Defined Networking.
SLA	Service Level Agreement.
TE	Traffic Engineering.
TLS	Transport Layer Security.
TLV	Type-length-value.
ToS	Type of service.
VLAN	Virtual Local Area Network.

1

INTRODUCTION

Despite the widespread use of traditional networks, designed to meet the requirements of enterprises, carriers and end users, the underneath structure of networks became less flexible. This occurs mainly because of the integration and interconnection of many proprietary, vertically integrated devices, where vendors dictate specific configurations methods, commands and software [2]. It can be seen from the transition from IP (Internet Protocol) version 4 to IP version 6, that is taking decades to be accomplished, or the introduction of new routing protocols, that can take a decade to become fully operational, that today's networks are rigid and somehow resilient to progress or new solutions. The use of the traditional networking architecture, means that any change will affect the entire network, so we reach a point where networks are relatively static as their operators seek to minimize the risk of disruptions.

Over the years we witnessed the arrival of new trends, such as server virtualization and cloud services, an increasing number of mobile devices and online content, leading the networking industry to deliberate about how traditional network architectures can be adapted or even deciding if a new perspective for it should be taken [3].

The SDN (Software-Defined Networking) architecture proposes to structure the network in three different layers: the infrastructure layer, the control layer and application layer. This organization has the purpose of decoupling the data and control planes allowing some networking tasks such as forwarding and monitoring to be held by a centralized node called *Controller* [4]. The arise of a centralized controller assists decision making for each device since it enables a full view of the network. It is also where the high-level application plane policies are translated to the low-level instructions, providing an explicit control point.

To connect the controller to the infrastructure layer an API (Application Programming Interface) must be selected, being the OpenFlow standard the most popular in the SDN domain. Then, the controller software, called NOS (Network Operating System), runs the data plane protocol so the infrastructure layer and control layer can communicate with each other, enabling networking tasks to be performed [3].

1.1 MOTIVATION AND OBJECTIVES

SDN researching is largely focused on how to apply this novel architecture to fulfil today's needs, how innovation can be lead, its challenges and benefits, and other topics such as scalability, security and forwarding solutions. The topic discussed in this work is related to the use of sampling in the network monitoring context.

Sampling provides an overview of the network dynamics by collecting some specified data (packets) in specific nodes, at specific time or count interval, allowing to retrieve information about what happens in the network [5]. However, it is also important to define how this data will be retrieved. This is known as *sampling technique*. Different sampling techniques are used to estimate different types of patterns or measurements in the network, making it important to choose what sampling technique should be used, considering what information is sought [6].

In the context of sampling-based network monitoring, this research work explores the use of SDN concepts, elements and architecture to sustain the selection and configuration of sampling techniques in the network environment being monitored. The aim is to take advantage of SDN to provide an insight on selecting the most suitable sampling solutions for monitoring an SDN network. Furthermore, this work introduces the elements that coexist in the network, what are the options when building a simple topology and what are the most feasible monitoring approaches, taking into account that today more and various resources are used in a network, meaning more load.

Considering the aspects above and the importance of monitoring and managing networks in a flexible and efficient way, studying a solution on how packet sampling techniques must be approached in an SDN architecture to sustain monitoring operations is the main objective to fulfill in this work. For this purpose, the following list of partial objectives to be met has been defined:

- Identify and present SDN-related elements, such as data and control planes standards;
- Study existing solutions using traffic sampling in the SDN context, their advantages and drawbacks;
- Select a virtual workspace to test SDNs concepts and devised solutions;
- Study and propose an approach that allows the flexible support of sampling techniques in the SDN architecture, without disregarding network efficiency.

1.2 RESEARCH METHODOLOGY

To achieve the main objective of this work, a research methodology is introduced, allowing to go further on the objectives itemized above. The study of the state of the art covering

the main aspects of an SDN architecture, an explaining SDN elements and their use, make the first step of the methodology. Secondly, it is provided a description of solutions for network sampling, together with an overview of data plane standards, and what is available to perform monitoring and sampling on the SDN context. To conclude, an analysis of the best solution for the defined problem is presented and justified.

1.3 DOCUMENT LAYOUT

From this point, the document is organized as follows: the introduction of the motivation to use SDN, its concepts and architecture are presented in Chapter 2. This chapter also presents the OpenFlow standard as communication protocol in the lower layers of SDN together with an overview of this standard's versions. Network operating system as the software that operates in the controller together with scalability and security issues that exist in SDN are discussed in Chapter 2. Monitoring and sampling concepts along with OpenFlow-based tools are introduced in Chapter 3. In Chapter 4 the main elements, strategy and proposed solution for sampling-based network monitoring in SDNs are explained. Conclusions and prospect for future work are presented in Chapter 5.

2

STATE OF THE ART

In this chapter, general concepts of what can be called as the traditional network architecture are faced with what the SDN concept brings to the networking field. Its architecture is confronted with the networking system that is mostly used. The separation of the data and control planes is prominent, since they are usually working together in a single unit, such as a router or switch. To have both planes separated working together, results in the need for specific software. Openflow is a data plane protocol that provides this communication between planes being widely adopted by the SDN community for such purpose.

On the other hand, the diversity of NOSs available requires a selection based on the analysis of functionality and what is expected for it to do. Hence, an emulating tool to provide an environment for implementation and tests is required, justifying the explanatory section on available emulation tools included in this chapter. Furthermore, some scalability and security issues are addressed.

2.1 GENERAL CONCEPTS

Today's networks are mostly build by layers of switches and routers arranged in an organized graph structure. This leads to a point where it is difficult to take further advantage of this architecture. This is mostly because of today's networks dynamic, resulting on the changing traffic patterns, where applications no longer rely on an exclusive client-server communication, but also on having permanent access from any device to different databases and servers. The continuing growth of cloud services usage, being them private or public, leads to growing requirements for security, compliance and auditing that, together with other key factors, motivate the need for a new network architecture [3].

The SDN architecture proposes a change to increase flexibility and fulfil the demands of its users. In Figure 1, the decoupling of data and control planes is faced with the traditional architecture scheme. As illustrated, SDN provides a centralized point of control that can directly influence multiple processes of a network element using freely programmable control software. This means that we are no longer relying on proprietary management systems.

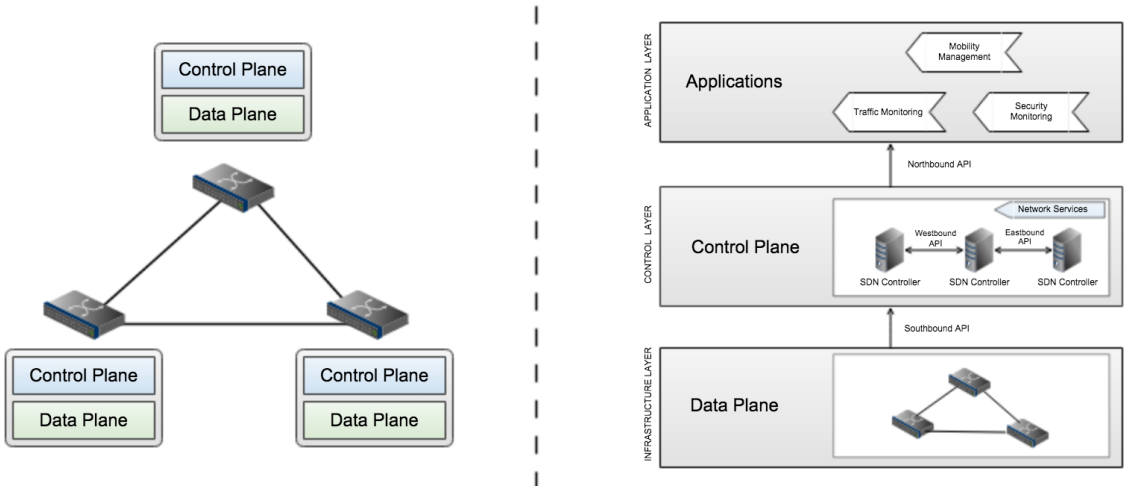


Figure 1.: Traditional Architecture vs SDN Architecture

Due to the separation of planes, the neutral software and the emerging of open and free software to control and operate networks, SDN permits the introduction of new features to be more easily implemented than in most of today's environments.

Since most of the current networks are dependent on vendor-specific software and hardware, usually taking a lot of time to make different vendor components work together, it did not take long until companies and researchers realized that the SDN architecture could benefit them.

Using SDN only for reaching forwarding goals, being the first direction on SDN usage, was not enough and other abilities were seen, such as implementing policies for network security, monitoring and management. The capability of having a central point of control, accessing and viewing the whole system, while having the possibility of making different kinds of traffic engineering decisions in different regions of the network, provides an increase of flexibility on network management and monitoring [2].

2.2 SDN ARCHITECTURE

The SDN architecture is divided in three different layers, as illustrated in Figure 2.

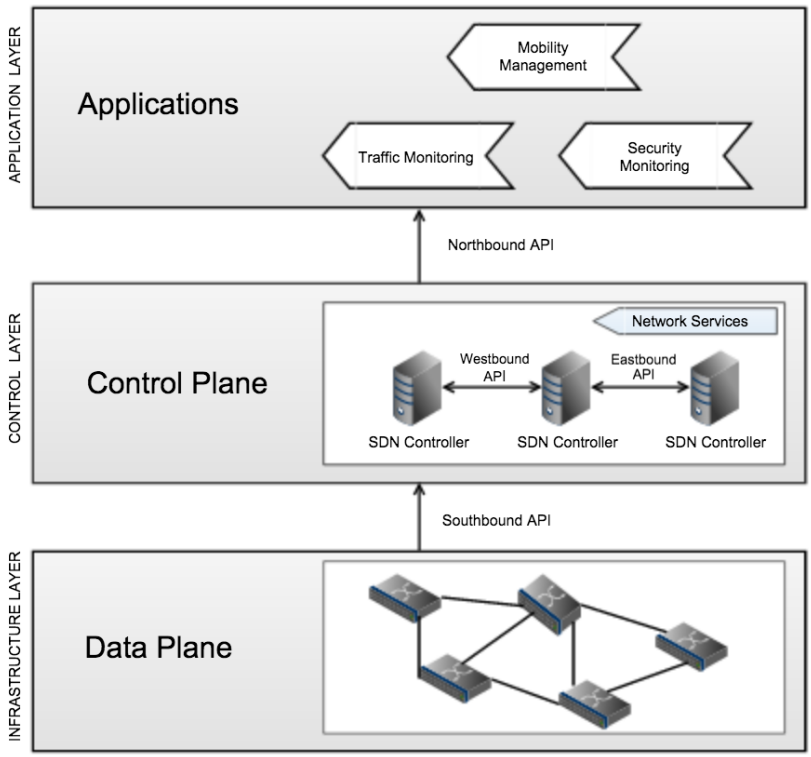


Figure 2.: SDN Architecture - A General Overview

The infrastructure layer is where networks devices such as switches and routers are, forming what is known as *data plane*. The middle layer, called control layer, is made by one or more controllers, which provide several ways of centrally operating the network. The connection between these layers is done via an API, known as southbound API. If it is decided to have several controllers, their connection is done via west and eastbound APIs. On the top we have the application layer that provides, via a northbound API, the possibility of communicating with the control layer, sending specific instructions through functional applications, which may have several purposes such as monitoring and controlling access for operation and management of the network [4].

This separation of data and control planes is important as it becomes easier to address these two different functions and make each of them more flexible and manageable. It also allows data and control planes functions to be physically separated by hardware. Contextually, this means that an SDN switch only has a data plane module and does not have any conventional control plane functionality, fully relying on the external controller entity to make decisions. As SDN provides a more centralized control, network operators only need to manage the controllers, enabling a possible NaaS (network-as-a-service) reality.

The vision of SDN is a key enabler for simplifying management processes leading to keen interest from both the industry and the research community. Exploring the SDN

architecture in network management can solve many problems because, in this way, flexibility, programmability, simplification of tasks and application deployment can be achieved through a centralized network view [7]. Managing a network with SDN means that in a single node of the network, the controller, has the power to configure, collect and store data from numerous points of the network and then analyse them.

The heterogeneous choice of SDN architecture can be observed in Figure 3. As Northbound API, the choice varies from REST(Representational State Transfer) to Procera[8] and Frenetic[9]. A variety of NOSs are available to function as SDN controllers, such as RYU [10], POX [11] and Beacon[12].

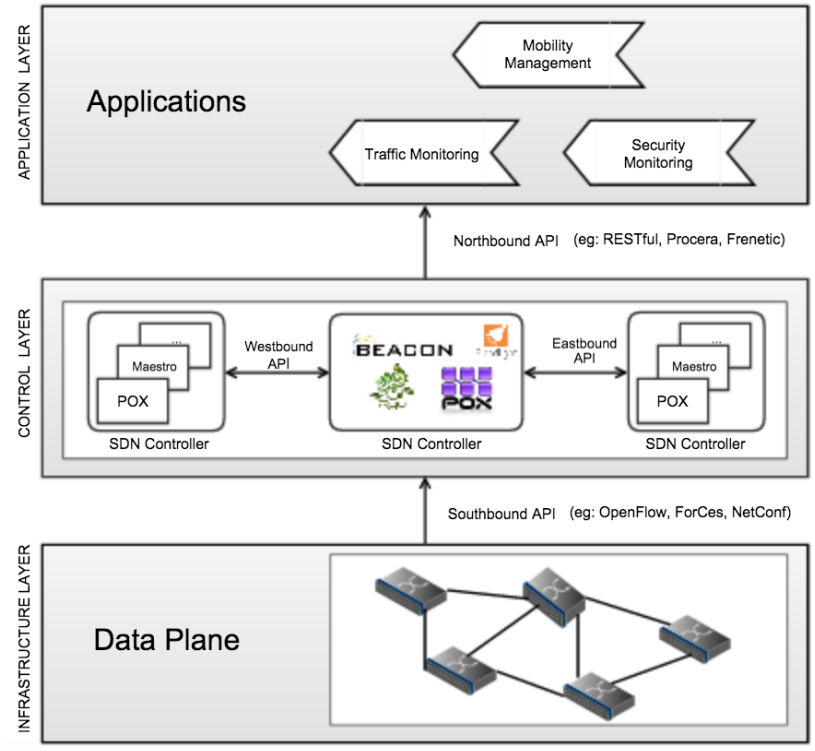


Figure 3.: Detailed SDN Architecture

From a bottom up point of view, we first come across the Southbound API. There are several available interfaces, OpenFlow [13] and ForCES (FORwarding & Control Element Separation) [14] being the ones with greater expression.

