Ayesha Imran

SDN Controllers Security Issues

MS Thesis document in Web Intelligence and Service Engineering

November 9, 2017

University of Jyväskylä

Department of Mathematical Information Technology

Author: Ayesha Imran

Contact information: ayesha.a.imran@student.jyu.fi

Supervisors: Prof. Timo Hämäläinen, and Dr. Muhammad Zeeshan Asghar

Title: SDN Controllers Security Issues

Työn nimi: SDN-ohjaimet Turvallisuusongelmat

Project: MS Thesis document

Study line: MS-WISE

Page count: 62+0

Abstract: Software-Defined Networking (SDN) is essentially varying the way we design and manage networks, which makes a communication network programmable. In SDN, a logically centralized controller has straight control over the packet-handling functions of the network switches, using a standard, open API (Application Programming Interface) such as OpenFlow. OpenFlow is a broadly used protocol for software-defined networks (SDNs) that presents a new model in which the control plane is inattentive from the forwarding plane for the network devices. In SDNapproach centralized entities called "controllers" perform like network operating systems run dissimilar applications that accomplish and control the network via well-defined APIs. SDN permits network engineers and administrators to reply rapidly to the changing business requirements by determining traffic from the central controller deprived of having to touch the physical switches. They use software to rank, redirect or block traffic globally or in varying degrees down to individual packet levels. In short, SDN can deal with the more protected network if it is properly secured. SDN has the ability to transform the network industry. It is observed that currently enterprises requiring SDN deployment demand, multiple controllers that lead security challenges associated with SDN Controller design. In this thesis, a comprehensive literature review concerning SDN controllers security has been done. We demonstrate a comprehensive studies on SDN by clarifying its concept, the OpenFlow protocol architecture and how it works, general benefits and applications of SDN. We also present a discussion on SDN controllers and several threat

vectors which may enable for the exploitation of the vulnerabilities of SDN Controllers. This study also focuses on designing a dependable Controller platform including the requirements for a secure, resilient, and robust SDN Controller. The thesis is finalized by discussing that how we can secure SDN controllers by making recommendations for security improvements for future SDN Controllers. The discussion highlights the existing gap between the actual security level of the current SDN Controller design and the potential security solutions.

Keywords: SDN, SDN Controllers, DoS attacks, Openflow Controller

Suomenkielinen tiivistelmä: Ohjelmistokohtainen verkkoyhteys (SDN) on olennaisesti eri tavoin suunniteltu ja hallinnoitu verkkoja, mikä tekee viestintäverkon ohjelmoitavaksi. SDN: ssä loogisesti keskitetyllä ohjaimella on suorat käskyt verkkokytkinten pakettikäsittelytoiminnoista käyttäen standardia avointa API: ta (Application Programming Interface), kuten OpenFlow. OpenFlow on yleisesti käytetty protokolla ohjelmistoverkkoihin (SDN), jotka esittelevät uuden mallin, jossa ohjaustaso ei huomioi verkkolaitteiden välitystasoa. SDNapproach-keskusyksiköissä, joita kutsutaan "ohjaimiksi", suorittavat samankaltaiset verkko-käyttöjärjestelmät käyttävät erilaisia sovelluksia, jotka suorittavat verkon ja hallitsevat sitä hyvin määriteltyjen sovellusrajapintojen kautta. SDN sallii verkkoinsinöörit ja ylläpitäjät vastaamaan nopeasti muuttuviin liiketoiminnan vaatimuksiin määrittämällä liikenteen keskusohjaimelta, joka ei ole joutunut koskettamaan fyysisiä kytkimiä. He käyttävät ohjelmistoja sijoittamaan, ohjaamaan tai estämään liikennettä maailmanlaajuisesti tai vaihtelevasti yksittäisten pakettitasojen alapuolella. Lyhyesti sanottuna SDN voi hoitaa suojatun verkon, jos se on asianmukaisesti suojattu. SDN pystyy muuttamaan verkkoalalla. On havaittavissa, että nykyään yritykset, jotka vaativat SDN: n käyttöönottoa, tarvitsevat useita ohjaimia, jotka johtavat SDN Controller -suunnitteluun liittyviin turvallisuusongelmiin. Tässä työssä on tehty kattava kirjallisuuskatsaus SDN-ohjainten turvallisuudesta. Esittelemme kattavia tutkimuksia SDN: stä selkeyttämällä sen käsitettä, OpenFlow-protokollaarkkitehtuuria ja sen toimivuutta, SDN: n yleisiä hyötyjä ja sovelluksia. Esittelemme myös keskustelun SDN-ohjaimista ja useista uhkaavektoreista, jotka voivat mahdollistaa haavoittuvuuksien hyväksikäytön SDN-ohjaimia. Tämä tutkimus keskittyy myös luotettavan ohjainlaitteen suunnitteluun, joka sisältää turvallisen, joustavan ja kestävän SDN-ohjaimen vaatimukset. Opinnäytetyö viimeistellään keskustelemalla siitä, miten voimme turvata SDN-ohjaimet tekemällä

suosituksia turvallisuuden parantamiseksi tuleville SDN-ohjaimille. Keskustelussa korostuu nykyisen SDN-ohjaimen todellisen suojaustason ja mahdollisten tietoturvaratkaisujen välinen ero.

Avainsanat: SDN, SDN-ohjaimet, DoS-hyökkäykset, Openflow-ohjain

Preface

I am grateful to Allah Almighty, the omnipotent, the most merciful and beneficent. His blessings enable me to achieve my goal. Best of praises for the entire messenger and especially the last messenger Holy Prophet Hazrat Muhammad (P.B.U.H) who is always torch of guidance and knowledge for humanity. Special and profound thanks to Pakistan and Finland as a country whom I love from the depth of my heart.

I have a reverence and admiration for my supervisor Prof. Timo Hämäläinen. I am highly indebted for his moral support, patience and enable guidance all through the study.

Special thanks to Dr. Muhammad Zeeshan Asghar, who is my Co-supervisor. It was his efforts that I am able to successfully complete research work. He helped and motivated me a lot during my research work.

In my personal life, I would like to express my great feelings of emotions for my grandparents and for my Uncles and Aunt. Their infallible love and support have always been my strength.

I feel at the loss of words to express my thanks to my parents Muhammad Zafar and Alam Khatoon, who work so hard that life becomes so easy and pleasant for me without their precious efforts, prayers, moral, spiritual, and financial support it would not be possible for me to produce this piece of work. Their patience and sacrifice will remain my inspiration throughout my life. I feel a deep sense of gratitude for my sisters Sadia and Bushra for their untiring help and guidance throughout my studies. For supporting me throughout writing this thesis and my life in general. Without their help, I would not have been able to complete much of what I have done and become who I am.

A creative energy in my soul cannot be resisted like a storm when my caring, loving family evokes me to devote myself to my planned schemes and ensure me to a clear path before me. In the end, I would like to say thanks to my other family members and friends for their support and encouragement.

Jyväskylä, November 9, 2017

AYESHA IMRAN

Glossary

SDN controller	The Software Defined Network Controllers is essentially the "brain" of the network (Yoon and Kim 2015).It manage the flow-control to the routers or switches through the Southbound AIPs and the application the business logic through the northbound APIs (Yoon and Kim 2015).
OVSDB	The Open Virtual Switch Database is management protocol which supports the communication between the network devices in SDN system. This Protocol guides how the SDN Controllers and the network devices exchange statistical and control information (Bittman et al. 2013).
i2rs	The Interface to the Routing System developed by the Internet Engineering Task Force (IETF) with the objective of offering an efficient routing operation. The design of the i2rs architecture is mainly to enhance the network control and allow applications to build on top of the system (Bittman et al. 2013).
Ryu	Ryu is an Open Source SDN Controller which is designed to provide various software components for use in SDN applications. The Ryu OpenFlow Controller supports the creation of new networks management and controlling of applications. According to (eg. "Ryu Controller" 2017).
NBI	A North Bound Interface (NBI) can be called as a boundary between the controller and the application layer. It supports most of the application layer protocols for interactions i.e. HTTPS, SFTP etc.
SBI	The South Bound Interface (SBI) is responsible for the communication between the controller and the underneath switches.SBI integration is also supported by Simple Network Management Protocol (SNMP), Command Line Interface (CLI), etc. (Arbettu et al. 2016).

List of Figures

Contents

1 Introduction

Computer networks are designed to build from a large number of network devices such as routers, switches and various types of middleboxes. Meaning that the devices with many complex protocols implemented on them which control the traffic for other purposes than packet forwarding, for instance a firewall. Network operators are responsible for designing the strategies to deal with a broad range of network events and applications. Though adapting to changing network conditions they perform the manual transformation of these high level-strategies into low-level configuration commands. Usually, they have a very restricted access to the tools to complete these very difficult tasks.

Thus, the modification of network management and performance is relatively challenging and thus error-prone. The reason that the network devices are generally vertically-integrated black boxes worsens the challenge network operators and administrators face.

One more unbeatable challenge network consultants and scholars face has been stated as "Internet ossification". The reason is that its massive deployment base and the fact it is considered part of our society's critical infrastructure. Internet has become enormously problematic to progress both in terms of its physical infrastructure as well as its protocols and performance. Nevertheless, as current and evolving Internet applications and services become progressively more complex and demanding, it is imperative that the Internet be able to progress to address these new challenges.

The awareness of "programmable networks" has been proposed as a way to simplify network development. In particular, Software called SDN (Nunes et al. 2014).

A software-defined network (SDN) is a new networking paradigm that has come up with new opportunities for network enthusiasts to experiment and organize advanced ways of network management and to enthusiastically take control of packet forwarding in their network. It gives the hope to change the limitations of current network infrastructures.

The concept of this is based on splitting the network intelligence out of the packet switching device and putting it into a logically centralized controller. The forwarding decisions are

made by the controllers, which are located into the switches via standard protocols, like OpenFlow(Lara, Kolasani, and Ramamurthy 2014) .

OpenFlow is a standard for a communications protocol that enables the control plane to interact with the forwarding plane. Keeping in mind that OpenFlow is not the only protocol available or in development for SDN. OpenFlow is widely used in implementation architecture of SDN.

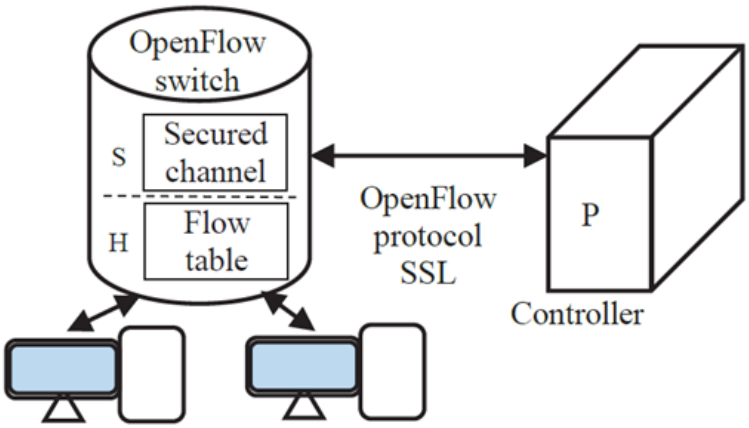

Figure 1. OpenFlow protocol communication

Software Defined Networking (SDN) is a new architecture thathas been aimed to allow more agile and cost-effective networks. It allows dynamic reconfiguration of the network by taking a new methodology to the network architecture.

The SDN architecture could be generally decomposed into three main layers (Fig. 1)

- Data Plane
- Control Plane
- Application Plane

Let's consider the layers from bottom to top. Data plane is the bottom most layer of SDN architecture, this layer deals with the implementation of data-path. It comprises of switches, which get flow rules from higher layers as an instruction which will be continued in the switches flow table. In some cases, if the received packet does not match any entry in the

flow table, the switch is responsible to forward that packet to the controller for a decision.

The middle layer is the control plane is responsible or the implementation of the control-path of a legacy network. This is the most critical layer of an SDN architecture which accepts the traffic tasks i.e. traffic engineering, traffic shaping, and network management. The First layer is the application layer with the help of a controller this layer is responsible for the customization of packet forwarding, policy management, user management, and Quality of Service (QoS). In SDN architecture all of the network functions and monitoring tools are usually parts of the application layer.

There are two protocols which are used to help in the communication between different layers(Fig. 2)

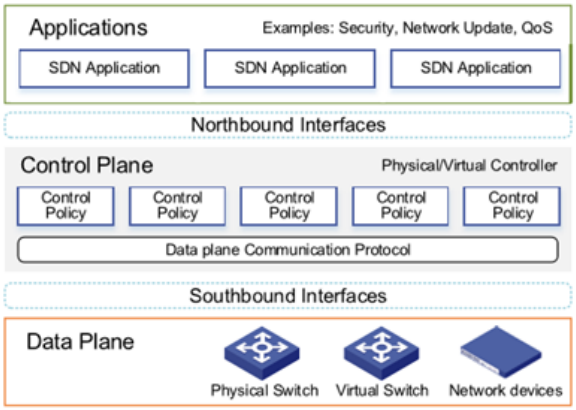

Figure 2. SDN Architecture

- A North Bound Interface (NBI) can be called as a boundary between the controller and the application layer. It supports most of the application layer protocols for interactions i.e. HTTPS, SFTP etc.
- The South Bound Interface (SBI) is responsible for the communication between the controller and the underneath switches, OpenFlow is being widely used as a SBI protocol. SBI integration is also supported by Simple Network Management Protocol (SNMP), Command Line Interface (CLI), etc. (Arbettu et al. 2016)

It deals with cloud and network engineers and administrators to answer promptly to changing business requirements through a centralized control support. It includes multiplekinds of

network technologies which are aimed to make the network more easy and agile to support the virtualized server and storage structure of the modern data center and it was formerly defined an approach to designing, building, and managing networks that separates the network's control ("Software-Defined-Networking" 2017).

1.1 Motivation

Traditional networks merge the control and data planes on a physical device, which characteristically consists of exclusive hardware and software. Software-defined networks bring the feature to the control plane out to a SDN controller. SDN controller uses a protocol such as OpenFlow to control switches, which are responsible for handling the data plane. Considering a security perspective, this division of responsibility has both advantages and disadvantages. The ability to easily separate the control plane network from the production data network is an obvious advantage. The SDN controller's has the capability to control complete network makes it a very high value target, which is a possible disadvantage compared to a traditional network with a more distributed control plane. Moreover, the control plane administration interfaces wide-open by the SDN controller, it has another attack surface: the data plane of the switches it manages. When an OpenFlow switch come across a packet that does not match any forwarding rules, it passes this packet to the controller for instruction. Thus, it is very easy for an attacker who is simply able to send data through an SDN switch to exploit vulnerability on the controller.

1.2 Research Objectives

The goal of this thesis is make a a comprehensive literature review concerning SDN controllers security. To meet these goals, the research aims to answer the following research questions:

- Why SDN is important in today's modern datacenters?
- What type of issues and challenges have in SDN Controllers security?
- How the design of a dependable Controller platform can be improved?
- How to enhance the security of future SDN Controllers?

- What are the existing gap between the actual security level of the current SDN Controller design and the potential security solutions?

1.3 Thesis Outline

The structure is as follows, Chapter 2 clarifies the SDN concept, the OpenFlow protocol architecture and how it works, general benefits and applications of SDN. Chapter 3 discusses the SDN controllers. Chapter 4 is about securing SDN controllers. Finalized by Chapter 5, which discusses how we can secure SDN controllers and then Chapter 5 concludes this thesis.

2 SDN Concept

Software-defined networking (SDN) is a methodology in computer networks, which allows the controlling of the network and the improvement of new network functions. Instead of managing each network device separately through a vendor-specific interface, in SDN the management of the network can be centralized to a specific SDN controller. This gives new opportunities for designing computer networks and makes the administration easier than before.

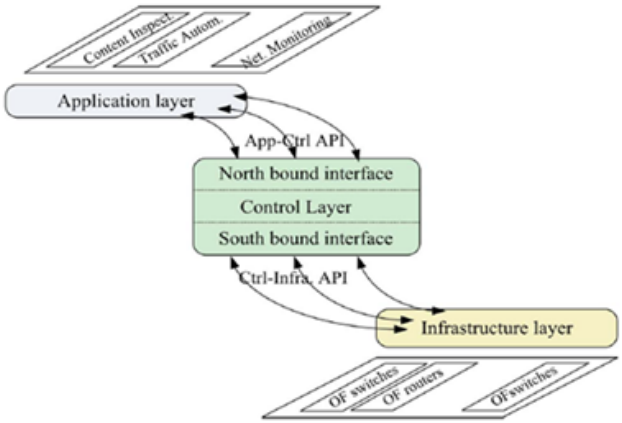

Figure 3. The three layers in SDN architecture

The major difference between SDN and the earlier approaches is that a software component running on a server or a CPU is added to the architecture of the network. The software component in SDN is responsible for the control plane of the network. That's the reason why we say that SDN decouples the control and data planes, as this difference was not as clear in previous approaches.

One important characteristic of SDN is its capability to provide a network wide abstraction. Keller et al. (Keller and Rexford 2010) discuss the idea of the "platform as a service" model for networking. According to the authors, it is a common trend to decouple the infrastructure management from the service management. In this model, the underlying physical network and the topology are hidden to the user. Instead, the abstraction presented to the user is a

single router. According to them, the customer is mostly interested in being able to configure policies and defining how packets are handled. We will see during the rest of this survey that a large number of publications aim at hiding the complexity of the network and providing an easier way to configure a service. Using names instead of IP addresses or high level policies instead of access control configuration files are examples of this abstraction.

2.1 Three layers in SDN architecture

The concept of SDN (shown in Fig 3) is to detached the control layer and merge it to one single point of network which means that every single network device need only take care of data layer and move data packets from one point to another based on the forwarding decisions made by the SDN controller.

Therefore every switch is controlled by one specific controller through application programming interface (API) and the controller is directed with application layer SDN applications.

- **Infrastructure layer:** All the hardware exists and is connected physically, in this layer. Software run on these hardware devices which provide a control data plane interface (Southbound API) which is used to communicate with the upper level.
- **Control layer:** This is the most important layer in SDN architecture. It has a controller which communicates to all the network devices in the infrastructure and it saves track of the topology. During the exchange of information of the network state with upper layer applications (through (Northbound API), the controller makes the commands understandable to the network devices to have corresponding and desired network behavior.
- **Application layer:** All features, services and policies are defined in this layer. Information of the network devices is requested by the applications and the topology in order to act upon it. According to the changes in the network these applications can create features end-to-end and make big picture decisions. As soon as the network topology, feature, or policy requirements changes, applications have the control to change dynamically the network behavior from one single point.

There are Application Programming Interfaces (APIs) between these layers, which provide

the important communication tools between the layers.

2.2 Northbound API

The Northbound API is provided by the controller and the applications have to manage their communication to the controller through it. This is a technique to manage the controller and the whole SDN network. Northbound API is generally implemented using restful API (representational state transfer); by this we can easily manage the controller with basic HTTP-methods like POST, GET, PUT and DELETE.

In SDN, the northbound API interface on the controller enables applications and the overall management system to program the network and request services from it. This application tier often includes global automation and data management applications, as well as providing basic network functions such as data path computation, routing, and security. Currently, no formalized standards have been ratified for northbound APIs, with several dozen open and proprietary protocols being developed using different northbound APIs.