SDN philosophy is considered as the key to provide a faster pace of innovation in communication networks as well as more competition on the market, reducing the costs for network operators and providing better solutions on network management [2].

The insertion or replacement of SDN networks in existing network structures is a delicate issue mostly due to the currently large number of installed networks. Moreover, it becomes

difficult to discard these working networks and replace them with a new SDN infrastructure, where all network elements would be enabled or already have support to work with SDN. This results on a need to simultaneous support of SDN and legacy equipment, if an integration or transition from an existing network to an SDN is going to be made [4].

2.3 FORWARDING & CONTROL ELEMENT SEPARATION

ForCES [14] is an IETF (Internet Engineering Task Force) working group established in 2001 which proposes a more flexible approach to traditional network management, providing a clear separation between control and data plane without changing the current architecture of the network. This means that there would not be the need for a logically-centralized external controller since the control and data planes are separated but can be kept in the same network element [14] [15] [16].

The ForCES protocol is composed by FEs (Forwarding Element) and CEs (Control Element), which respectively is where the forwarding plane is instantiated and where the control place can be found. The protocol is also composed by a physical or virtual form that is referred as NE (Network Element), that can be seen as a packet processing entity. Finally, ForCES also includes CE and FE managers that, beyond being accountable for bootstrap and subsidiary mechanisms, are also responsible for discovering CEs and FEs and determine which ones will communicate [17].

The interfaces used in the ForCES architecture (see Figure 4) are defined by the protocol and consist of:

- Fp, Fi and Fr representing the CE-FE, FE-FE and CE-CE, respectively;
- Fc represents the interface between the CE manager and a CE;
- Ff defines the interface between the FE manager and an FE
- The interface between the CE manager and the FE manager is characterized has Fl.

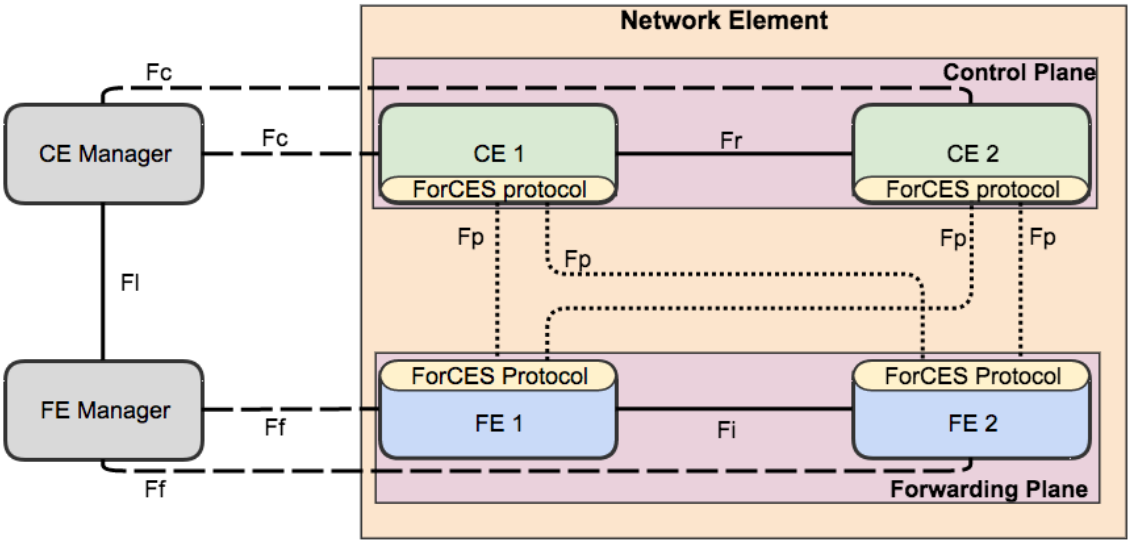


Figure 4.: ForCES Architecture

The literature often relates ForCES has a perfectly usable southbound API for SDN [4] [18] [19]. Other studies refer that, although ForCES was not designed for SDN, it describes a good role model of how an SDN southbound API should be designed [20].

There is also a study to the applicability of using ForCES along SDN-enhanced NFV (Network Function Virtualization) [21]. Other studies present the possibility of combining ForCES and OpenFlow to improve forwarding capabilities of the SDN architecture [22]. In practice, OpenFlow is more widely used when compared to other protocols, in an SDN context, mostly because of its implementation of the SDN philosophy and the help of switch manufacturers that implemented devices ready to support this standard.

2.4 OPENFLOW

OpenFlow is a non-proprietary communication standard, which provides a way to establish connection between the control and infrastructure layers of an SDN architecture. Despite OpenFlow (and SDN) being used by the industry, it was initially deployed in academic campus networks [13]. It is supported by the ONF (Open Networking Foundation), which is responsible for the promotion of SDN and publication of OpenFlow switch specifications. The OpenFlow goal is to provide an interface between the separated control plane and data plane, providing an accurate implementation of the SDN idea [13] [15] [16].

OpenFlow allows connection and operation of data plane, enabling a direct control of the network through setting up packet forwarding rules on network devices, such as switches.

An overview of OpenFlow's scope of activity in the SDN architecture is presented in Figure 5.

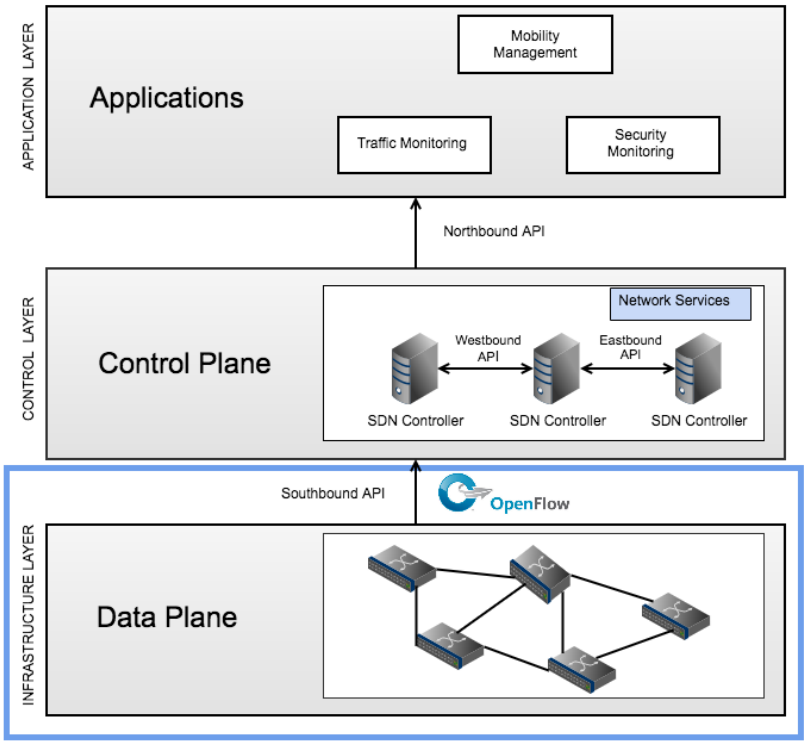


Figure 5.: OpenFlow-based SDN

Instructions or primitives provided by OpenFlow specifications can be used by software applications to apply rules on the SDN infrastructure layer devices. Its implementation is done on both the interfaces: at the infrastructure layer and at the control layer [13] [23].

OpenFlow specifies network traffic based on what are called flows (packets that match the same entry in a flow table). These flows together with a set of headers, are combined with a set of admissible fields on the flow table. Flows can be programmed by the control layer in a static or dynamic way.

SDN architecture working together with OpenFlow, in physical or virtual networks, gives the ability to instantly respond to changes due to the granular control provided mainly by OpenFlow ability of per-flow programmability [13] [24].

The deployment of OpenFlow-based SDN on existing networks can be easy since several network devices support the OpenFlow standard simultaneously with traditional forwarding (called OpenFlow-hybrid switch). This allows to progressively introduce OpenFlow-based SDN technologies, even in multi-vendor network environments. Network devices that only support OpenFlow forwarding rules are called OpenFlow-only switches [23].

The goal of separated control and data plane can be seen in both ForCES and OpenFlow but, regarding the form and architecture perspective, OpenFlow differs from ForCES. The latter resides in the same network architecture with a new architecture of network devices, while OpenFlow implies changing the architecture to the three-layer SDN [13] [14] [16]. This means ForCES does not have the same issues as when using Openflow to provide the decoupling of data and control planes since it does not require a change to the current network architecture from the traditional network.

In this project OpenFlow will be used as southbound API [13].

2.4.1 *OpenFlow Specification*

The OpenFlow Switch specification is a document that describes the OpenFlow protocol. The classical OpenFlow architecture consists of an OpenFlow- compliant switch, a secure channel and a controller as represented in Figure 6 [16] [18].

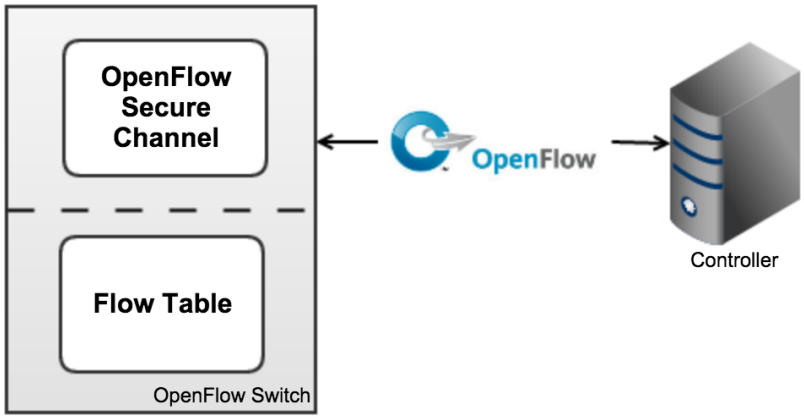


Figure 6.: Representation of the Main Components of an OpenFlow 1.0 Switch

Switches can apply rules from the OpenFlow protocol using flow tables, which are manipulated by a controller, via the OpenFlow protocol. The communication between controller and switch is done through a secure channel interface, allowing several kinds of interactions such as sending instructions from the controller to the switch or replying to a statistics request to the controller.

A flow table includes a list of flow entries used to forward packets. Each of these entries has header fields to match against incoming packets. Counters are updated for a matching packet (used for flow statistics), and actions are applied to matching packets. Actions are instructions for packet matching that allow discard, modify, queue, or forward operations to the packet.

After incoming packets are compared with the match fields of each entry, and if there is a match, the packet is processed according to the action contained in that entry. If not the packet can be encapsulated and sent to the controller. OpenFlow version 1.0 was the first version with official vendor support. The last released OpenFlow specification is 1.5.1, dating March 2015, however, despite being the most recent, older versions such as 1.0 and 1.3 are widely used. The following section describes the main OpenFlow specifications and their major changes [18].

2.4.2 *OpenFlow Versions Overview*

In this subsection, we present an overview of the most important changes from the previous OpenFlow version to the next. The main reason to do such overview is to provide a perspective of how this protocol evolved to fulfil networking needs.

OpenFlow 1.0 [23]

Release Date: December 2009
Protocol version: 0x01

Notes:

- An OpenFlow switch contains a flow table with 12 header fields included in the header and payload of the incoming packets;
- A field in the flow table can have the value represented by the key "ANY" and it will match all packets;
- Flow entries in the table are organized in descending order of priority;
- The first flow entry on the flow table is where the lookup of the packet header starts;
- When a match is found, the actions in the matched flow entry are performed on the packet, otherwise, the packet is sent to the controller via a secure channel for processing;
- Supports multiple queues per output port;
- An opaque identifier, called cookie, was included in flows and it is returned to the controller as part of each flow stats and expired messages;

- Added the possibility to match IP fields inside ARP (Address Resolution Protocol) packets;
- Support for IP ToS(Type of service)/DSCP(Differentiated Services Code Point) bits matching;
- New field for querying port stats for individual ports;
- Improved flow duration resolution in stats/expiry messages by expressing it in nanoseconds resolution.

OpenFlow 1.1 [25]

Release date: February 2011
Protocol version: 0x02

Notes:

- Openflow 1.1 introduces multiple flow tables and a group table composed by group buckets;
- Changes in match fields;
- MPLS (Multiprotocol Label Switching) fields are used to support MPLS tagging and better VLAN (Virtual Local Area Network) support.
- Support for virtual ports on OpenFlow switch;
- Controller connection has new features to deal with failures;
- New protocol object instructions where actions are encapsulated.

OpenFlow 1.2 [26]

Release date: December 2011
Protocol version: 0x03

Notes:

- IPv6 addressing support is added, such as match and header rewrite;

- A more distributed solution is provided since the switch can now connect to multiple controllers concurrently due to controller role change mechanism;
- Introduction of OXM (OpenFlow Extensible Match) to support a variety of header fields.

OpenFlow 1.3 [27]

Release date: April 2012
Protocol version: 0x04

Notes:

- It is possible to control the rate of packets through per flow meters.
- Included a more flexible framework to express capabilities mainly by improving description of tables.
- More flexible table miss support through the use of instructions and actions;
- The existence of IPv6 extension headers can now be matched;
- Packet-rate can be controlled and measured due to the addition of per-flow meters that can be attached to flow entries;
- A controller can configure an event filter on its connection to the switch through a new set of messages;
- Auxiliary connections can be created between the switch and the controller;
- Packet-in messages have now a cookie field representing the value from the flow that sends the packet to the controller;
- Most statistics include a duration field to add accuracy on statistics.