The lack of a standard API is likely due to the varied nature of applications sitting above the controller, which can include managing cloud computing systems, network virtualization schemes, and other disparate or specialized functions. Nevertheless, work on open northbound APIs is being done for specific vertical applications.

OpenStack, a cloud computing effort backed by Arista Networks, Big Switch Networks, Brocade, VMware, and other SDN vendors, has developed the Quantum API, which is a vendor-agnostic API for defining logical networks and related network-based services for cloud-based systems. Several vendors have developed plugins for Quantum, which has helped it to become the default networking API for OpenStack, one of the largest open source cloud management platforms (Kirkpatrick 2013).

2.3 Southbound API

The Southbound API is the communication between the controller and the network devices. This environment is centralized to SDN controller and switches are managed by the con-

troller through southbound API.

The controller, view the network and it is configuring necessary flows to every switch under the controller. Switches will keep up flow tables which tell where to forward packets. In case if the switch can't make decision based on beforehand programmed flows, it will implement the configured default action that can be for example sending the packet to the controller.

Though not explicitly required by SDN, OpenFlow is a protocol often used as the southbound API that defines a set of open commands for data forwarding. These commands allow routers to discover the network's topology and define the behavior of physical and virtual switches, based on application requests sent via the northbound APIs. Note, however, that while commonly used in SDN architectures, OpenFlow is not a requirement of SDN, and organizations may opt to use other types of southbound APIs for the control of switches and devices.

According to Dan Pitt, executive director of the Open Networking Foundation, a trade organization working to promote software-defined networking and the use of the OpenFlow protocol, the open protocol can assist organizations in scaling and reconfiguring their networks, while supporting the growing trend of network virtualization.

"The perpetuation of manual configuration through command-line interfaces has long held networking back from the advances in virtualization enjoyed by the computing world, and has led to high operating costs, long delays in updating networks to meet business needs, and the introduction of errors," Pitt says. "Eliminating the need to tie applications to specific network details like ports and addresses makes it possible to evolve the network's physical aspects without the delay and cost of both rewriting the applications and manually configuring the network devices"(Arbettu et al. 2016).

2.4 OpenFlow protocol

2.4.1 Overview

OpenFlow is widely used protocol for software-defined networks (SDNs) that presents a new model in which the control plane is abstracted from the forwarding plane for the network

devices. This approach varies from the conventional networking architecture, where both planes exist in on the same networking device. In centralized SDN approach, main entities are called as "controllers" which act like network operating systems run different applications that cope and control the network via well-defined APIs. The forwarding plane in SDN architecture is OpenFlow switch that consists of tables of packet handling rules. Traffic passing the switch is compared against these rules and a match – action method is applied to the traffic. Depending on the rules installed by a controller application, an OpenFlow switch can act as a router, a switch, or a middlebox without much caring about what kind of vendor to use in the network. In Data centers networking is one of the most powerful applications that presented effective integration with OpenFlow protocol by making the network more reliable to the speedily expanding number of virtual machines. On the other hand the growing traffic in the data centers, the need for high controllers performance increases.

In mid-2000, the numerous researchers in the scope of Internet technologies have begun to show greater interest in large networks which are proficient of performing tests and trying out new technologies and protocols. NFS (National Science Foundation) have established GENI project (Global Environment Networking Innovations). Essential architecture for creating new solutions in numerous fields of IT was made. OpenFlow protocol arose from one of such projects. Research group on Stanford University have created Clean State program and focused on local area testing that could be easier to control. The campus network of all universities in the United States was created. This allowed procreation of protocol which could replace L2 and L3 protocols. The progress of OpenFlow protocol is still an ongoing process. Each fresh version of OpenFlow protocol comes up with new features (Godanj, Nenadić, and Romić 2016).

OpenFlow deals with the concept that switches have forwarding tables and an open API which OpenFlow controller is using. The controller has the understanding of the network; it will populate the required forwarding rules to the switch. Described in above (figure 4) of an OpenFlow switch, it primarily states the communication between an OpenFlow controller device and an OpenFlow switch and also with the performance of data plane function in an OpenFlow switch. This communication is done through a safe channel where both the OpenFlow control messages and the transferred data packets are moving from the controller

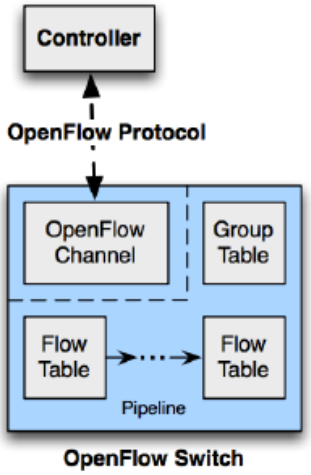

Figure 4. Three main parts of Openflow Switch

to a switch and vice versa. OpenFlow switch includes pipeline processing including multiple flow tables where the incoming data packet is processed.

2.4.2 OpenFlow Architecture

The OpenFlow network architecture consists of three basic models:

- OpenFlow-compliant switches that comprise the data plane.
- Control plane comprises one or more OpenFlow controllers.
- Secure control channel links the switches with the control plane.

Communication of switches is done with the hosts and with each other using the data path software can provide, communication of controller with switches is done by using the control path as shown in Figure 5 .

Secured connection is maintain between the OpenFlow controller and the switch by using SSL or TLS cryptographic protocols, in this case the switch and the controller are jointly authenticated by exchanging certificates signed by both sides' private key. Even though this is a very powerful security algorithm, the controller may be vulnerable to denial of service (DoS) attack, or Man in the middle attack; thus, suitable security practices must be implemented to stop such attacks.

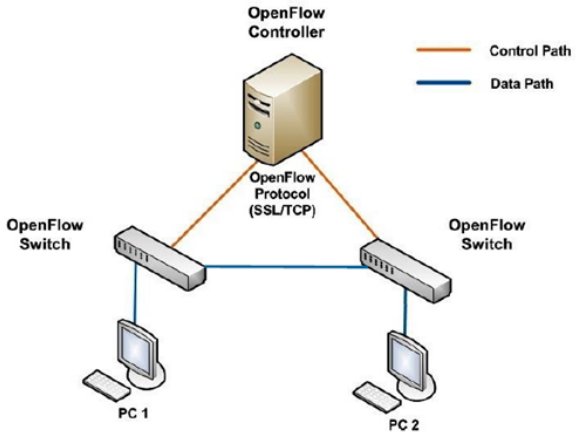

Figure 5. OpenFlow Network Architecture

2.4.3 Flow and group tables

Behavior of switches with data flow coming from different interfaces i.e. physical and virtual. The table comprises of a set of rules, where the flow of the communication data is defined. The Switch responds upon every flow according to the rules, these rules are called as flow rules.

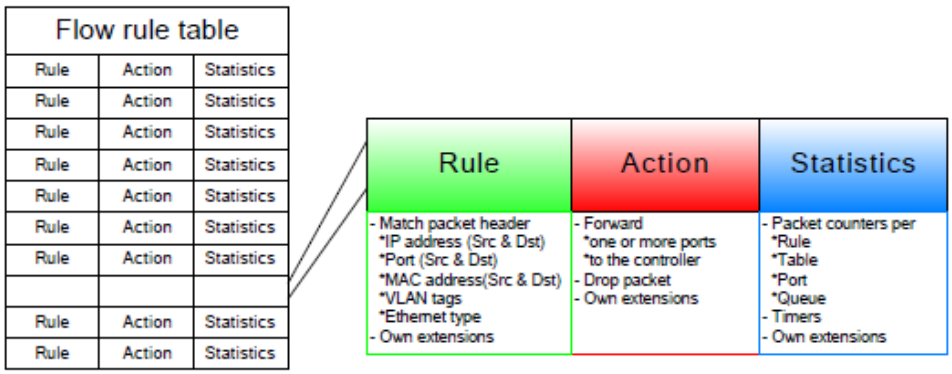

Figure 6. Example of OpenFlow

An OpenFlow switch is comprised of one or more flow tables and a group table for frame lookups and forwarding. A flow rule consists of three fields (see Figure 6)

- **Rule:** Header to match with the frames of the flows. There are many supported Eth-

ernet headers in OpenFlow specification (Godanj, Nenadić, and Romić 2016) , but as OpenFlow is designed to be extensible, custom headers can be furthermore defined. The switch only performs a bit mask match. For this reason, OpenFlow switch is open for advanced non-IP traffic.

- **Action:** Rule is matched with traffic; and defined that which action has to be performed. These actions are also open for extensions, but some basic actions are already provided in the specification. For example, forwarding to one or more ports, forward to the controller, drop the frame, and modify frame fields. In order to add the customized actions the only requirement is that the data path must have flexibility whereas providing high performance and low cost.
- **Statistics:** Always when a flow rule is matched, the switch has to update the frame counters, which shows the popularity of a specific flow. Counters are available for every table, each flow, all the ports and every queue. Furthermore a timer of last activity and initial set of the flow are maintained.

An OpenFlow channel is the connection between the switch and the controller. This channel is usually encrypted with Transport Layer Security (TLS) protocol; though, the channel can also be run by using plain Transmission Control Protocol (TCP). This provides an interface for the controller to manage and adjust flow and group tables of the OpenFlow switch. In parallel, the switch also supplies the controller with its hardware information, the connectivity status of ports, and meter statistics of every flow rule. The OpenFlow protocol predefines the communication message pattern which is used when communicating between the controller and the switch or between the switches.

OpenFlow channel is the connection between the switch and the controller.

OpenFlow protocol predefines the communication message pattern which is used when communicating between the controller and the switch or between the switches.

Network policies and services are implemented as OpenFlow applications in OpenFlow which interact with the control plane through the north-bound API (application programming interface) of the control plane. Functionalities of the control plane are implemented in an OpenFlow controller which interacts with the data plane through the OpenFlow protocol

(south-bound API). SDN applications which are OpenFlow-based are developed that use the essential network infrastructure and deploy various functions at run-time.

Therefore, control of the network traffic is transferred from the infrastructure to the administrator. Network operators will gain high levels of network control, automation and optimization with the help of SDN applications.

2.4.4 Flow types

Flows from the OpenFlow controller can be classified into two types:

- **Microflows:** microflows are beneficial when we need a small number of flows to be fixed in the switch, e.g. campus network. In this type the flow tables contain one entry per flow and exact matching is needed to perform an action.
- **Aggregated:** This type is beneficial for large networks that need a large number of flow table entries, e.g. backbone networks. One flow entry (Wildcarded) covers a large number of flows, each of which must belong to a specific group. Development of the TCP / IP architecture, the Internet has a great achievement in these 30 years, and has been one of the most broadly used technology in the world today. The fast development of the information society, the new application trends such as high-definition television, video on demand, VoIP and high-speed Internet, for the current network open a great challenge. SDN network, relying on its programmability, centralized control, resource virtualization and other advantages, will be widely used in industry.

2.5 SDN Applications

- **Internet Research:**
  In the meantime the Internet is a life network and is frequently being used, it will be difficult to do any updates or tests for new ideas that might solve the issues or problems that current Internet infrastructure faces. Through SDN we have more control, as the controller part of the network and the data traffic is separated, in other words we can say that separating the hardware part from software. These separations permit for testing new ideas about future Internet architecture before implanting it in the live

network (Hu et al. 2013).

- **Load Balancing for Application Servers:**

  The most essential requirement for enterprise networks is Load balancing. Hence it can offer high availability and scalability for the requests to a particular service. Generally this functionality of balancing loads between numerous servers is implemented by a dedicated device that is implemented in the network.

  Though with SDN an OpenFlow switch is able to deal with this functionality automatically and will allocate the traffic to different servers. However it does not scale well; thus it is conceivable to write an application that works on top of the controller that can provide a scalable and efficient load-balancing application (Wang, Butnariu, Rexford, et al. 2011). And with this application the requirement for a dedicated middle in the network will be removed.

- **Data Centers Upgrading:**

  Data centers are an important internal part of many large scale companies. Considering the example of, Google Facebook, Amazon and Yahoo have large numbers of data centers to put up the huge number of requests and response to them rapidly. These data centers are enormously expensive and complicated to keep well and run. Companies can have a cost cutting by setting up and configuring the datacenter which is allowed by SDN and OpenFlow. As data forwarding parts of the network can be managed from a central location (Hu, Hao, and Bao 2014).

3 SDN Controllers

3.1 Overview of SDN Controllers

The Software Defined Network Controllers also called SDN controller platform is essentially the "brain" of the network (Yoon and Kim 2015). The SDN controller is the application functions as the strategic control point in the Software Defined Network. In SDN network, the SDN controllers manage the flow-control to the routers or switches through the Southbound AIPs and the application the business logic through the northbound APIs (Yoon and Kim 2015). This arrangement allows SDN controllers to the intelligent network. SDN controllers are based on protocols which allow servers to direct switches how and where to send the packets. The most common protocol used with SDN Controllers is OpenFlow.

SDN controller platform mainly consists of modules (mostly pluggable) which can perform varieties of Network tasks. For example, plugged modules keep the record of all the devices in the network and the capabilities of each device. Additionally, the SDN Controller modules gather the information relating to the network statistics to enable SDN controller to manage the flow control to support intelligent networking (Benamrane, Mamoun, and Benaini 2017). Moreover, to enhance the module performance of SDN Controller Module, additional features can be inserted which support more advanced functionalities such scoring new.

The NOX, initially developed by Nacira Networks, was the first SDN controller. In 2008, the developer (acquired by VMware) deployed its NOX to the SND community. With the donation of this open source version SDN controller, Nacira Networks laid the foundation for the development of many SDN controller solutions. The company collaborated with other two application developers, namely, Google and NTT to co-develop ONIX which became the base for Nacira/VMware) Controllers (Pol et al. 2011). Today, a large number of open source SDN Controllers are being developed such as POX and Beacon. The SDN Controller market has attracted many companies including Cisco, IBM, HP, Juniper, and VMware. Cisco, HP, and IBM Controllers are all shifted from the Beacon Controller, and the companies are now moving toward the OpenDaylight.

3.2 SDN Controller Protocols

Particularly, there are two main protocols used by SDN Controllers communicate with routers and switches in the network. These include the OpenFlow and the Open Virtual Switch Database (OVSDB) (Benamrane, Mamoun, and Benaini 2017). Apart from the OpenFlow and OVSDB Protocols, SDN Controllers can also use NetConf and YANG Protocols. Additionally, there are much more SDN Controller Protocols which are being developed and customized to function in SDN environment. An excellent example of such Protocols is the IETF's Interface to the Routing System (i2rs) which is designed to provide high scalability and routing efficiency (Benamrane, Mamoun, and Benaini 2017).

3.2.1 The OpenFlow Protocol

The OpenFlow Protocol is the first standard SDN Control Protocol. It defines the open communication standards which guide how the SDN Controllers function in collaboration with the "forward plane" (such as routers and switches) and modify the necessary changes on the network. As such, the OpenFlow Protocol provides businesses with opportunities for implementing their changing network needs and achieve high controls over the system (Benamrane, Mamoun, and Benaini 2017). The Protocol is primarily the first and most widely used SDN Controller Protocol which was initially designed as an open interface to control the manner in which packets are routed in a SDN-based network (McKeown et al. 2008). In a traditional network infrastructure, the data path and the control logic communicate through the internal proprietary bus and are co-located on the same device. However, with OpenFlow Protocol, the arrangement is quite different (McKeown et al. 2008). The OpenFlow Protocol moves the control path or logic to a commodity PC (or external Controllers) (McKeown et al. 2008). In this case, OpenFlow provides the communication for the Controller to "talk" to the data path over the network.

During the above process, the routing or forwarding directives are abstracted as "flow-entries" by the OpenFlow Protocol. A flow entry consists of a list of actions, a bit pattern, and a set of counters. Each flow is a packet set which matches the given pattern (Pol et al. 2011). Each time a packet arrives at a router or a switch, the device checks the flow-table

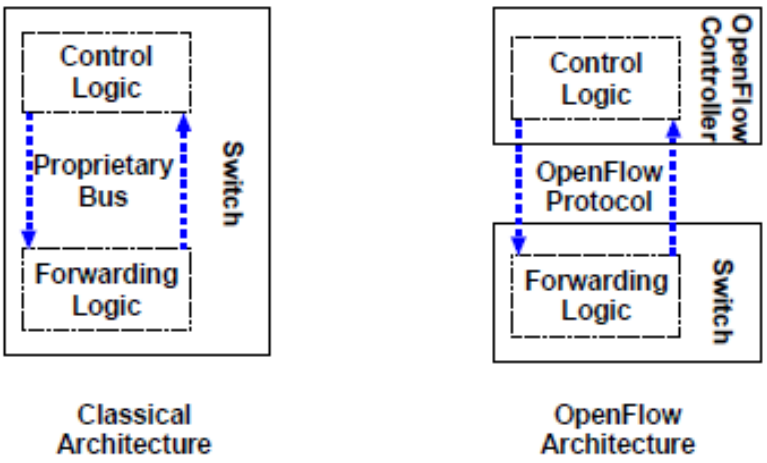

Figure 7. The control path is moved to an external controller by the OpenFlow Protocol

(a collection of flow entries on the device) to find packet and then performs the relevant action or "action set." However, some packets may fail to match any existing bit patterns in the flow-table, for example, when the packet does not have its corresponding entry. In such a case, the packet is queued, and OpenFlow Protocol sends a new entry to the external Controller which will then respond with "flow-modification message." The message makes a new rule (meant for handling the queued packet) to be added to the flow-table [5]. Next, the new flow-entry is cached in the flow-table for the handling of the succeeding packets in the flow-table.

By its architecture, OpenFlow employs the idea that the modern routers/switches logically implement flow-tables and flow-entries (Benamrane, Mamoun, and Benaini 2017; Pol et al. 2011). Accordingly, the network devices can be made to comply with OpenFlow by upgrading the firmware. So, there is no need for additional hardware support.

Benefits of OpenFlow Protocols

- The OpenFlow Protocol provides flexibility in network usage, operation, and selling. The software which governs the OpenFlow-based SDN controller protocol can be written by service providers or application developers within the enterprise using ordinary software environment.

- Since the operator can implement the features they desire for the software which they use, OpenFlow Protocol promotes rapid service introduction through customization. Therefore, the operator does not need to wait for a vendor. In this way, OpenFlow protocol also provides time-saving the advantages.
- OpenFlow protocol results in few errors, and therefore, lowers the operation costs. Additionally, since the occurrences of errors are minimized, the network has low downtime especially due to automatic configuration supported by OpenFlow Control protocol.
- The OpenFlow can be readily integrated into computing to support the management and maintenance of network resources.
- Many organizations find the OpenFlow protocol useful for aligning the network with business objectives.
- It also fosters open multi-vendor market offering a standard means for conveying flow-table operation.

3.2.2 The Open Virtual Switch Database (OVSDB)

The Open Virtual Switch Database is management protocol which supports the communication between the network devices in SDN system. This Protocol guides how the SDN Controllers and the network devices exchange statistical and control information (Bittman et al. 2013). For example, for an Open Virtual Switch Database client machine on a software-defined network Controller to communicate with Open Virtual Switch Database Server on a network device, a connection has to be established between the device and the Controller. This connection establishment is accomplished through a configuration process (Kirkpatrick 2013).