OpenFlow 1.4 [28]

Release date: August 2013
Protocol version: 0x05

Notes:

- More extensible wire protocol by adding TLV (Type-length-value) structures;
- Now the controller can have a better view why packet-in messages were sent because of the introduction of more descriptive reasons;
- New set of optical ports properties that can be used either on Ethernet optical ports or optical ports on circuit switches;
- A flow monitoring framework was introduced to provide the definition of a number of monitors by a controller;
- Improvement of role status events when defining the master controller in a architecture with a master-slave controller environment;
- The controller has access to real time changes done by other controllers to the group table and meter table because of a change in the group and meter notifications.

OpenFlow 1.5 [29]

Release date: December 2014
Protocol version: 0x06

Notes:

- Introduction of *Egress Tables* who provide processing in the output port;
- Now statistics can be automatically sent to the controller based on thresholds;
- Connection status from the switch to a controller can be collected.

2.5 OPENFLOW SWITCH IMPLEMENTATION

To use the OpenFlow protocol in a virtual environment, an OpenFlow software switch implementation is mandatory. All the OpenFlow switch implementations discussed here support the DPCTL (Datapath Control) utility. The DPCTL provides datapath control over the switch and allows several operations, such as querying status and adding or changing flows to the flow table. This is of major interest since it avoids changes to the controller software code when changes in the switch software code are made.

The *OpenFlow reference switch* [30] contain files with initial code from the Stanford University OpenFlow development team, who created it. It includes:

- OFdatapath - implements the flow table in user space;
- OFprotocol - implements the secure channel component of the reference switch.

Another OpenFlow switch implementation is LINC [31], an OpenFlow version 1.2/1.3.1 software switch created by the Infoblox and FlowForwarding group. It is written in Erlang and implemented on the operating system's userspace as a node. It provides flexibility and fast development for testing new OpenFlow features efficiently when considering computational costs.

2.5.1 *Open vSwitch*

Open vSwitch [32] is a virtual switch consisting on a software layer that resides in a virtual machine host [33]. It was designed to bring flexibility and platform-free usage, meeting the needs of the open source community. Similarly to the *OpenFlow reference switch*, Open vSwitch also contain files with initial code from the Stanford University OpenFlow development team.

In the beginning, Open vSwitch had the goal to provide features for applications such as network virtualization. This vision was easily discarded by the team in charge, when they discovered that the key to success relies, not only in high programmability, but also in speed [34].

For the past several years, the focus in its development was to achieve a high level of performance in different platforms while sharing resources and workloads. To prevent problems such as consumption of hypervisor resources, Open vSwitch implements what is called flow caching [34]. Flow caching means that traffic handling is cached on the kernel module the first time a packet from a flow not handled previously, arrives at the switch. This occurs so subsequent packets that match the same flow entry do not have to be handled by the userspace module again.

Today, Open vSwitch is very popular mainly due to its integration with OpenStack Networking service and it is also accepted as the genuine standard OpenFlow implementation.

Figure 7 offers an overview of the Open vSwitch architecture and its main components.

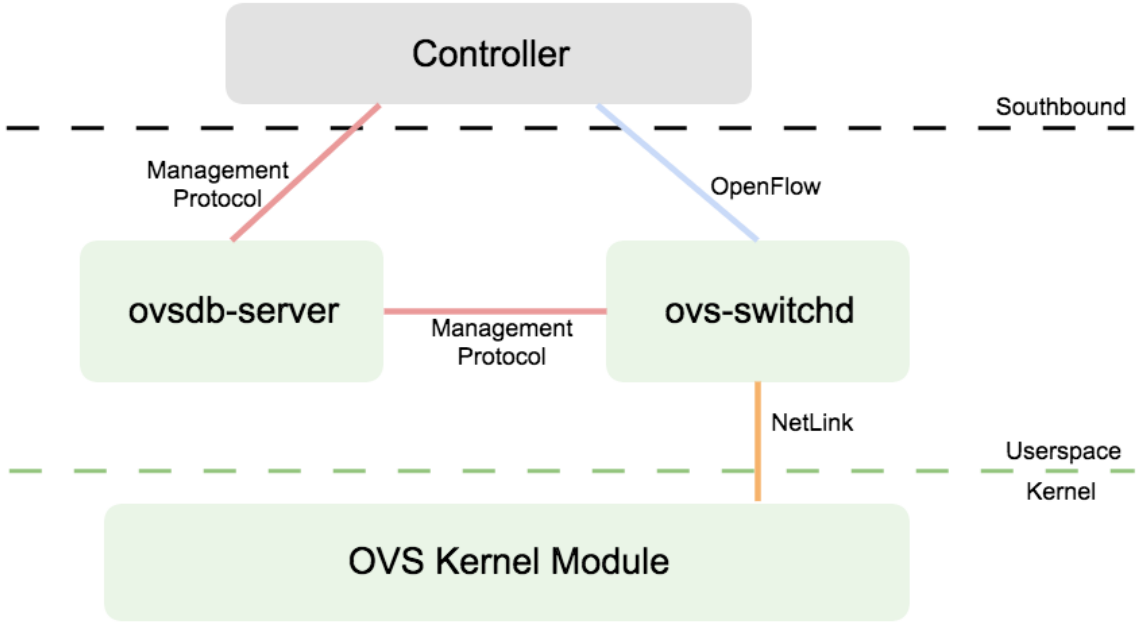


Figure 7.: Open vSwitch Architecture

The Open vSwitch kernel module uses Netlink message framing format through its AF_NETLINK sockets to access the ovs-switchd daemon which implements and manages all the Open vSwitch devices. The OpenFlow protocol is used to exchange messages between the Controller and ovs-switchd.

The datapath (ovs kernel module) uses Netlink socket to interact with ovs-switchd daemon that implements and manages any number of ovs switches on local system, and the SDN controller interacts with ovs-switchd using OpenFlow protocol. The ovsdb-server maintains the switch table database (persistent) and external clients can talk to ovdb-sver using JSON notation.

2.5.2 *ofsoftswitch13*

The ofsoftswitch13 [35] is a userspace software switch implementation of an OpenFlow version 1.3 switch, targeted primarily for research and switch customization. This project is available at GitHub and its code is a changed Ericsson TrafficLab 1.1 softswitch implementation to support OpenFlow 1.3 on the data plane. Formerly maintained by CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações) in technical collaboration with Ericsson Research, it is currently supported by Ericsson Innovation Center in Brazil [35]. Its support is limited, so any code bug or error is solved on a best-effort base. Most of the complications and problems while using this switch implementation seem to be related with the

Linux version and distribution used [36]. When compared to Open vSwitch, ofsoftswitch13 offers a user-level software switch which only supports version 1.3 of OpenFlow while Open vSwitch offers a production quality software switch that implements the OpenFlow protocol up to version 1.5 (experimental).

An overview of ofsoftswitch13 components and their interaction is represented in Figure 8.

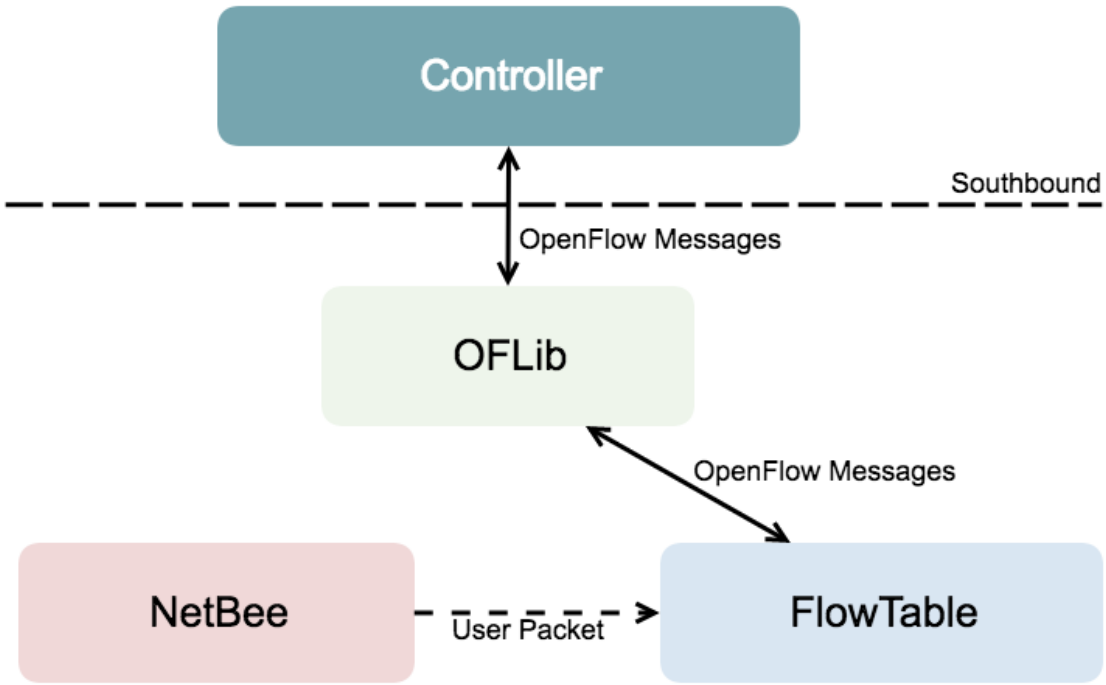


Figure 8.: ofsoftswitch13 Architecture

The ofsoftswitch13 switch provides the OFLib library that converts internal messages to and from OpenFlow 1.3. It also uses the NetBee library [37] for packet decoding [36].

2.6 Network Operating System

The SDN architecture proposes a data plane formed by network devices and a control plane constituted by a centralized controller. Having his centralized controller implies that for devices to be able to apply rules, transmitted to them by the southbound interface, an operational controller must be present. The controller can control, monitor and manage the network and is connected to user applications in the application layer through a northbound interface, thus establishing a connection point between the low and high levels of the SDN network [16].

Using the OpenFlow standard, a manipulation of elements is required such as flow tables or group tables, depending on the version. To manipulate these elements, a NOS, running on the controller is required. This operating system, besides communicating with infrastructure layer devices, should notify the application layer about network events. This means that in a top-down approach in the SDN architecture, the NOS provides abstractions and common APIs to developers [38].

The capabilities of a NOS in providing services such as node discovery, together with other elements, create an environment to speed up the introduction of network applications and protocols [16].

There are several NOS available with different properties. These properties are: support for virtualization, open source, multiplatform support, southbound and northbound API support, programming language, among others [38]. For instance, the RYU [10] and POX [11] use Python as programming language. Floodlight [39] and Beacon [12] use Java. All of them support OpenFlow as the southbound API.

As demonstrated above, the SDN architecture relies partially in the controller and its network operating system as it is the main component of the control plane to generate rules or configurations on the network by the user.

2.6.1 *Network Operating Systems Overview*

After understanding the importance of this element on an OpenFlow architecture, a study of what network operating system could better suit on a simple SDN solution was needed. Some premises were established for this study, so needless complexity is discarded. Only centralized controllers were targeted and support for OpenFlow version 1.3 was preferable but not mandatory [38].

Other properties that NOSs should match:

- They must have virtualization compatibilities (Mininet & Open vSwitch);
- Southbound OpenFlow interface support;
- The controllers must have platform support on Linux;
- Only ”open source” controllers are eligible.

Table 1 lists the NOS that are compatible with the properties defined above.

Table 1.: Network Operating System list

Name	Architecture	Northdbound API	Prog. Language	OpenFlow Support
RYU	centralized multi-threaded	REST API	Python	v1.0|1.2|1.3|1.4|1.5
POX	centralized	ad-hoc API	Python	v1.0
FloodLight	centralized multi-threaded	REST API	Java	v1.0|1.3
Beacon	centralized multi-threaded	ad-hoc API	Java	v1.0

Relevant details of each NOS to guide the decision of the NOS to adopt on this research work are provided below:

RYU[10]:

- Was selected as the best NOS using adapted Analytic Hierarchy Process (a Multi-Criteria Decision Making method) from topmost five controllers (including POX, RYU and FloodLight) [38];
- Any programming language can be used to develop a new component [38];
- REST can be used for both Northbound and Southbound API allowing communication with other systems and browsers.

POX[11]:

- Python 2.7;
- "Pythonic" OpenFlow interface;
- Reusable sample components for path selection, topology discovery, etc.;
- "Runs anywhere" – Can bundle with install-free PyPy runtime for easy deployment or "standard" CPython;
- Supports the same GUI and visualization tools as NOX NOS;
- Last Update around March 2013.

FloodLight[39]:

- FloodLight v1.0 dated Jan2, 2015;
- Full support for OpenFlow v1.0 and 1.3;

- Experimental support for OpenFlow v1.1, 1.2 and 1.4.

Beacon[12]:

- Supports OpenFlow v1.0.

'Ryu' network operating system was considered the most universal controller. It fully supports most of OpenFlow standard's versions and it is very used in the research field.

2.7 Network Virtualization and Mininet

Network Virtualization allowed the networking community to perform a variety of tests without the need of having physical elements at disposal, which is often very difficult to have. This process of turning a real network into a virtual one is the result of network's hardware, software and functionality combination in a single entity called virtual network.

Sometimes the lack of resources is not the only reason to use network virtualization. There are circumstances where software under development is tested in a virtual environment that simulates the one where the software will be working when fully operational. Tests performed on a virtual platform will allow a validation according to the accuracy of the network virtualization, how close it is in emulating real hardware and functionality [40].

Mininet is a network virtualization platform that enables the use and experiment of SDN, Openflow and several NOS. It can run a network application on both small and large networks using lightweight virtualization. Experimenting on this emulator is also a plus since it enables the implementation from a single feature to a brand-new architecture, allowing users to test it in their own topologies and, then, deploy its code on a production network.