First, the information about the SDN Controller has to be specified (using the IP address corresponding to the controller). Next, the connection protocol and the port over which the connection will occur must be defined. After configuration, the connection is established between the SDN Controller and the management port of the device.

Usually, the OVSDB server will store and maintain the Open Virtual Switch Database database schema defined by the hardware. The OVSDB contains the statistical and control informa-

tion provided by the Open Virtual Switch Database client on the SDN controllers and the device. In the scheme, the information is stored in various tables, and the addition, deletion, or modification of any data on the schema is done by OVSDB client which continually monitors the schema ("OVSDB" 2017). In a Juniper Network, for instance, schema provides the means for information exchange between the SDN controllers and the devices over the network. Thus, the OVSDB is a virtual switch which offers automation and supports standard management Protocols and Interfaces such as NetFlow. Additionally, the protocol is also necessary for distribution across multiple physical servers.
In an OVSDB implementation, a switch daemon and a database server are employed. This enables the OVSDB protocol used for controlling cluster along with other SDN Controllers and managers to send configuration data to the switch database server. Moreover, within an Open vSwitch implementation, the OVSDB can be used by IT experts in determining the number of virtual bridges. In this way, OVSDB supports the creation, configuration, and deletion of ports and tunnels from a bridge. Moreover, OVSDB allows engineers to create and perform other modifications on queues ("OVSDB" 2017).

3.2.3 Interface to the Routing System (i2rs)

The i2rs is SDN Controller protocol developed by the Internet Engineering Task Force (IETF) with the objective of offering an efficient routing operation. The design of the i2rs architecture is mainly to enhance the network control and allow applications to build on top of the system (Bittman et al. 2013).

The most important architectural feature of i2rs protocol is the simplicity. Although it is often challenging to maintain simplicity particularly when the Protocol has to support access to multiple data types in a networking system, the i2rs has been made to successfully achieve simplicity with the ability to support different data types stored on a variety of networking devices (Benamrane, Mamoun, and Benaini 2017).
Moreover, the i2rs is easily extensible and scalable which makes it ideal for use in high-performance routing systems where a great number of modifications or operational changes are frequent.

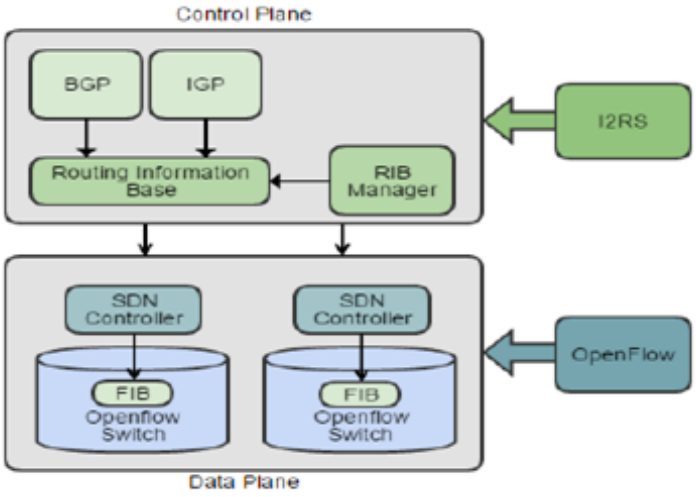

Figure 8. i2rs and OpenFlow Interactions

Major components of i2rs:

Architecturally, i2rs has five main elements for achieving scalability through filterable data access. The five fundamental i2rs elements include the network application, i2rs server, client, agent, routing element.

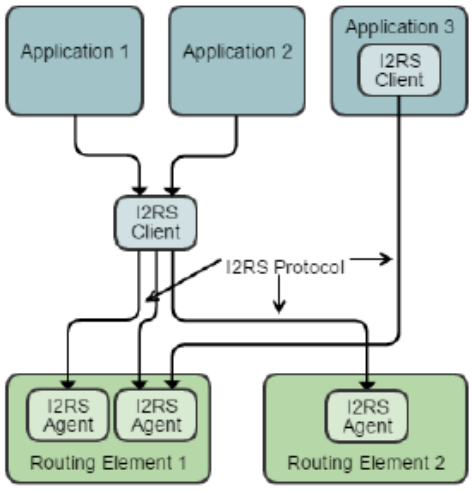

Figure 9. Interaction between i2rs components

The network application: A piece of software which is system oriented with the objective of manipulating or accessing network states. The network application achieves this goal by the i2rs client.

Client: The i2rs Client is an entity which implements the i2rs protocol and initiates communication between i2rs and the agents to modify the network information. Primarily, an i2rs client could be a piece of code or an external i2rs library (Hazboun 2016).
Server: The i2rs server consists of a set of function to support information access and modification based on the usage policy. The i2rs servers are specified using a particular data model, for instance, BGP or MPLS services. Agent: The i2rs agent is mainly an entity which interacts with the subsystem's routing element to access and modify the states of the element. The i2rs Agent offers this functionality as a service provided by i2rs to requesting agents.
Routing element: An i2rs routing element refers to a device which implements some functions related to routing. For example, the path element could involve a traditional router implementing SDN Controller's logical control plane.
However, no matter how a particular path element is implemented, the behavior of an i2rs Agent should remain unaffected. For instance, in a system which is physically distributed, needs to continue supporting the accessibility of data from the entire element.
Moreover, if multiple i2rs Agents resides within a routing element, the i2rs Agents will ensure simplicity, which is a critical design goal of i2rs, by serving separate sets of information (Hazboun 2016).
Essentially, the type of protocol supported by SDN controller has a significant effect on the overall network architecture. For instance, the IETF's Interface to the Routing System (i2rs) splits the process of making decisions by leveraging the routing protocol. The splitting provides the i2rs protocol with the capability of performing distributed routing to support the modification of routing decisions by applications (Benamrane, Mamoun, and Benaini 2017). However, in OpenFlow protocol, on the other hand, the packet forwarding decision-making tends to be completely centralized.

3.2.4 OF-Config and Netconf Protocols

Over the time, many paths to Software-Designed Network have been in use. In many networking environments, paths to SDN is created by first placing the network management in a centrally located controller (eg. "OF-Config-OpenFlow-Configuration-and-Management-

Protocol" 2017).“The Control strategy of the physical network then unlinked to allow the centralized controller to route flows between the nodes in the network by use of OpenFlow as the Southbound protocol" (eg. “OF-Config-OpenFlow-Configuration-and-Management-Protocol" 2017).

Although this Protocol is often useful for management of flows and controlling of the manner in which each packet is forwarded to its destination, the OpenFlow does not offer the necessary management and configuration for port allocation and Internet Protocol (IP) address assignment. This means that in as much the OpenFlow Protocol is critical for managing and forwarding the packets, an alternative protocol may be required to manage configurations and assign IP addresses. In this case, the OpenFlow Control Protocol becomes the ideal solution.

In most traditional networking systems, vendor resorted to using branded management and configuration approaches. Some of these methods depended on Simple Network Management Protocol for monitoring devices as well as for product configuration tasks. Additionally, in other systems, command lines were employed for configuring each network device on the system.

However, SDN enables Networking engineers to view each network component and set up policies for traffic management across the matrix of the devices. In this case, the OpenFlow will not provide specifications for the control protocol or switch configuration databases. Instead, the OpenFlow will only define the packet flow operation.

OpenFlow Configuration Protocol (also called OF-Config) supports standard approach for the management and configuration of switches by establishing the relationship between the network switches and Controllers. It allows the network administrators to choose switches from their preferred vendors to select the most appropriate devices for particular network location. Additionally, network engineers can use the OF-config to set communication parameters between switches and controllers.

OF-Config Basics:

Open Networking Foundation (ONF), the developer of OF-Config designed OpenFlow Configuration Protocol for use with all OpenFlow protocols (“Ethernet OAM enabled OpenFlow Controller" 2011)."[Thus], OF-Config supports all OpenFlow implementations including virtual and physical switches" (“Ethernet OAM enabled OpenFlow Controller" 2011).As

SDN Controller protocol, OpenFlow Configuration Protocol has successfully addressed various controller-switch management components (eg. “Ethernet OAM enabled OpenFlow Controller” 2011).Some of these include:

- The OpenFlow Logical Switches
- OF-Config Point – OpenFlow Configuration issues command called "OF-Config commands".
- OpenFlow Capable switches "[which] include both virtual and physical devices to support switching" (“Ethernet OAM enabled OpenFlow Controller” 2011).
- OpenFlow capable Switches has a number of queues and ports (“Ethernet OAM enabled OpenFlow Controller” 2011).

Mostly, the OF-Conf Point is often located with the OpenFlow controller within the same workstation or server. Alternatively, the both OpenFlow and the OF-Config can also be found in a traditional network management product. However, whichever way, the configuration points can perform multiple management of multiple OpenFlow Capable switches. Conversely, in SDN environment, a single configuration point can manage many Capable Switches. OF-Config is the OpenFlow Management and Configuration Protocol’s version 1.1. (eg. “OF-Config-OpenFlow-Configuration-and-Management-Protocol” 2017).
Additionally, Configuration Flow logical switches which reside within an OpenFlow Capable Switch can communicate with Configuration Point. For example, the control point supplies logical switches with port numbers and IP addresses of the OpenFlow Controllers which manages the flow of packets through the switch. Moreover, the control point also specifies the protocol (TLS or TCP) to be used for the communication between the Controllers and the switches. After determining the communication standard, it then performs configuration to identify, specify and send the certificate to allow the Controllers and the Switches to communicate. Primarily, within the same OpenFlow Capable Switch, each OpenFlow logical switch operates separately and is not dependent on the other logical switches.
The implementation of OF-config in a switch requires that the internal configuration database of the switch must be modified. Moreover, the NetConf protocol must also be implemented to allow switches to communicate with Configuration Points. Ideally, the Netconf employs XML encoding system for the configuration of Protocol data and messages (eg.