Mininet performance increases when it is working on a laptop using Linux, allowing the network to be bundled in a virtual machine where several operations can be done, such as modifying and running the virtual network. Using virtual machines have advantages of being easily modified but in the other hand, scalability issues are, most of the time, present. Mininet's limitations on performance and multi-machine support are directly related to its implementation, but its support on prototyping SDN networks with speed is a notable feature [41].

If, for instance, there is a need for experimenting on wireless networks with the OpenFlow protocol (using Mininet-WiFi, which is a fork of the Mininet SDN network emulator), some of the Mininet's limitations have a harsh effect on the emulation when computational demand is high [42].

Another network simulator for SDN is NS-3 (Network Simulator 3) [43]. Focused on research and educational use, distributed as free software. Simulations modelling OpenFlow switches using ns-3, use an OpenFlow module [44], which, in turn, relies on an external OpenFlow switch library linked to the simulator. This became a main disadvantage since this module implements an OpenFlow version 0.8.9 switch, nowadays considered obsolete. To use a more recent OpenFlow switch, in this case version 1.3, the installation of the OFSwitch13, which uses the ofsoftswitch13 library along with several modifications to work in ns-3, is required [36].

2.8 Scalability and Security Issues

When considering an architecture, such as SDN, that introduces a centralized point where several instructions and decisions are made and transmitted, there are two main issues that cause concern: security and scalability.

Until version 1.2, OpenFlow specifications were directed to a single controller, where the possibility of communication between a switch and several controllers were addressed. Some of the NOS are designed to work as centralized controllers, where multi-threaded environment is explored to achieve highly concurrent systems.

To satisfy the demand of large-scale systems and be able to work on less demanding networks, distributed NOSs are used. There are several ways these controllers can be distributed and it results on different solutions for a set of different scenarios. Some properties that these NOSs offer are consistency and fault tolerance [2] [16].

Security is another important matter to consider. Centralized or distributed, controllers are a point of the network where management, measurement and monitoring are available, meaning that a lot of damage to the network can be done. Therefore, it is considered to be a tempting target of security attacks, where the attacker can directly exploit it. Between controllers the use of security communications protocols, such as TLS (Transport Layer Security) (used in OpenFlow standard), with authentication between nodes can provide an extra protection layer to reduce vulnerability at that point.

The increase of SDN network nodes opens a gap to vulnerability. Security applications for controllers can be used, depending on the software running on the controller. On the other side, we should not forget the capabilities that SDN architecture supports (such as full system view and control from a single node) that provide a powerful feature for reactive security monitoring, analysis and response system, allowing the implementation of security policy techniques such as *Active Security* [4] [16] [45].

It is reckless to think that the security concern is static. New techniques of detection and protection emerge, such as security threats. The solution is to adapt the network security level to the network requirements, where protection is intended [4] [16].

2.9 SUMMARY

This chapter presented an introduction of the SDN concept, the description of this architecture's elements and how they are applied. The decoupling of data and control planes is the basic principle of SDN, which contrast with the traditional network architecture where these are strictly connected.

A protocol called Openflow was devised to work in the data plane following the SDN methodology and applying per-flow rules. An observation of what can be called the metamorphosis of OpenFlow can be analysed through a general description of its versions, helping on having an fundamental understanding about this SDN data plane protocol. Next, the control plane was addressed through the introduction of NOS. A NOS is basically an operating system that works in the control module of an SDN network, and has the main goal of installing rules on the data plane and making the connection between the data and application plane.

An introductory section about the SDN Network Virtualization tool Mininet comes as an essential part since virtualization is today's number one choice for software testing, and Mininet is the number one choice when it is necessary to emulate an SDN network.

This chapter concludes with a brief approach to scalability and security in SDN. Having a centralized point of control is a major concern when scalability is demanded and, at the same moment, given its power to enable direct access to the majority, if not all the network, is a natural target for malicious exploitation.

3

MONITORING USING SOFTWARE DEFINED NETWORKING

Monitoring tasks along with sampling will always be an important part of maintaining a network. This chapter will start by introducing a general view of monitoring and sampling concepts, and then which tools can sustain these tasks in an SDN network that uses OpenFlow as data plane protocol.

3.1 MONITORING AND SAMPLING

Monitoring and sampling are demanding services on a network. Capacity and other properties need to be monitored in order to provide a better view of what happens on the network, so appropriate decisions related to network provision and functionality can be made.

Sampling allows retrieving information about the whole network behaviour without the need of analysing all the data, reducing the impact of monitoring operations in the network.

Sampling techniques are widely used and represent an important step of monitoring since collecting traffic samples allows to get the data by which we retrieve relevant information of the network behaviour.

With the SDN architecture approach of a centralized point of control that simplifies management and manipulation tasks in the network together with OpenFlow providing ways of implementing TE (Traffic Engineering). OpenFlow-based SDN are by excellence, a good way to enhance monitoring and sampling while, at the same time, providing a simplification for the introduction of new network applications. [16].

Network applications targeting monitoring and sampling can either provide new functionalities for distinct networking services or improve features previously provided by OpenFlow-based SDN. These network applications may, not only perform tasks involving network management and traffic engineering, but also tasks related to performance evaluation, network security, SLA (Service Level Agreement) and QoS (Quality of Service) control, being the last two widely done by ISPs [46].

In this work, it is expected to identify techniques of how to implement relevant sampling techniques, in an SDN environment, then increasing and mixing details until a newly

general-purpose sampling-based measurement architecture can be demonstrated and implemented.

For better understanding traffic sampling several concepts should be considered, such as the interval between samples and sample size, described bellow.

Packet sampling is widely used to support network measurements, mostly due to the development of networking infrastructures, high-speed technology and services diversification. Tasks related with sampling-based network measurements include [47]:

- Planning and management of network operation;
- Performance optimization, traffic modelling, characterization and control through traffic engineering;
- Network security;
- Measure and report SLA compliance;
- Control of QoS parameters.

For the understanding of concepts related with packet sampling, the following terminology, illustrated in Figure 9, is used [47]:

- Sample – selected network packets used for network parameters estimation. Can also be referred as an individual action of selecting and capturing packets from the stream;
- Sample size – number of packets selected and captured to constitute a sample. It can also be a time interval. Sample size is controlled by triggers that delimit size by packet position into the stream or timestamp;
- Interval between samples – Number or time interval of ignored packets of a stream. Analogous to the sample size, it is also controlled by triggers.

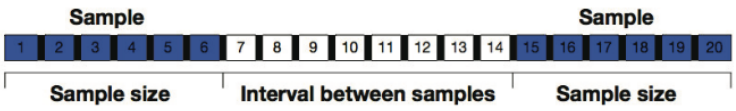


Figure 9.: Sampling Concepts [1]

Sampling techniques can be divided into three components in accordance with granularity, selection scheme and selection trigger in use. Each of these components is divided into a set of approaches. These techniques and their components are described as [47]:

- Granularity - determines the element's atomicity that is under the sampling process analysis: in packet-level, packets are qualified as single independent entities; in flow-level, sampling is applied to packets that belong to a flow or set of flows;
- Selection scheme - represents the function that defines traffic collection and selection. This function can be deterministic, random or adaptive;
- Selection trigger – identifies temporal and spatial sample limits by count, event or time-based approach.

Below, content-independent sampling techniques will be presented. These techniques do not access the packet content for selection and capture decisions. Instead, the sampling triggering process is controlled considering the position or timestamp of the packet on the stream. Systematic Count-based and Systematic Time-based are governed by deterministic functions, while Random n-out-of-N and Random Uniform Probabilistic use non-deterministic functions [1].

Systematic Count-based

The starting point of a sample and sampling size are operated by the spatial packet position (resorting to packet counters) using a deterministic function that results in a periodic behaviour. Figure 10 illustrates the periodic selection of every 5th packet, with sampling size of 1 and the interval between samples equal to 4.

Figure 10.: Systematic Count Based [1]

Systematic Time-based

The systematic time-based sampling technique, similarly to the systematic time-based, also used a deterministic function to rule the sample size and interval between samples. The difference resides in the type of triggers: in this technique, they are oriented by the packet arrival time [1]. Figure 11 illustrates a sample size of 100 milliseconds and an interval between samples of 200 milliseconds.

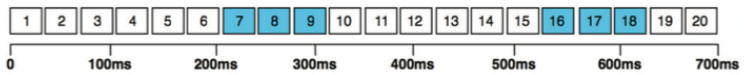


Figure 11.: Systematic Time Based [1]

Random n-out-of-N

The packet selection is ruled by a random process, being the simplest and widely deployed mechanism the capture of n packets from a sequential stream of N packets (n-out-of-N). A pseudorandom function generates n numbers (between [1,N]). Then the packets that have a position equal to one of the random numbers are selected and captured [1]. The probability p (with p = n/N) is applied for all the N packets to be selected and compose the sample. In Figure 12, packet is collected from every five incoming packets.

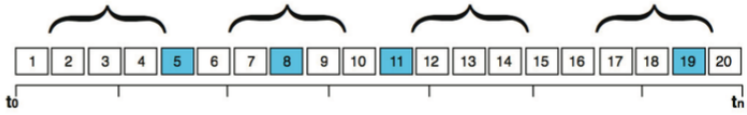


Figure 12.: Random n-out-of-N [1]

Uniform probabilistic

A predefined uniform probabilistic function decides the packet selection to compose a sample, having all packets the same probability of being selected. An example of a random uniform probabilistic technique is a count-driven technique with an independent random variable with distribution of mean 1/p and successive intervals between samples (with sample size equal to 1 packet) [1].

3.2 OPENFLOW-BASED MONITORING SOLUTIONS

3.2.1 *sFlow*

ONF is focused not only on the dissemination and development of the OpenFlow standard on the network industry, as many members of the ONF are major network operators and manufacturers. Some of these members are shared with the sFlow.org industry consortium that has similar objectives for the sFlow standard, making its support available in OpenFlow and non-Openflow switches [48].

sFlow proposes that operations such as monitoring no longer be implemented on the switch, instead sampled packet headers are sent to a separate component of the control plane, called monitor, that gets this packet headers, decodes them and aggregates the data through a traffic analysis application [49].

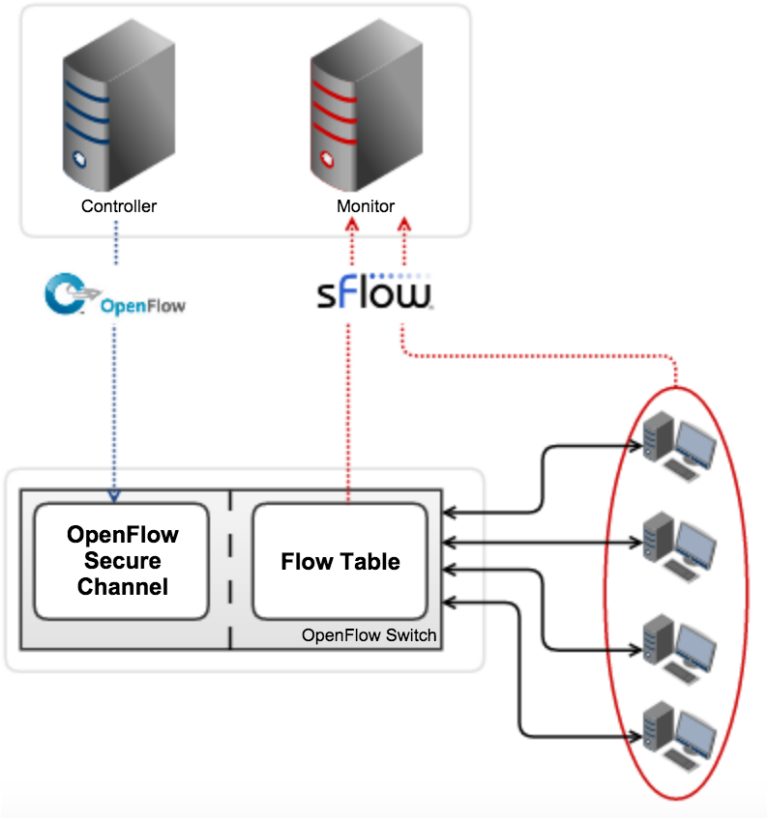


Figure 13.: sFlow OpenFlow-based SDN Architecture

As it can be noticed on Figure 13, sFlow and OpenFlow work in what can be called a partnership. It is intended that the controller, using OpenFlow, configures the forwarding tables in switches and sFlow increases the visibility by providing real-time access into traffic that flows in the network. Having this type of visibility means that the network can adapt to changing demands [50] [48]. This represents the use of sFlow when packet forwarding is controlled by OpenFlow.

One major problem regarding sFlow is that its reports do not include the entire packet, which can be a problem when more packet information is required. In addition, it only provides uniform sampling methods [46].

3.2.2 *FleXam*

FleXam is a per-flow sampling extension for the OpenFlow standard, allowing the controller to access packet-level information. Its priority is to overcome some problems, which may arise in the use of the OpenFlow alone. One of these problems is the increase of flow-entries. From OpenFlow version 1.0 to, for instance, OpenFlow version 1.2, a flow entry went from

a 12-tuple match to OXM based on TLV structures, that allows switches to support a wider range of header fields (for instance, OpenFlow 1.4 supports 41 different types, where TLV was also added to ports, tables and queues) [46] [51] [52].

Packets in FleXam can be sampled stochastically, meaning that a predetermined probability is set, or deterministically, which implies a pattern. This flexibility in sampling is enhanced by the possibility of the controller to define several rules on the packets, such as which should be sampled, what part of it should be selected and where they should be sent [51].

The way FleXam was implemented and how it operates is something to consider. It was implemented as a patch to Open vSwitch and enables the access to packet-level information at the controller where an application should run, allowing the installation of rules, processing sampled packets and collect information.

This sampling extension, in addition to presenting itself with two sampling techniques, is considered flexible for some reasons such as providing a stochastic sampling and a generalized version of the deterministic sampling. The stochastic sampling consists in the selection of packets that are included in a flow, with a probability of p. On the other hand, the generalized version of the deterministic sampling is formulated as selecting m consecutive packets from each k consecutive packets, ignoring the first δ packets.

It was taken in consideration that manipulating full packets was not always necessary and sometimes it could result in excessive load for the network. Therefore, FleXam allows the controller to choose what packet sections should be sent [11]. In Figure 14, an explanation of the action implemented in the Open vSwitch patch, FleXam [10] is provided:

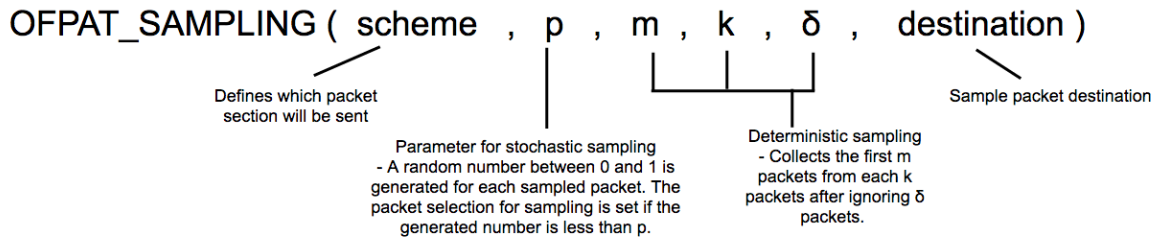


Figure 14.: Implemented Action and its Parameters - FleXam

Analysing FleXam and getting an insight on how it is implemented (Figure 14) allows a simple and effective perspective on how to include a custom sampling mechanism. A switch can be parametrized by the controller unit without compromising substantially the performance or changing the OpenFlow and SDN purpose. The downside of FleXam when compared to sFlow is that it offers a per-flow sampling, having the advantages linked to it, but does not have the possibility of per-packet sampling.

3.3 SUMMARY

Monitoring and sampling allows a knowledge of what is happening in a network without fully analysing it. The advantages of having monitoring enables network operators to detect which and how resources are being used and thenceforth find the most suitable solution to enhance their network properties.

There are some existing solutions for this proposed by SDN, being the most known sFlow, that works through agents collecting data from the elements to be later analysed in a separate unit. Other solution that drew attention was FleXam, since it provides in a single function more than one way of sampling data.

4

PROPOSED SOLUTION

In this chapter all the previously presented elements are taken in consideration and the most suitable solution for implementing sampling-based monitoring in SDN is chosen. As a first approach, using what already exists and make it work such as it is intended is, most of the times, considered and it offers a viable solution. Moreover, a deeper evaluation of how the network elements interact is presented and conclusions are drawn. The elements that will take part of the solution are analysed and an implementation proposal is presented.

4.1 DESIGN GOALS

To propose a solution that better explores and fits the SDN architecture some items were defined as design goals:

- Compatibility with popular software for SDN;
- Efficient and lightweight implementation without compromising the SDN proposal;
- Explore existing solutions of data and control planes and attach monitoring to them without the need of brand new software;
- Open and standard protocols/software sustention.

4.2 FIRST APPROACH

To implement sampling techniques an operational environment is a must. Mininet is the network virtualization tool that will be used since it supports the implementation of OpenFlow-based SDN with the advantage that what will be programed in the controller can be directly deployed in real world. Since the topology that it's going to be used does not consist in many nodes, performance is not an issue to be concerned about [53].

A working topology is a requirement since, with it, it is possible to implement and test solutions.

By default, Mininet provides what they call as "minimal" topology, which is composed by two hosts, connected to an OpenFlow kernel switch, that itself is connected to an OpenFlow reference controller which is set to behave as an Ethernet learning switch. Furthermore, is not ready to be easily modified in some aspects such as the used OpenFlow switch or the controller network operating system.

The OpenFlow version that works in the OpenFlow reference distribution (both the switch and controller) is 1.0. To work with a different OpenFlow version (newer than 1.0), the Open vSwitch virtual switch platform must be used. Regarding the controller NOS, the default topology provides a reference controller that has severally limited functionality.

Under these circumstances, a customized topology, also called parametrized topology, is needed and Mininet provides the right tools through its Python API. This API allows flexibility through the configuration of parameters, such as number of hosts or switches and even performance settings, that after being passed, will lead to the intended configuration. If the intention is to have a topology configuration not so far from the "minimal" topology, some options can be used to change configurations such as number of hosts or switches, so a novel configuration is not required. In this case, with the decided specifications, the Python API was used to configure a topology accordingly.

The topology developed consists of three hosts, each one connected to a switch. Switches are interconnected between themselves and to the controller. The hosts have an unique IP and MAC (Media Access Control) address. All the switches are running Open vSwitch with version 1.3 and have an uniquely assigned MAC address. This topology is ready to connect to a remote controller that is running RYU network operating system.

With the elaboration of this parametrized topology, represented in Figure 15, there is a substantial benefit, which resides in the fact that this code (see Appendix B) can be reutilized, through changing variable's values or deleting code lines, in a few minutes a novel topology with different parameters is build and ready to work. This parametrized topology can be seen as a skeleton to be used as starting point. To install and deploy this environment in a computer, an installation guide is provided in Appendix A.

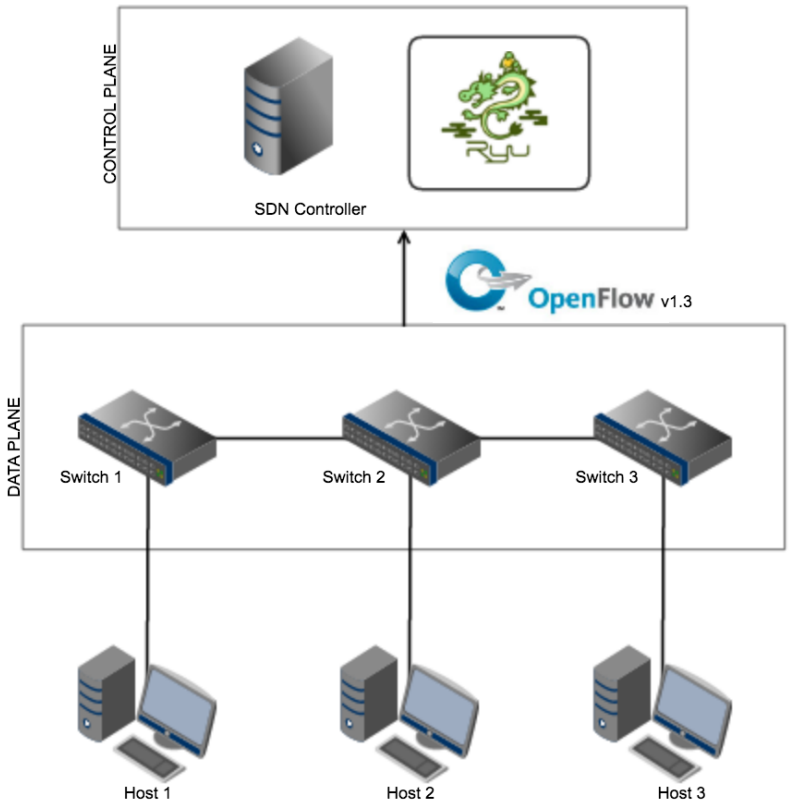


Figure 15.: Implemented Parametrized Topology

4.3 INTERACTION BETWEEN ELEMENTS

The goal is to collect network packets through count or time intervals using the Open-Flow specification provides. When OpenFlow was created, the main goal was to accelerate the innovation on production networks, however, without a mechanism to control bandwidth and delay proved to be a quite difficult task.

The concern of knowing or having control of what happens in the network begun with the first QoS implementation in OpenFlow 0.8.0, but it was not until version 1.3 that OpenFlow substantially increased its QoS framework functionality.

After implementing this kind of mechanism there is still work to do in order to sampling and analysing/characterizing traffic, particularly at packet-level. In this scope, tools such as FleXam are used, as OpenFlow itself has nothing to provide. In this work, it is intended to perform packet-level sampling applying an appropriate solution. Here we divide in two possible approaches the way packet-selection rules can be applied: one from the controller to switch and the other from the switch to the controller. Each strategy/approach is characterized by where and who controls the rules applied for packet selection.

4.3.1 *Controller to Switch*

Here, we assume that it is the controller who is responsible for making sure the sampling intervals are accomplished and the packet information is stored where it should be. This means that the controller is responsible for managing the rules of sampling, with the switch not being aware that a specific selection is being made, because it is the controller who, in some way, forces that. Succinctly, this means that what we have here is the controller selecting, for instance, packet x, y and z, and requesting them, instead of having the switch selecting and sending them. A generic representation is shown in Figure 16.

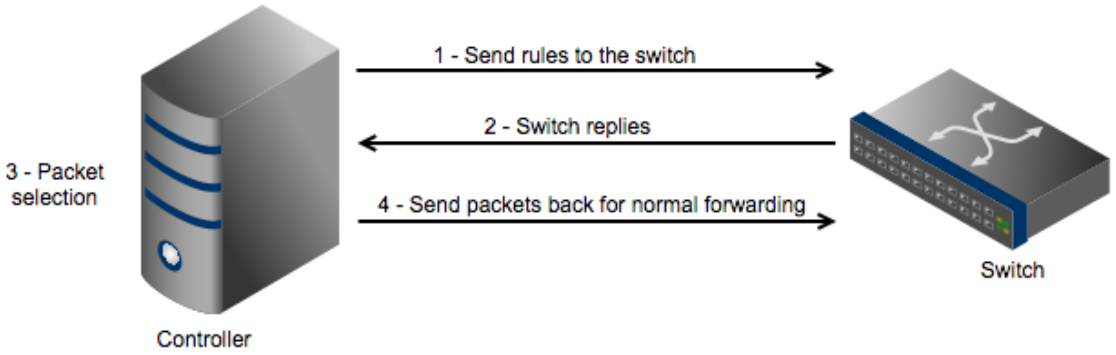


Figure 16.: Controller to Switch - Minimal Approach

In this context, there were two solutions that appeared to be the best path to be taken:

1. Sending all packets, being them original packets, or copies of the packets, matching a flow entry, to the controller [13]. In this solution, every packet is forwarded to the controller, where they are counted. When the counter reaches the intended value, the packet is collected. If the packet is not supposed to be collected, there are two options, depending if the packet sent by the switch is the original or a copy. If it is the original packet, the controller must send a *Packet-Out* message containing the packet, injecting the packet in the data plane. In the other hand, if it is a copy of the packet that is being handled, discarding the packet is the normal procedure.

   Problems identified in this solution:

   - Accumulated traffic in the controller;
   - Possible bottleneck in the controller;
   - Possible delays in the packet forwarding to the switch (this is not a problem if only a copy of the packet is delivered);
   - Possible packet-loss between the controller and switch communications;

- This can be understood as a universal solution since in traditional networks the same result can be obtained adding a monitor to store packets.

2. Request flow statistics from the switch within a pre-set time interval. In case the statistic's packet counter is next to the intended value, change the rule to send the packet to the controller. This solution can fit both time-based and count-based sampling.

   In the OpenFlow specifications, statistics can be demanded from a controller. There are several types of statistics that can be requested such as flow and table statistics. This exchange of information starts by a request message from the controller, mentioning what statistics are wanted, and is followed by a reply from the switch containing the requested information, implying two interactions to get the statistics.

   The idea is having a thread which is launched from the controller, from time to time, making a flow statistics request to the switch. The flow statistics reply from the switch includes information such as the number of packets in flow. This value is the one which the controller should pay attention to as it comes closer to the value pretended. The procedure behaves as follows: a *Flow Modify* message would be sent to the switch, followed by a *Packet-In* message sent to the controller, so that packets, matching that flow, are forwarded to the controller. When the packet arrives, it is collected. After that, another *Flow Modify* message is sent to the switch and, as the packets are forwarded normally, counters are reset.

   Problems identified in this solution:

   - Statistics requests may not match with counter values intended since the communication between requesting statistics and responding to them implies delay.
   - To obtain higher accuracy with this solution, a previous knowledge of the network dynamic is essential and, even though it can be obtainable, it does not guarantee good levels of synchronism.

There could be similar solutions to the ones presented here but all of them rely in the fact that, to have packet selection/collection, several messages between the controller and switch communication are involved, which ultimately leads to lack of performance.

4.3.2 *Switch to Controller*

On the other hand, switch to controller interaction is considered, consisting in applying rules directly on the switch. In this situation, the role of the controller is only to set the parameters of sampling required by the application, sending it to the switch, which in turn, after receive it, will apply it accordingly.

Since it is the switch the one to apply the parameters, this means that switch will be responsible by the selection and collection of packets, and responsible to send them automatically to where the controller ordered. It can be to a monitor or to the controller itself.

To summarize, the controller will only fill a rule with parameters to the switch and the switch will do the whole work and then redirect the outcome. In Figure 17, a graphic scheme of the interaction is represented.

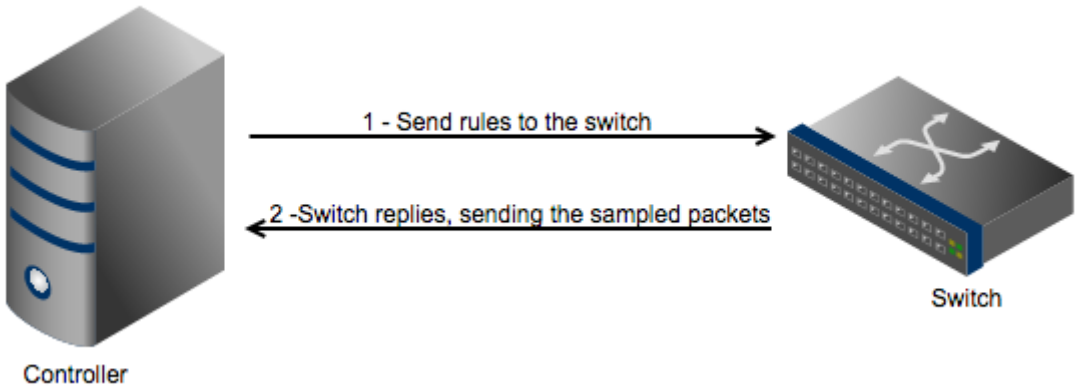


Figure 17.: Switch to Controller - Minimal Approach

After considering the OpenFlow standard, it comes clear that the option that could produce a better solution is to implement a sampling mechanism within the OpenFlow itself by biding the operations on the switch, as presented in this subsection.