"OF-Config-OpenFlow-Configuration-and-Management-Protocol" 2017).Usually, the configuration of data is retrieved from and sent to switches through a process called remote procedure calls. "The Netcofig protocol can send/retrieve partial/full configuration description" (eg. "OF-Config-OpenFlow-Configuration-and-Management-Protocol" 2017). At the same time, the Protocol has the capabilities of conveying asynchronous notifications from the switch. And because of its extensibility, the Netconfig provides support to the OF-Config as more capabilities are added.

3.3 Open Source SDN Controllers

These types of Controller versions enables the testing of the applications. Although the open source SDN Controllers even supports the promotion of Network Virtualization and NTF, most users are not confident in using the open source software-based networking. According to (Pol et al. 2011), most companies prefer commercial SDN Controllers to the Open Source Version for various reasons. For example, in most organizations which are operating super-sized networks in which high-performance speed is required amidst huge amounts of data to be handled, commercial SDN Controllers is mainly used because of the perception that Commercial versions are simpler and easy to manage. The following are some of the most common open source SDN Controllers

3.3.1 OpenDaylight Open-Source SDN Controller

The OpenDaylight Platform was hosted by Linux foundation as an open source SDN project. The aim of hosting the Controller was to enhance the SDN by providing an industry supported and community-led framework for the OpenDaylight platform. Moreover, under the Linux foundation, the OpenDaylight Platform comprises the OpenFlow Protocol and has the capabilities to support open SDN Standards. This open source SDN Controller can also support modular controller framework and is often deployed in various production network environment (Medved et al. 2014).Moreover, the OpenDaylight Controller has being employed by various applications for the collection the network data, conducting analytics by executing algorithms and creating rules within the system (eg. "OVSDB" 2017).

The OpenDaylight platform is implemented entirely in software and is stored in its Java Virtual Machine. In this way, OpenDaylight SDN Controller can be deployed on operating systems and hardware which supports Java. However, for best performance, it is necessary to use OpenDaylight SDN Controllers with a Java Virtual Machine (JVM) and a recent Linux Distribution (eg. “OVSDB” 2017).In regards to its releases, "Hydrogen" was the first software code version for the OpenDaylight platform. Hydrogen features three different editions for users. These include (eg. “opendaylight Controller” 2017).

- Base Edition
- Service Provider Edition
- Virtualization Edition

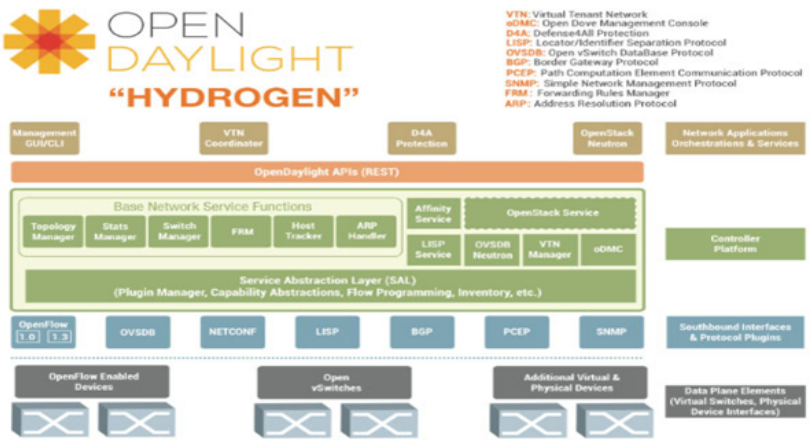

Figure 10. OpenDaylight Infrastructure

Helium was the second code release for OpenDaylight SDN Controller which offered a new user interface as well as a more customizable and simplified installation process due to the use of a special container called “Apache Karaf container.” (Hares and White 2013).Additionally, Helium version code release also has a higher integration with OpenStack as well as features such as Distributed Virtual Router, Load Balancing services, and Security Group. The third launch, Lithium, was realized in 2015 leading to the repositioning of the OpenDaylight Controller to the OpenDaylight platform. This version was later followed by Beryllium.

3.4 OpenContrail SDN Controller

Primarily, OpenContrail SDN Controller product appeared in December 2012 when Juniper Network acquired Contrail and began to build SDN capabilities. This Open Source SDN Controller forms part of Apache version 2.0 which is useful for enabling network virtualization. Developed by Jupiter Networks, the Contrail Controller is a product for cloud network automation that employs the SDN-based technology to orchestrate the creation of highly scalable virtual networks (Hares and White 2013).

Mostly, OpenContrail SDN Controller integrates physical routers, switches, and scale-out framework to improve the infrastructure beyond the data center of cloud boundaries. In this way, OpenContrail SDN Controller provides workload mobility in a hybrid environment. It functions alongside virtual network routers to be located on hypervisor hosts and is also useful as a network platform for a cloud computing infrastructure (Cui, Yu, and Yan 2016).Additionally, the source code of OpenContrail SDN Controller is hosted across many software repositories.

Moreover, OpenContrail SDN Controller, as SDN Controller platform which uses OpenStack distribution for automation and orchestration aims to improve availability, security, and flexibility in networking performance. For most companies running big networks, Open Contrail is the Open Source SDN Controller of choice for various reasons. First, OpenContrail SDN Controller is the only SDN Controller which can support nearly all necessary features for virtualizing network and mobility elements ("OpenContrail as SDN controller" 2017). Secondly, this open source SDN Controller is readily integrated into existing networking environment to provide capabilities for faster troubleshooting and committing of bugs since the Controller is based on highly developed MP-BGP technology ("OpenContrail as SDN controller" 2017).

3.4.1 Floodlight Open SDN Controller

The Floodlight Controller to function alongside the OpenFlow Protocol for orchestrating traffic flows in SDN environment. It is an Open Source SDN Controller which was initially created by "Big Switch Networks" for use by companies and individuals "who intend to improve the functionality of OpenFlow Protocol operated in SDN environment" (Wallner and

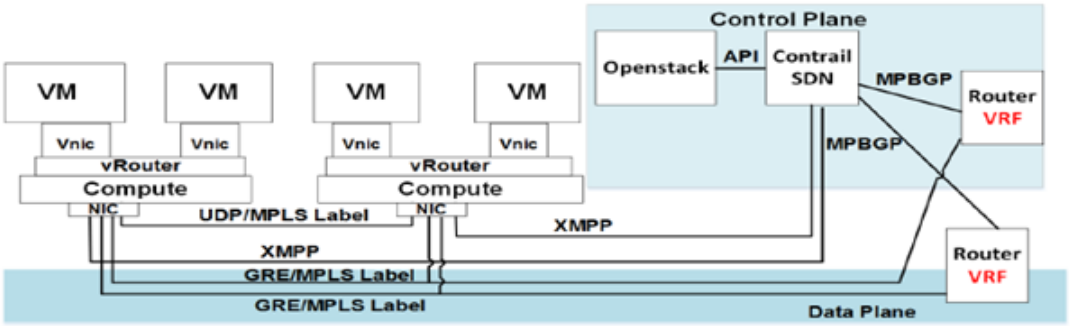

Figure 11. The use of High-Level OpenContrail SDN Controller in combination with OpenStack and Containers in Networking

Cannistra 2013).The Floodlight Controller maintains and controls all the rules within the network in addition to providing the instructions necessary to guide the fundamental infrastructure on how to handle the data traffic within the network (Cui, Yu, and Yan 2016).Therefore, the Floodlight Controller provides users with capabilities necessary for better controlling of the systems through simple adaption approach to the changing needs (Wallner and Cannistra 2013).

Additionally, this Controller as an Open Source SDN Controller offers many advantages to developers. For example, it allows developers to develop applications through software adaptation and writing in Java. Moreover, the Floodlight Controller include state transfer application programs with which Application Developers can readily program interface with products. The Floodlight website further provides coding examples to help programmers with building products.

Fundamentally, Floodlight Controller Open Source is tested with both Virtual and Physical OpenFlow Compatible switches. The Controller can function in different network environments and can be integrated to an existing networking system to support systems in which OpenFlow Compatible Switches are connected via Non-OpenFlow/traditional switches.

3.4.2 Ryu OpenFlow Controller

Ryu is an Open Source SDN Controller which is designed to provide various software components for use in SDN applications. The Ryu OpenFlow Controller supports the creation

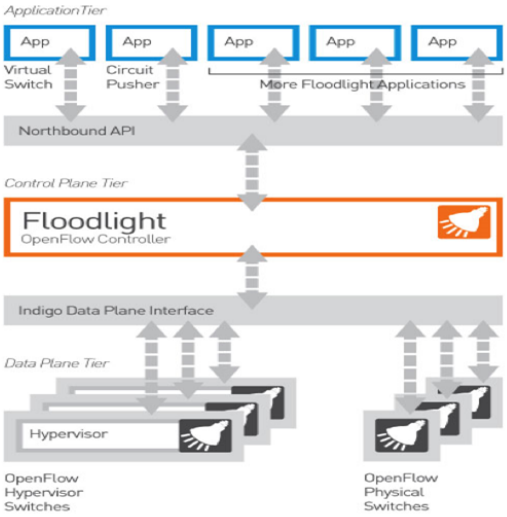

Figure 12. The working of Floodlight Controller in a SDN-environment

of new networks management and controlling of applications. According to (eg. “Ryu Controller” 2017). Ryu is notably advantageous because of its ability to support different protocols such as Netconf, OpenFlow, and OF-config for managing the devices in the network. With these protocols, Ryu allows users to control the "network gear" such as routers and switches according to the system demands. It provides software components with well-defined APIs which makes it simple for organizations to customize the deployment to meet the particular needs of the company. For example, the organization’s developers can easily and quickly implement new or adapt existing components to ensure that the underlying network is consistent with the changing demands of the key applications (eg. “Ryu Controller” 2017).