4.4 PROPOSED METHOD

The wiser approach for a new solution to control sampling processes in SDN environments is to create custom actions in the OpenFlow switch to implement the sampling rules. Those customized actions would be added in the form of patches to the OpenFlow specification, resembling FleXam's implementation. Opposing to FleXam's approach, which unifies some sampling techniques in a single action, the ideal solution for this work would be that a single action corresponds to a single sampling technique.

For this patch, the OpenFlow version to be used must be at least version 1.0. The reason behind this choice is that OpenFlow version 1.0 was considered as the unified version from which all vendors should start adopting the OpenFlow standard, resulting in a large software support for version 1.0. If the patch is not implemented in the OpenFlow version 1.0, but instead on a newer version, the concern is what software should be used to provide support to work with that version.

The first sampling technique to be implemented (represented in Figure 18) will be a count-based sampling, where packets will be counted and then sent to the controller for storage, following selected parameters.

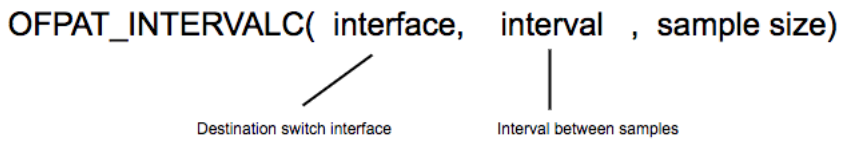


Figure 18.: Switch to Controller - Minimal Approach

OFPAT_INTERVALC represents the action name, that includes two parameters: interface and interval. The interface parameter indicates on which switch interface will the incoming packets be sampled and interval represents the counting interval in which the packets will be selected as sample.

Open vSwitch will be the OpenFlow implementation software where the path will be developed. Since it is largely used by the OpenFlow community, is widely supported from SDN software, has a solid OpenFlow implementation and there is documentation available about how it works. Open vSwitch offers the possibility to have a usable and practical developed patch.

For this implementation, it is intended to add a counter field in the packet code structure, first in the userspace datapath of Open vSwitch, and then transport it to the kernel module. With the counting implemented, it is time to implement a rule for all packets not selected for sampling to be forwarded to the right path. Selected packets should be duplicated with one copy being sent to the controller for sampling and another copy to be normally processed and forwarded.

The environment in which this work will be done is virtual. The choice of using Mininet remains the same has the ones presented in the First Approach section.

An OpenFlow's switch implementation will be used. A brief approach to what is available is required as, to make possible the production of results, the most suitable for this work is too the most used OpenFlow's switch.

4.4.1 *Using Mininet*

The Mininet network emulator allows a user to run several networking elements, such as virtual hosts and switches, using the same Linux kernel. Mininet's particularity of having switches supporting Openflow and SDN systems, alongside its popularity, was what raised interest in using it for this work.

Regarding OpenFlow-compliant switches, Mininet easily allows any user to work with three kinds of Openflow switches: the Openflow reference switch, the Open vSwitch and the ofsoftswitch13.

Open vSwitch is very popular among virtual switches since it can be managed by any controller, unlike other solutions that include a native controller. Also, it includes a kernel space module, absent in other switch implementations.

4.4.2 *OpenFlow*

Due to the numerous iterations in the development of OpenFlow's standard version the past years like, for example, seven releases between the years of 2011 and 2013, it's hard for network operators and manufacturers to stay in conformity with all those developments. This resulted in a poor adherence to some versions and a lack of backing by network devices for supporting those versions. The most used versions of OpenFlow are the 1.0 and the 1.3. The 1.3 version of OpenFlow raises the scope of functionalities from the previous versions (notice that version 1.2 was the first released by the ONF) and was targeted to be a base to the coming versions.

OpenFlow-based solutions were not fully considered to use in this context since sFlow uses a uniform per-packet sampling and in FleXam, besides supporting both a stochastically and deterministic sampling, only per-flow sampling is available. In addition, FleXam implementation is not available for use [46].

To implement the patch in the OpenFlow specification a switch implementation of it must be chosen to start working on it.

4.4.3 *Using Open vSwitch*

When choosing the OpenFlow implementation, the possibility of using the LINC switch was not considered since it uses Erlang, with which there is no coding language familiarity and does not have the popularity of C, used by most of the switch implementations. All Openflow reference switch, Open vSwitch and ofsoftswitch13 switch are implemented in platform-independent C language. However, these three types of switches have a detail that can make a huge difference, specially in the performance scope. While in the ofsoftswitch13 and OpenFlow reference switch the kernel space datapath is neglected and only validation of user space is required, the Open vSwitch architecture is more natively kernel space. Taking this into account, the later seems to be the way to go.

The Open vSwitch source code is available at Git, and can be easily cloned and modified [32]. It natively supports Netflow and sFlow.

Besides having integration in Mininet, Open vSwitch has a support team that can be contacted through a mailing list such as Mininet, and being that the same team that manages both Mininet and Open vSwitch, more knowledge is available. Open vSwitch is also very open to modifications since it relies on collaborations in order to correct bugs and add new functionalities. It includes several configurations to build and run code which are described in the documentation files.

The forwarding Components in Open vSwitch [34] are:

- ovs-switchd (Slow-Path, also known as userspace deamon)
  - Responsible for the forwarding logic.
- openvswitch_mod.ko (Fast-Path, also known as kernel module)
  - Packet manipulation and forwarding.

There are two possibilities, represented in Figure 19, of how the first packet can be handled by Open vSwitch. In one of them the ovs-vswitchd has previously instructed the datapath how to handle packets of the given type and, in the other case, it has not.

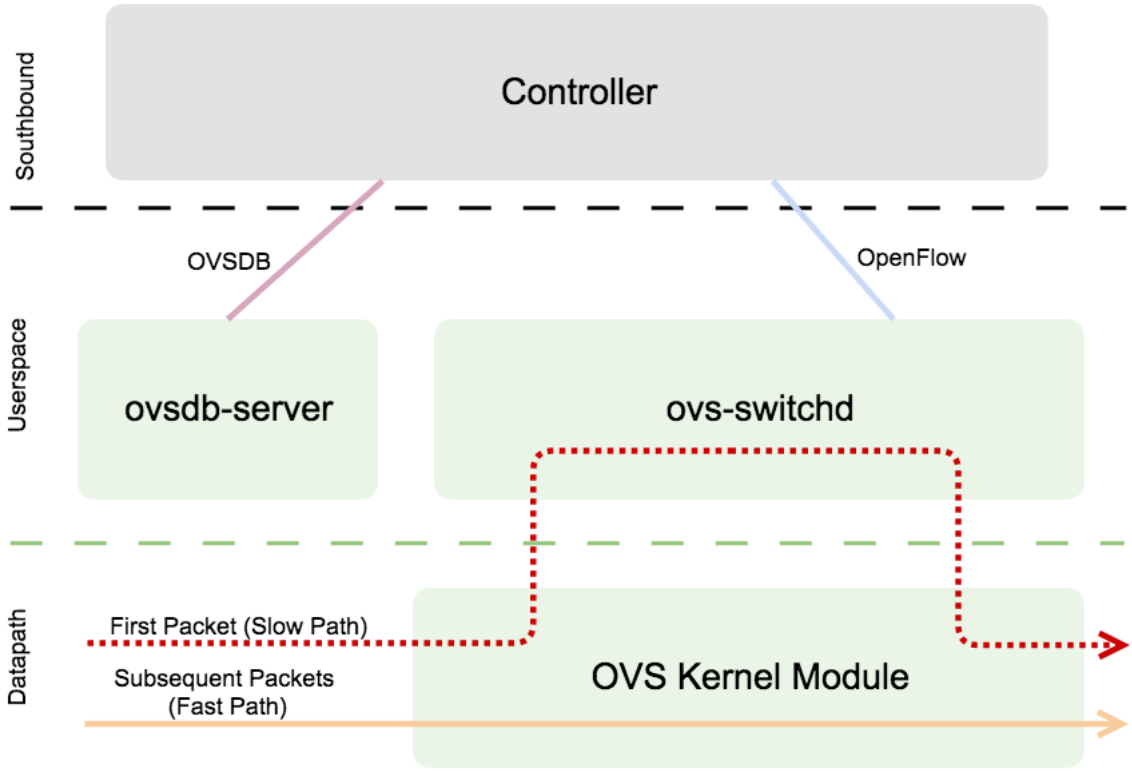


Figure 19.: Open vSwitch Forwarding Components and Operation

The first case is where the datapath simply follows the OpenFlow actions that were given by the ovs-vswitchd. These actions, beside giving instructions of where and how the packet should be transmitted, can also specify packet modifications, sampling (sFlow) or drop of the packet whenever the datapath receives a packet. If it does not have a previous instruction on how to operate the packet, the datapath delivers it to the ovs-vswitchd. There, the handling of the packet is determined, sending it back to the datapath to execute the instructions of how the packet should be handled [34].

Subsequent handling can be cached in the datapath by ovs-vswitchd command, so the datapath does not have to always ask the ovs-vswitchd how to handle packets that follow different actions.

Open vSwitch also includes some tools to manage and retrieve information about the operation of its components.

Listing of Open vSwitch tools and their functionality:

- ovs-vsctl - is a program primarily used to manage the ovs-vswitchd through its connection to the ovs-server, managing it to perform operations into a database;
- ovs-appctl - is used to change commands at runtime, printing the ovs-vswitchd response to it. Alongside ovs-vsctl, it is a tool to manage the ovs-switchd itself;
- ovs-ofctl - with ovs-ofctl one can monitor and administer OpenFlow switches, not just Open vSwitch;
- ovs-dpctl - this program acts on the Open vSwitch datapath, enabling CRUD (Create, Read, Update and Delete) operations on it.

An overview of Open vSwitch tools and their relation with userspace and datapath modules is presented in Figure 20.

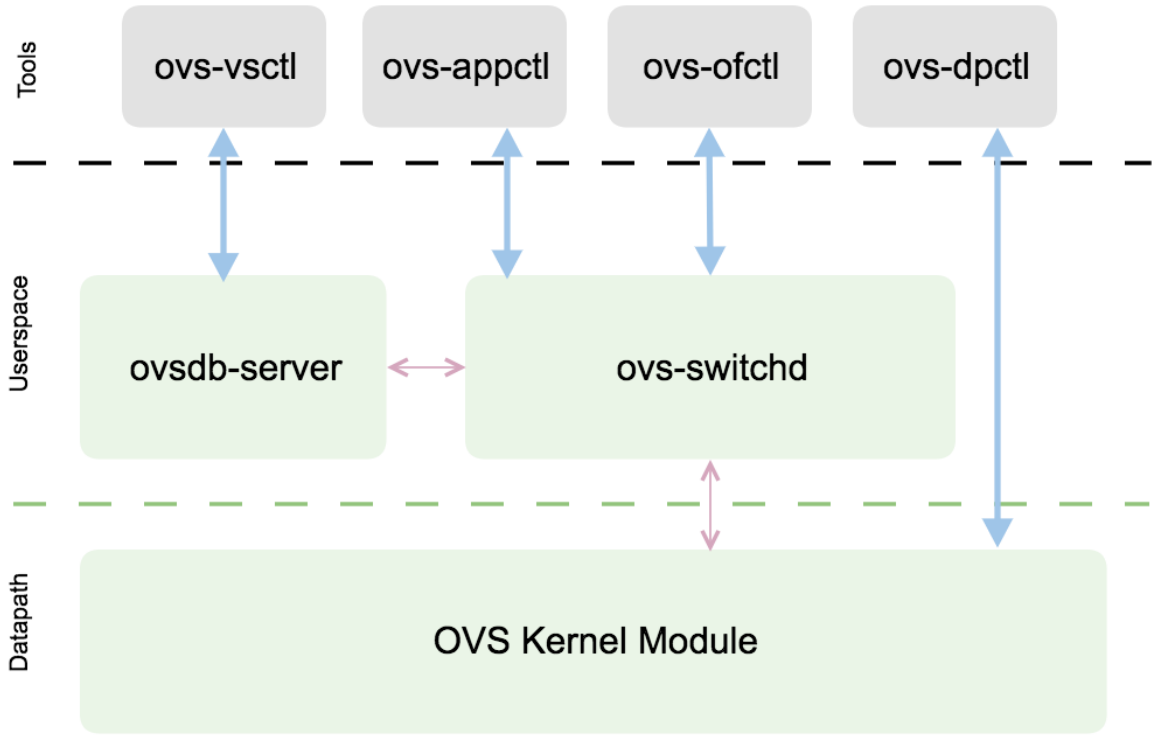


Figure 20.: Open vSwitch Tools and Components Relationship

Some implementation details of Open vSwitch kernel module are presented in this subsection.

As mentioned in Chapter 2, Section 2.5.1, when the Open vSwitch architecture is presented, it stated that the ovs-switchd (userspace datapath) communicates with the kernel module via the Netlink protocol. The communication between the kernel and userspace is done by the exchange of commands defined in the kernel in order to execute actions on packets. When a packet is received, it is viewed by the kernel as a struct vport that immediately searches the flow table using the function ovs_flow_tbl_lookup() for information (an unique key that identifies the flow) on how to operate the packet, on what is called as flow-cache. This flow key consists on packet information extracted by the ovs_flow_extract() function. This behaviour is executed when the datapath has information about how to handle the packet. When a packet that is not familiar arrives, the procedure changes, such as mentioned above when describing the Open vSwitch operation. In this situation, there is no flow cache, so the kernel module will communicate with the userspace using the commands defined in the kernel. The ovs-vswitchd will consult its ovsdb-database, collecting information on how to handle that packet (which OpenFlow action must be used). It also sends an execution command so the kernel executes the operation. The kernel module uses

the do_execute_actions() function to execute the commands ordered by the datapath, and forwards packets using the do_output() function.

There are some structures that are relevant to the Open vSwitch kernel module. They are:

- struct sw_flow - representation of a flow;
- struct sw_flow_actions - representation of actions on a flow;
- struct datapath - datapath representation;
- struct vport - representation of ingress and egress ports;
- struct sk_buff - representation of all the control information required for the packet hadling.

Figure 21 illustrates the relevant structures to the Open vSwitch kernel module and shows how they are linked.

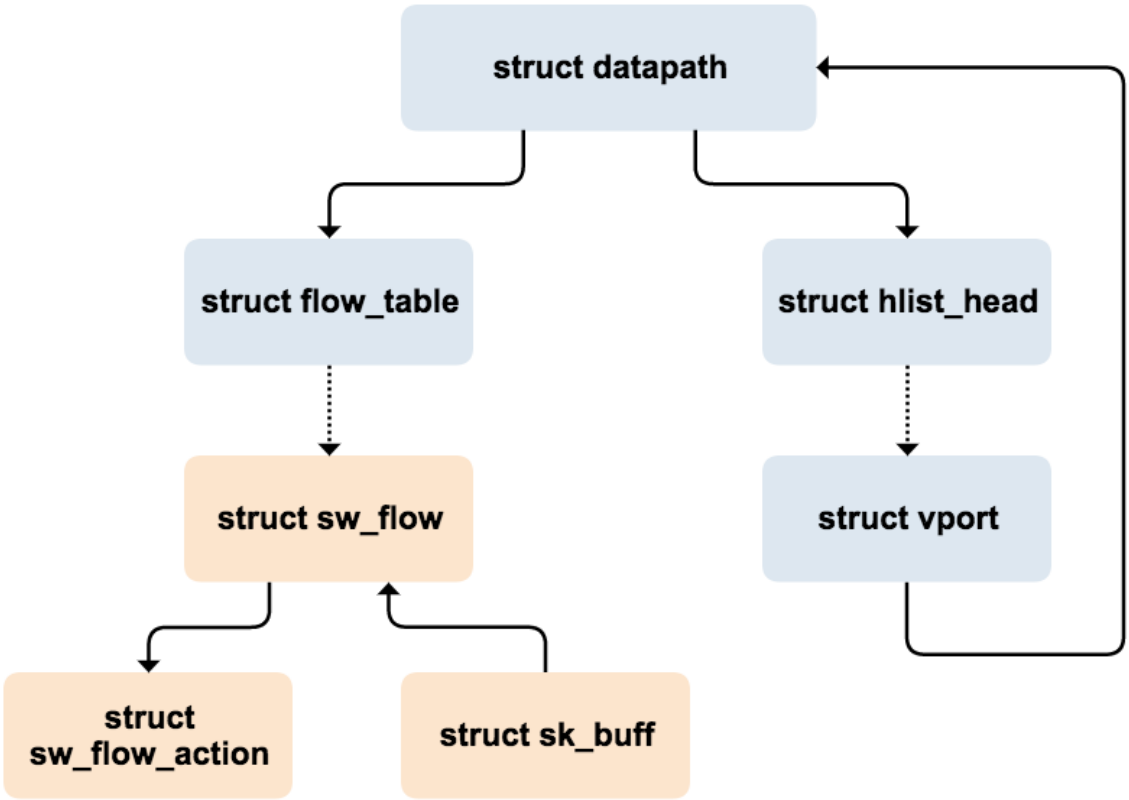


Figure 21.: Open vSwitch Kernel Module Main Data Structures

4.5 SUMMARY

This chapter has discussed ways to support sampling in SDNs. The first approach proposed to use what is available and wrap a solution that fits it. Since the OpenFlow protocol was not developed thinking on sampling, most of the solutions on top of it would be working with either low performance or accuracy, or both.

To face this challenge, the perspective of adding something to this protocol to make it work under these rules was shown up and followed. To do so, the use of Mininet testbed and Open vSwitch will be indispensable, being here discussed.

5

CONCLUSIONS AND FUTURE WORK

This chapter contains a reflexion of the developed work and a prospect for what can be done to fully test the proposal for applying packet sampling techniques in an SDN architecture.

5.1 SUMMARY

One of the most important aspects of the SDN architecture is to favour the introduction of new concepts and its high programmability. This ability to facilitate the changes in operations is mainly due to the separation between the control and data layer, allowing a single control on several data elements in the network.

Monitoring and sampling are essential tasks to perform in any system, and a system that works under an SDN concept is no different. While taking advantage of SDN features, the goal is to introduce sampling techniques that will allow to monitor the network. The data layer SDN protocol OpenFlow focus solely on the task of forwarding packets meaning that no concerns about monitoring the state of the network were taken into account when this protocol was designed. However, there are some tools based on this protocol to perform this kind of tasks, being the most known sFlow.

To have SDN working on a network, a control module is mandatory, so the rules can be sent to the data plane to be applied. In Chapter 2 a brief analysis of NOS was carried out.

A new proposal of development emerged after realizing that none of the existing tools executes monitoring tasks as intended. To begin, an approach based on not changing any of the network elements standard operations was made. However, this solution was considered invalid given that the performance decrease would not allow the solution to work on a real network environment.

Finding a solution on what to do to avoid this problem required a deconstruction of how the elements operate, and, by doing it on Chapter 4, the conclusion was that, for benefit of the solution, sampling operations had to use the data layer.

As previously said, the solution to be developed must consider that OpenFlow has limited resources when it comes to monitoring tasks, including sampling.

Throughout the use of SDN network emulator Mininet and, the OpenFlow implementation, Open vSwitch, both ensuring high programmability of features, this work proposes the implementation of a patch to the OpenFlow protocol that enables the sampling of packets, beginning by doing it so in a per-packet count-based sampling method.

5.2 Prospect for Future Work

The future work required for the proposed solution consists on the development, implementation and testing of sampling methods. A model of how the first sampling method can be built is presented on Chapter 4. To make a first functional patch the following steps are recommended:

1. Implement the sampling action on the Open vSwitch userspace datapath;
2. Test its functionality through the DPCTL tool that monitors and administrates OpenFlow datapaths;
3. To have it working on a real network environment, add a patch to a NOS.

5.3 Final Considerations

The goal of implementing flexible network measurements resorting to packet sampling and monitoring can be obtained by manipulating the OpenFlow configuration, since it is not prepared to provide flexible solutions for monitoring operations. To implement these changes, a lookup to all the elements in an SDN architecture is essential.

After analysing those elements, the conclusion of focusing on the data plane, more specifically on the switch capacity to handle packets immediately, provides an opportunity to process operations at working time. At the same time, it does not require more processing capacity or delegate too many functionalities to the switch.

Using other solutions, such as sFlow, can be considered since it provides a good solution to monitor SDN networks with an agent-based solution. If different methods for packet sampling are required, a change in the OpenFlow switch implementation is the key to achieve a solution that has the capability to be viable.

The conceptual background presented on this work provides the basis for the development and practical experiment required to test this solution.

BIBLIOGRAPHY

[1] J. M. C. Silva, "A modular traffic sampling architecture for flexible network measurements," *Doctoral thesis, Universidade do Minho, Braga, Portugal*, 2015.

[2] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[3] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," *ONF White Paper*, pp. 1–12, 2012.

[4] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[5] N. Duffield, "Sampling for passive internet measurement: A review," *Statistical Science*, vol. 19, no. 3, pp. 472–498, 2004.

[6] T. H. T. Zseby and B. Claise, "Packet sampling for flow accounting: Challenges and limitations," *Lecture Notes in Computer Science*, vol. vol. 4979, p. 61–71, 2008.

[7] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive Resource Management and Control in Software Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 18–33, 2015.

[8] A. Voellmy, H. Kim, and N. Feamster, "Procera: A language for high-level reactive network control," ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 43–48. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342451

[9] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," ser. ICFP '11. New York, NY, USA: ACM, 2011, pp. 279–291. [Online]. Available: http://doi.acm.org/10.1145/2034773.2034812

[10] N. Telegraph and T. Corporation, "Ryu network operating system," *https://osrg.github.io/ryu/*, 2016.

[11] M. McCauley, "Pox," *https://github.com/noxrepo/pox*, 2016.

[12] D. Erickson, "The beacon openflow controller," *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pp. 13–18, 2013.

[13] N. McKeown, "OpenFlow: Enabling Innovation in Campus Networks," 2008. [Online]. Available: http://archive.openflow.org/documents/openflow-wp-latest.pdf

[14] A. Doria, J. H. Salim, R. Haas, W. Wang, L. Dong, and R. Gopal, "Forwarding and control element separation (forces) protocol specification," Tech. Rep., 2010.

[15] J. Halpern and J. H. Salim, "Software-Defined Networking : Experimenting with the control to forwarding plane interface Extending the OpenFlow protocol with ForCES concepts ." *2012 European Workshop on Software Defined Networking*, pp. 91–96, 2012.

[16] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, D. Kreutz, and F. Ramos, "Software-Defined Networking: A Comprehensive Survey," pp. 1–61.

[17] I. Kovačević, "FoRCES protocol as a solution for interaction of control and forwarding planes in distributed routers ," 2009.

[18] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," *Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 493–512, 2014.

[19] B. N. Astuto, M. Mendonça, X. N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *Communications Surveys and Tutorials, IEEE Communications Society*, vol. 16, no. 3, pp. 1617 – 1634, 2014, accepted in IEEE Communications Surveys & Tutorials.

[20] Z. Wang, T. Tsou, J. Huang, X. Shi, and X. Yin, "Analysis of Comparisons between OpenFlow and ForCES," Internet Engineering Task Force, December 2011.

[21] E. Haleplidis, S. Denazis, O. Koufopavlou, D. Lopez, D. Joachimpillai, J. Martin, J. H. Salim, and K. Pentikousis, "ForCES applicability to SDN-enhanced NFV," in *Third European Workshop on Software Defined Networks*, 2014, p. 6.

[22] E. Haleplidis, S. G. Denazis, O. G. Koufopavlou, J. H. Salim, and J. M. Halpern, "Software-defined networking: Experimenting with the control to forwarding plane interface." in *EWSDN*. IEEE Computer Society, 2012, pp. 91–96.

[23] B. Heller, "OpenFlow Switch Specification v1.0.0," *Current*, vol. 0, pp. 1–36, 2009.

[24] K. Blaiech, S. Hamadi, P. Valtchev, O. Cherkaoui, and A. Beliveau, "Toward a semantic-based packet forwarding model for openflow," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE, 2015, pp. 1–6.

[25] B. Heller, "OpenFlow Switch Specification v1.1.0," *Current*, vol. 0, pp. 1–36, 2011.

[26] ——, "OpenFlow Switch Specification v1.2," *Current*, vol. 0, pp. 1–36, 2011.

[27] B. Pfaff, B. Lantz, B. Heller, C. Barker, D. Cohn, D. Talayco, D. Erickson, E. Crabbe, G. Gibb, G. Appenzeller, J. Tourrilhes, J. Pettit, K. Yap, L. Poutievski, M. Casado, M. Takahashi, M. Kobayashi, N. McKeown, P. Balland, R. Ramanathan, R. Price, R. Sherwood, S. Das, T. Yabe, Y. Yiakoumis, and Z. L. Kis, "OpenFlow Switch Specification v1.3.0," vol. 0, pp. 0–105, 2012.

[28] Open Networking Foundation, "OpenFlow Switch Specification v1.4.0," vol. 0, pp. 1–206, 2013.

[29] O. N. Foundation, "OpenFlow Switch Specification v1.5.0," *Current*, vol. 0, pp. 1–36, 2014.

[30] O. W. G. S. University, "Openflow switching reference." *git clone git://gitosis.stanford.edu/openflow.git*.

[31] I. . FlowForwarding, "Linc," *https://github.com/FlowForwarding/LINC-Switch*, 2016.

[32] Openvswitch, "Open vswitch," *https://github.com/openvswitch/ovs*.

[33] "Open vswitch." *http://openvswitch.org/*.

[34] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, pp. 117–130, 2015.

[35] CPqD, "Openflow 1.3 software switch," *http://cpqd.github.io/ofsoftswitch13/*, 2016.

[36] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "Ofswitch13: Enhancing ns-3 with openflow 1.3 support," *Proceedings of the Workshop on Ns-3*, pp. 33–40, 2016.

[37] N. at Politecnico di Torino (Italy), "The netbee library." *http://www.nbee.org/doku.php*, 2016.

[38] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based comparison and selection of software defined networking (sdn) controllers," in *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*. IEEE, 2014, pp. 1–7.

[39] P. Floodlight, "Floodlight. a java-based openflow controller," *http://www.projectfloodlight.org/*, 2016.

[40] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," pp. 19:1–19:6, 2010.

[41] K. Kaur, J. Singh, and S. Ghumman, "Mininet as software defined networking testing platform," *International Conference on Communication, Computing and Systems (ICCCS–2014)*, 2014.

[42] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg, "Mininet-wifi: Emulating software-defined wireless networks," *2015 11th International Conference on Network and Service Management (CNSM)*, pp. 384–389, 2015.

[43] N. S. 3, "ns-3 network simulator," *http://www.nsnam.org*, 2016.

[44] ——, "Gsoc 2010 openflow," *https://www.nsnam.org/wiki/GSOC2010OpenFlow*, 2016.

[45] R. Hand, M. Ton, and E. Keller, "Active security," *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, no. Section 2, p. 17, 2013.

[46] S. Shirali-Shahreza and Y. Ganjali, "Efficient Implementation of Security Applications in OpenFlow Controller with FleXam," in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*. IEEE, aug 2013, pp. 49–54.