3.4.3 FlowVisor OpenFlow Controller

The FlowVisor is a special purpose SDN Controller which operates between multiple OpenFlow Controllers and OpenFlow Switches. This Controller a useful tool for network virtualization which it performs by converting physical network into multiple logical ones through network division (Zhong et al. 2016)With FlowVisor Controller each Controller can be made to touch only the resources and switches assigned to the particular Controller. Additionally, the FlowVisor Controller supports the partition of flow table and bandwidth resources on

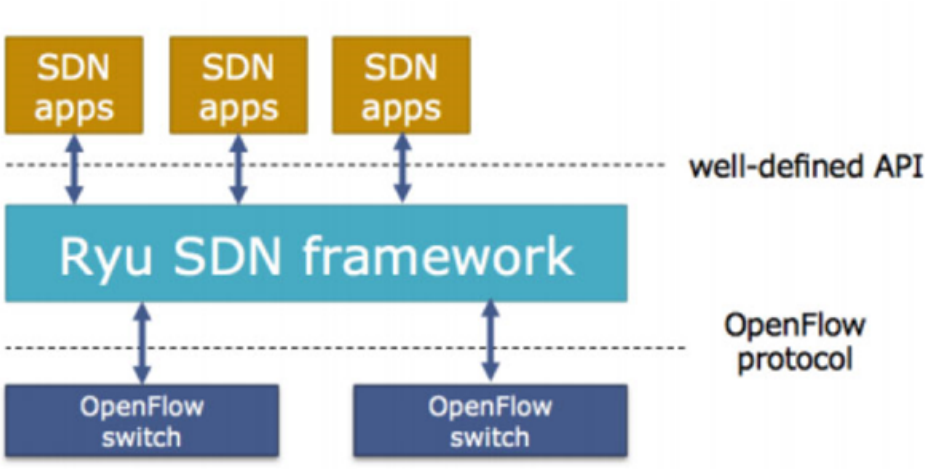

Figure 13. How the Ryu OpenFlow Controller fits in SDN Environment

each switch after which each partition can be allocated to the particular Controller. Notably, the virtualization using FlowVisor Controller essentially works similarly as the virtualization layer in a computer. The Controller sits between the underlying software and the physical hardware which it controls (as shown in the figure below). And similar to the case of an operating system, the FlowVisor controls the network components by use of instruction sets through OpenFlow SDN Controller protocol. The FlowVisor hosts many OpenFlow controllers, but for each slice, it maintains one Controller. In this way, FlowVisor ensures that each Controller observes and controls its slice. At the same time, FlowVisor keeps each slice isolated from other Controllers other than the one which controls it (Zhong et al. 2016).

Therefore, FlowVisor Controller in SDN environment performs the following:

- Defining slice as a set of low which runs on switch's topology
- Resides between each SDN Controller Protocol –OpenFlow for this case –to ensure that guest Controllers observe and control only the specific switch it is ought to control.
- Partitions the link bandwidth by setting data rate limit for each set of flow which makes up a slice.
- Partitioning the Flow-Table in each switch by keeping track of every flow-entry belonging to each guest controller.

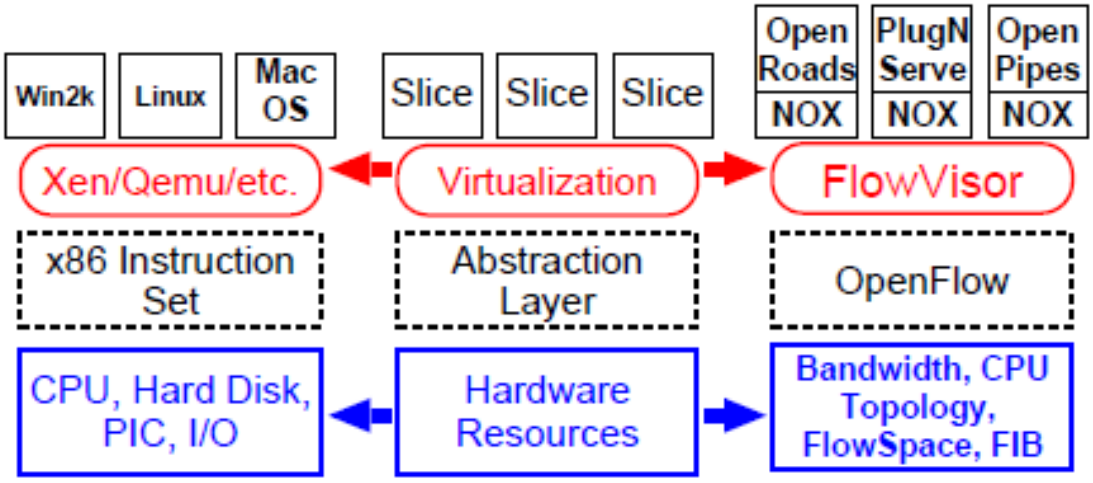

Figure 14. FlowVisor resides between the underlying software and the physical hardware which it controls

Benefits of using SDN Controllers

- SDN Controllers are used with OpenStack for network virtualization. In this integration, SDN Controller important in programming switches through OpenFlow Protocol while OpenStack does the required orchestration.
- SDN Controller provides network engineers with better ways of managing packet forwarding in a virtual environment. For example, the with SDN controller protocols such as the OpenFlow Protocol, forwarding packets to physical devices and software within the network is a simpler process.
- Reduction of system downtime –SDN controllers allows most of the physical networking devices to be virtualized. Therefore, upgrading can be performed on selected pieces. Additionally, recovery from failures due to upgrading tasks is fast. (eg. “Ryu Controller” 2017).
- Extensibility: various SDN controllers such as I2RS are designed for extensibility through providing a platform for easy upgrading

4 Securing SDN Controllers

4.1 Overview

Software-Defined Networking provides operators with high flexibility in controlling the network. Through SDN capabilities, managing the network has shifted from codifying functionalities in regards to low-level device configurations to building an application which facilitates the debugging and management of the system (Kreutz, Ramos, and Verissimo 2013), Additionally, SDN offers new techniques for solving long-standing problems such as routing challenges in the network while simultaneously permitting the application of security features including access control. Such capabilities are provided in SDN environment because SDN allows for the separation of the complexity of state distribution from network specification. However, the security of SDN system remains an open issue. Initially, SDN deployments used to be small in size where the SDN environment consisted of a single controller for use-case testing and research (Scott-Hayward 2015).

However,currently enterprises requiring SDN deployment demand, multiple controllers. With the increasing interfaces and components for evolved SDN implementation, security challenges associated with SDN Controller design have increased (Scott-Hayward 2015).

This chapter focuses on the security of SDN Controllers. The discussion begins by describing several threat vectors which may enable for the exploitation of the vulnerabilities of SDN Controllers. Next, the study focuses on designing a dependable Controller platform including the requirements for a secure, resilient, and robust SDN Controller. The chapter also analyzes the state-of-the-art open-source SDN Controllers in as far as the security designs of these Controllers are concerned. The chapter closes by making recommendations for security improvements for future SDN Controllers. The discussion highlights the existing gap between the actual security level of the current SDN Controller design and the potential security solutions.

4.2 Threat Vectors

SDN has two features which are a possible sources challenges for operators who are less prepared and attractive for malicious users (Kreutz, Ramos, and Verissimo 2013). The first property is the ability to control the network through software which could be subject bugs and other security vulnerabilities. The other feature is the centralization of "network intelligence" in controllers (Kreutz, Ramos, and Verissimo 2013) that provides abilities for any party who has accessibility to the servers which host the control software with capabilities to manage the entire network. In this way, SDN systems are associated with threats of different nature. Thus the security of SDN and SDN Controllers need to be addressed differently. The following section describes six security threats related to SDN and SDN Controllers as well as suggested solution criteria.

4.2.1 Faked or forged traffic flows

In some cases, malicious users or faulty devices fake or forge traffic which they use to attack SDN Controllers and switches. An example is where attackers use network elements such as PCs, servers, or switches to launch a Denial of Service attack against Controller resources and OpenFlow switches. Although a simple authentication mechanism could be useful, it can be quite difficult to control the problem if the attacker the system allows the attacker to monitor the application server in which the details of many users are stored.
Possible solution approach: The best way to mitigate faked or forged traffics on SDN system is to employ a detection system to help in identifying any abnormal flows.

4.2.2 Attacks on vulnerabilities in switches

In this case, attackers use a single switch to slow-down or drop packets in the network. Alternatively, attackers could inject forged requests or traffic with the aim of overloading neighboring switches or SDN Controller. Possible solution approach: These kinds of threats can be mitigated using software attraction mechanisms. Such techniques include automatic trust management solutions for software components (Yan and Prehofer 2011).

4.2.3 Attacks on control plane communications

In some cases, attackers use control plane communication to compromise the controller-device link. This vulnerability could allow the attacker to leak data while the standard production traffic flows by creating virtual black hole network, for instance, through the use OpenFlow-based slicing approach (Scott-Hayward, O'Callaghan, and Sezer 2013).
Solution criteria: One of the most relevant solution criteria for attacks on control plane communication is for the network operators to "use oligarchic trust models with multiple trust-anchor certification authorities [such as] one per subdomain or Controller instance" (Kreutz, Ramos, and Verissimo 2013). Alternatively, the SDN Controller could be secured by using threshold cryptography across Controllers to secure the communication.