[47] S. R. L. João Marco C. Silva, Paulo Carvalho, "Inside packet sampling techniques: exploring modularity to enhance network measurements," 29 March 2016.

[48] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Computer Networks*, vol. 62, pp. 122–136, 2014.

[49] P. Phaal, "Software defined networking," *http://blog.sflow.com/2012/05/software-defined-networking.html*, 2012.

[50] ——, "Openflow and sflow," *http://blog.sflow.com/2011/05/openflow-and-sflow.html*, 2011.

[51] S. Shirali-Shahreza and Y. Ganjali, "FleXam: Flexible Sampling Extension for Monitoring and Security Applications in OpenFlow," *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, p. 167, 2013.

[52] ——, "Traffic statistics collection with FleXam," *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*, pp. 117–118, 2014.

[53] A. L. Valdivieso Caraguay, L. I. Barona Lopez, and L. J. Garcia Villalba, "Evolution and Challenges of Software Defined Networking," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. IEEE, nov 2013, pp. 1–7.

A

INTRODUCTORY GUIDE TO MININET, RYU AND OPEN VSWITCH

An introductory guide on how to install and manage Mininet, RUY and Open vSwitch is presented, aiming at providing an easy and fast way to start working in this network environment.

Introductory guide to Mininet, Ryu and Open vSwitch

This guide was created to everyone interested to work with Software Defined Networking, using the network virtualization software Mininet and Open vSwitch virtual switches, with RYU Network Operating System running on the controller. Even for other Network Operating System, some topics may be suitable.
It is assumed that users are familiarized with computer programming and Linux Operating Systems. However, some unfamiliar aspects for a regular user may be highlighted.

Used Setup

Host:
Operating System: OS X

Mininet:
Network Virtualization: Mininet VM
Virtualization Software: Oracle VM VirtualBox

Installation Options

To start using Mininet on your machine you have three options:

➔ **Mininet Virtual Machine Installation**

It is recommended to choose this option mainly if you are starting from zero, since it gives you an already set machine, with a functional installation which probably just needs an upgrade to fulfill specific needs. For virtualization software, Oracle VM VirtualBox or VMware Workstation Player should equally serve the purpose. In this guide VirtualBox setup is used.

➔ **Native Installation**

If you decide to natively install Mininet, we recommend to do it on one of the more recent Ubuntu releases, specially because of support of the Open vSwitch newer versions.

- *Native Installation from Source*
- *Installation from packages*

Be aware that this installation may give you an older version of Mininet. You will have to install additional software like the Wireshark dissector.

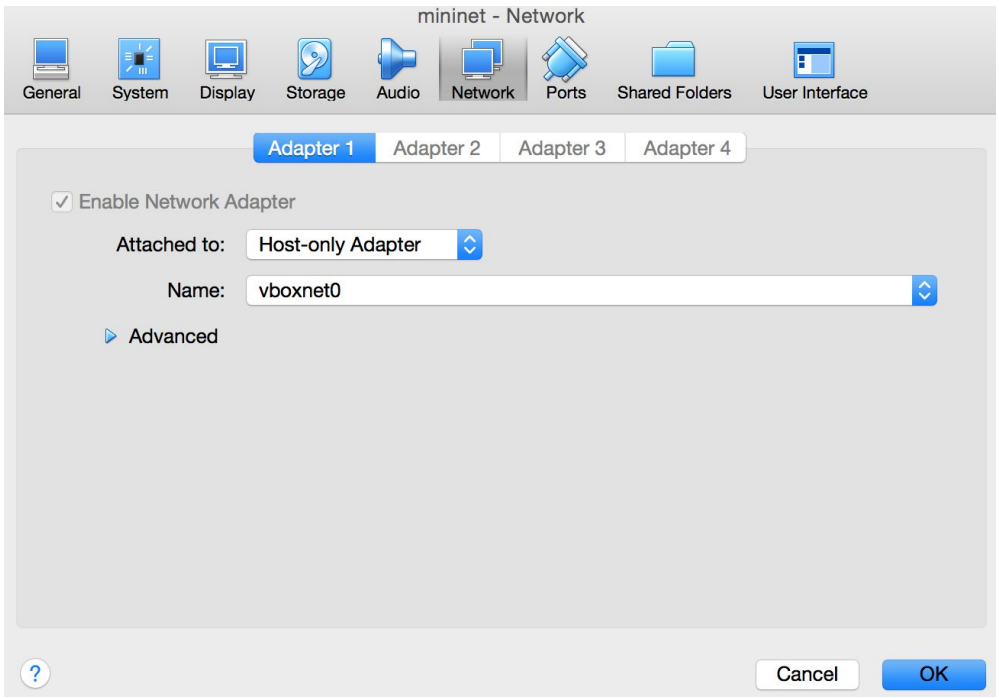
> ***TIP:***
>
> If you want to learn more about other options go to:
> **http://mininet.org/download/**

Setting VirtualBox:

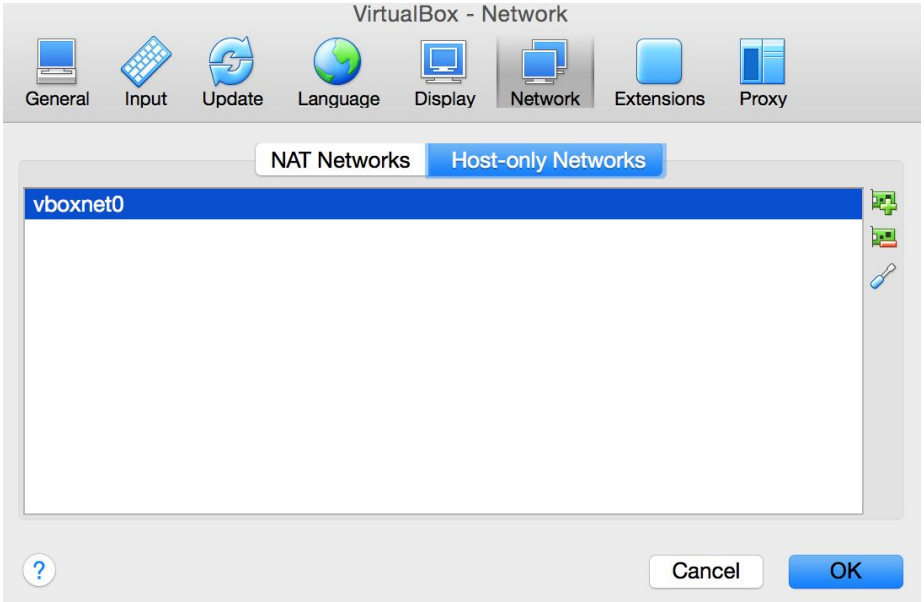
If your computer does not support Virtualization like VT-x (for Intel processors) or AMD-V (for AMD), you will not be able to install a 64-bits VM, even if your processor supports VT-d. To enable Virtualization you must boot to the BIOS and go to Advanced/Security tab.

CONNECT TO Virtual Machine (VM):

To connect via SSH you should choose the VM, go to Settings, Network Tab and attach to Host-only Adapter.



If you don't have any, you should go to VirtualBox **Preferences**, **Network** tab, **Host-only Networks** and add an adapter in "**+**".



You should now turn on the VM, insert the user credentials and retrieve the machine's IP address of the eth0 interface. The IP is identified as "inet addr".

```
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:b2:86:c6
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:103 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20694 (20.6 KB)  TX bytes:7009 (7.0 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:276 errors:0 dropped:0 overruns:0 frame:0
          TX packets:276 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:22036 (22.0 KB)  TX bytes:22036 (22.0 KB)

mininet@mininet-vm:~$
```

Open your computer's terminal and connect to the VM by SSH.

```
ssh -X [-Y] <username>@<IPaddress>
```

By connecting through SSH we also want to enable X11 forwarding so one can use a graphical client through the session (specially when using Wireshark).
The **-X** option means that the remote machine is not trustable, so if your command violates some security settings, an error is received. On the other hand, using **-Y** will mean you trust the remote machine, resulting on security settings not being checked. In this case the VM is running locally, and we can assume that the additional

precautions of **-X** are unnecessary. Using **-Y** implies no authentication timeout by default.

TIP:

If you are using a OS X or other UNIX-based Operating System you can do a shortcut to your machine's IP address by adding it to your hosts file.

OS X:
In Finder click in **Go** tab, select **Go to Folder** and write **/private/etc/hosts**. Open the file and add the line with `<IP> <hostname>` information. If you want to do it by the terminal you should go outside your user's folder and the go to private/etc/ and the open the hosts file with, for instance text editor nano and add the IP and hostname.

CHECK VERSIONS:

Before working with Mininet or installing any new software, I advise checking the current version of, Open vSwitch and Python. The reason why checking the Open vSwitch version can be important is that if you specifically want to use an OpenFlow version after 1.0.0, there is a possibility that the current Open vSwitch installation does not support it fully, so probably you need to update this package. Python should be checked because there were some changes to the code semantics between versions and to develop recent software, a recent Python version, such as Python3, should be used.

```
ovs-vswitchd --version
python --version
```

INSTALLING ADDITIONAL SOFTWARE:

Additional software may be required and before doing anything, one must update the system with:

```
sudo apt-get update
```

After that, and specially if you are using Python-related applications, it is advisable to install pip, which is the Python packet management system useful to install and manage software packages.

```
sudo apt-get install python-pip
```

The Mininet Virtual Machine does not provide any graphical environment for deployment. To avoid using terminal text editors like Nano or Vim, install gedit that is a versatile text editor.

```
sudo apt-get install gedit
```

GRAPHICAL INTERFACE:

Some software that require a graphic interface can be installed but for that to work, support of the X Window System is mandatory. If you are using OS X, XQuartz must be installed so your ssh session support X11. Note that some versions of XQuartz do not provide support for high-resolution Retina displays. Those will run in pixel-double mode.

UPDATE OPEN vSWITCH:

To update the existing installation of Open vSwitch you must do this 4 steps:

- Remove old packages;
- Download and unpack OpenVSwitch;
- Build Debian packages and install;
- Runs the unit tests.

> ***TIP:***
>
> To apply the instructions correctly, the instruction file on git is very useful. Go to:
>
> **https://github.com/mininet/mininet/wiki/Installing-new-version-of-Open-vSwitch**

INSTALL RYU:

To install Ryu Network Operating System, you can either use pip (if you have it installed, if not, see the instructions above), or clone it from git.

From pip:
```
pip install ryu
```

From git:

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

Customization of the network topology in Mininet:

Mininet provides what is called “minimal” topology (used has `mn` CLI) that consists of one OpenFlow reference controller and OpenFlow kernel switch connected to two hosts. There are some details that can be changed through the addition of some parameters like `--topo` or `--controller` that extend the `mn` command. If a major change is to be done, the `--custom` option can be used to invoke Python scripts that use the Mininet Python API to use a full customized topology. This, besides letting the user set the number of hosts and switches, makes it easier to implement starting configurations on them specially in the case of using a specific OpenFlow version, where all elements have to be configured.

TIP:

To start using the Python API that Mininet provides for parametrizing your topology, there are three things you can look for:

The reference manual of the API:

http://mininet.org/api/annotated.html

Documentation which is available using Python documentation strings, using `help()` mechanism.

Mininet also comes with examples available in the /examples file on the virtual machine or in git at:

https://github.com/mininet/mininet/tree/master/examples

B

PYTHON CODE TO CREATE CUSTOM TOPOLOGY IN MININET

Python script that builds the parametrized topology in Mininet.

```
# Python script to create a network topology
# Three directly connected switches plus a host for each switch:
#
#               switch1 --- switch2 --- switch3
#                  |           |           |
#                  |           |           |
#                host1       host2       host3
#
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import RemoteController, OVSSwitch
from functools import partial
from mininet.util import dumpNodeConnections

class MyTopo(Topo):
    def __init__(self, **params):

        # Initialize topology
        Topo.__init__(self, **params)

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        middleHost = self.addHost( 'h2' )
        rightHost = self.addHost( 'h3' )

        leftSwitch = self.addSwitch( 's1' )
        middleSwitch = self.addSwitch( 's2' )
        rightSwitch = self.addSwitch( 's3' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( middleHost, middleSwitch )
```

```
        self.addLink( rightHost, rightSwitch )

        self.addLink( leftSwitch, middleSwitch )
        self.addLink( middleSwitch, rightSwitch )

def setup():
    Topo = MyTopo()
    # Using Open vSwitch and OpenFlow 1.3
    Switch = partial(OVSSwitch, protocols='OpenFlow13')
    #controller
    net = Mininet(topo=Topo, switch=Switch, controller=RemoteController)

    # Setting up hosts
    net['h1'].setIP('10.0.1.1/24')
    net['h1'].setMAC('00:00:00:00:01:01')
    net['h1'].cmd('route add default gw 10.0.1.100')
    net['h2'].setIP('10.0.1.2/24')
    net['h2'].setMAC('00:00:00:00:01:02')
    net['h2'].cmd('route add default gw 10.0.1.100')
    net['h3'].setIP('10.0.1.3/24')
    net['h3'].setMAC('00:00:00:00:01:03')
    net['h3'].cmd('route add default gw 10.0.2.100')

    # Setting up switches
    net['s1'].cmd('ifconfig s1-eth1 hw ether 00:00:00:11:11:01')
    net['s1'].cmd('ifconfig s1-eth2 hw ether 00:00:00:11:11:02')
    net['s2'].cmd('ifconfig s2-eth1 hw ether 00:00:00:22:22:01')
    net['s2'].cmd('ifconfig s2-eth2 hw ether 00:00:00:22:22:02')
    net['s2'].cmd('ifconfig s2-eth3 hw ether 00:00:00:22:22:03')
    net['s3'].cmd('ifconfig s3-eth1 hw ether 00:00:00:33:33:01')
    net['s3'].cmd('ifconfig s3-eth2 hw ether 00:00:00:33:33:02')

    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    CLI(net)
    net.stop()

if __name__ == '__main__':
    # Printing useful information
    setLogLevel('info')
    setup()
```

mytopo.py