4.2.4 Attack on Controller vulnerabilities

These are possibly the most severe threats to SDN systems where a malicious or faulty Controller compromises the entire network. In this case, the application of the common intrusion detection systems may be insufficient due to the difficulty of finding the exact event-combination which are responsible for triggering a particular behavior and classify the behavior as malicious. Moreover, since SDN Controllers, in most cases, only provide the abstractions which translate to the issuing of command configurations to the fundamental structure, malicious applications can compromise security in any way it pleases (Kreutz, Ramos, and Verissimo 2013). Solution approach: various methods to mitigate an attack on Controllers. Examples of typical techniques include:

- Replication (detection, removal, or masking of the unusual behavior)
- Employing diversity of programming language, Controllers, software image or SDN Controller Protocols.
- Recovery of the system including periodic refreshing of the system to maintain a clean and reliable state.

4.2.5 Lack of mechanisms for ensuring security between the management applications and Controllers

This vulnerability often results when Controllers and applications are unable to establish trusted relationships between them. Solution approach: The most important solution criteria is to employ mechanisms which can create autonomic trust management to verify the trust of the application during its lifetime.

4.2.6 Attacks on vulnerabilities in administrative stations

The attacks, though most common with traditional networks, are also applicable to SDN systems to get access to the system Controller (Scott-Hayward 2015) Vulnerabilities at the central stations are more dangers especially because it becomes easy for malicious users or applications to cause more damage from a single location. Solution approach: At the administrator stations, operators can protocols which require double credential verifications can be a useful solution criteria (Kreutz, Ramos, and Verissimo 2013).

It is notable from the six threats that with SDN, the attack surface is more augmented in comparison to traditional networks. As such, straight from the design phase, various security mechanisms, as well as mitigation techniques, must be put in place to secure SDN Controllers (Kreutz, Ramos, and Verissimo 2013). In particular, an efficient design for SDN Controllers should address the different security issues to ensure that the network is safe and dependable.

4.3 Secure Controller Design

The Network Operating System (NOS) or Controller is essentially at the center of SDN. Although many types of Controllers (such as described by (Kreutz et al. 2015)), have been deployed in the past since the development of OpenFlow protocol, the focus of most of these early Controllers was more on developing support for OpenFlow API (Scott-Hayward, O'Callaghan, and Sezer 2013). And to achieve such design goals, the non-SDN or traditional network design began to focus on scalability and performance (Kreutz et al. 2015). In particular, the scalability in non-SDN systems was to produce a controller which is supportive

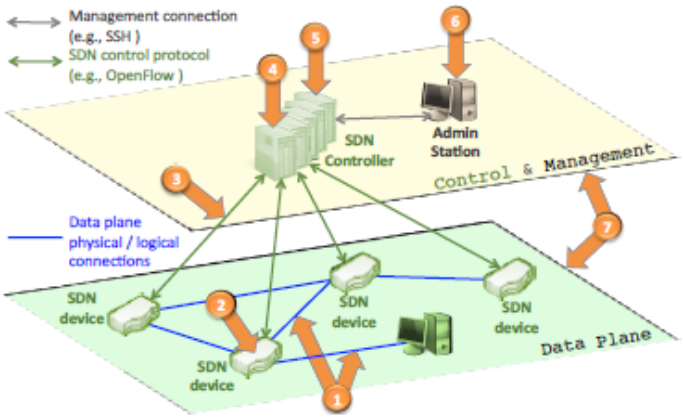

Figure 15. SDN Main Threat Vectors

to increasing number of hosts or switches. On the other hand, performance enhancement aimed to enable the Controller to achieve low latency but high throughput in regards to control event processing.

Concerning security and network reliability, early designs were concentrated on the centralized controller as a single failure point in the system (Scott-Hayward, O'Callaghan, and Sezer 2013). However, with a centralized controller, attackers can quickly take control of the network by hi-jacking the device in which the control decision-making occurs. This situation is significant because central controllers allow control decision-making to be held in one device. Additionally, the vulnerability of the central Controller is high in regards to denial of service (Scott-Hayward, O'Callaghan, and Sezer 2013).

However, as SDN evolves, (Scott-Hayward 2015) argues that the vulnerability of the central controller is no longer a challenge for Controller designers to address in isolation. Instead, a comparative analysis based on security features of five primary Controllers, namely, ONOS, OpenDaylight, ROSEMARY, Ryu, and SE-Floodlight shows that modern SDN Controller designs try to deliver security alongside resilience and robustness. In that regard, (Scott-Hayward 2015) identifies several approaches to developing a secure, robust, and resilient

SDN Controller. These strategies include:

- Control Process (Application) Isolation
- Implementation of Policy Conflict Resolution
- Multiple Controller Instances
- Secure Storage

4.3.1 Control Process (Application) Isolation

In order to provide a secure Controller, designers often separate the applications processes which are running on the controller. This separation, also called "process isolation," is critical to support authentication of each application, to apply levels of authentication dependent on trust, to provide logical segmentation, and to provide logical segmentation. Additionally, isolating the applications should offer resilience to the controller so that an error or a failure on a single application does not compromise the Controller (Scott-Hayward, O'Callaghan, and Sezer 2013). This technique has been used as a security mechanism in all the five controllers analyzed by (Scott-Hayward 2015).

For example, in ROSEMARY SDN Controller, the objective of preventing malicious applications is achieved through the use of micro-NOS architecture (Scott-Hayward 2015). This is accomplished by designing each OpenFlow application to run within an independent ROSEMARY instance. In this way, the application is sandboxed to protect the control layer from malicious operations or vulnerabilities. In SE-Floodlight Controllers, on the other hand, Control Process Isolation is achieved by use of northbound API to separate control processes from applications. Thus, in both ROSEMARY and SE-Floodlight Controllers, the controller is protected from buggy or malicious applications through a containerized or sandboxed approach in combination with reliable identification and authorization services.(Scott-Hayward 2015).

4.3.2 Implementation of Policy Conflict Resolution

In SDN systems, issues of policy conflicts often emerge when a controller receives incompatible flow rules from multiple applications. In regards to this problem,(Scott-Hayward, O'Callaghan, and Sezer 2013) suggests several solutions. However, in many SDN controllers, the issue of conflict resolution remain unaddressed. (Scott-Hayward 2015), finds that of the five controllers analyzed for security features, only ES-Floodlight and ONOS have implemented the Policy Conflict Resolution.

In SE-Floodlight, for instance, an algorithm called Rule-chain Conflict Analysis (RCA) has been used by SEK for detecting the occurrence of any conflicts between the flow rule in the flow table and a new flow-rule (Porras et al. 2015). However, in ONOS, (Botelho et al. 2014), applications define their network requirements as "intents" which are translated by ONOS according to the system configuration.

4.3.3 Multiple Controller Instances

In SDN, multiple controllers were first required to solve the security issues linked to centralized controllers. Solutions have since evolved to distributed controller design from simple controller replication techniques. However, with the distributed design comes the issues of consistency, timing, coordination, and synchronization (Scott-Hayward, O'Callaghan, and Sezer 2013).

In the assessment performed by (Scott-Hayward 2015), both SE-Floodlight and ROSEMARY do not consider multiple controller instances. On the contrary, ONOS has been designed for fault tolerance and through distributed architecture where an ONOS cluster can be formed by linking some ONOS instances. "Each instance is an exclusive cluster of a set of switches" (Scott-Hayward 2015). Thus, the connection of multiple ONOS instances is critical for ensuring that a new message is elected for each affected switch by remaining ONOS instances in case of failure of one ONOS instance.

4.3.4 Secure Storage

Since critical network information about the state of the system is contained within controllers, the controllers must be secured. This protection required the application of various security practices. For example, in some controllers, secure storage is achieved by enforcing a default permission on the log file. These log files ensure that owners both write and read privileges but read-only and not write to others. Moreover, more security measures are employed to individual controllers. In ROSEMARY, for instance, the controlling of the internal storage is achieved by limiting the ability of applications to perform certain operations in the internal storage. At the same time, data structures have privilege set with which accessibility to internal storage is controlled. (Scott-Hayward 2015).

4.4 Secure Controller Interfaces

In SDN Controllers, there are three potential interfaces. These include D-CPI, I-CPI, and A-CPI for Data-Controller, Intermediate-Controller, and Application Controller respectively. However, of these three, the D-CPI is the only standardized interface (eg. "Open Flow Switch Specification" 2014) In the OpenFlow Switch Specification, the use of Transport Layer Security (TLS) is recommended although this is optional. Besides, Graphical User Interface (GUI) is another commonly used interface to the controller. GUI is often provided to simplify the management and to offer network device information, a network topology viewer, and flow table details (Scott-Hayward 2015). For all these interfaces, whether D-CPI, ICPI, A-CPI, TLS, or GUI, sensitive communication has to be protected.

4.4.1 Secure Control Layer Communication

In most modern SDN-controllers, TLS is supported across D-CPI. Secure communication with SSL/TLS defines the authentication of the communicating parties by use of X.509 certificate with subsequent data encryption between the parties across the interface of communication (Scott-Hayward 2015). In SE-Floodlight, for instance, secure communication is supported across A-CPI. On the other hand, both OpenDaylight and Ryu Controllers support SSL/TLS although other Controllers such as ROSEMARY and ONOS do not support

SSL/TLS.

However, according to (Liyanage, Ylianttila, and Gurtov 2014), Controllers should be provided with additional protection against TCP Synchronization, IP spoofing, and DoS especially when the SSL/TLS-based communication is unable to protect the SDN Controller from IP-based attacks on control channels. In such as case, a protocol equivalent to TLS should be used for protecting the communication between the control layer and the application/data layers to prevent interference with message exchanges (Liyanage, Ylianttila, and Gurtov 2014). Additionally, key or certificate materials must be appropriately managed to ensure that the security of Public Key Infrastructure (PKI) is underpinned (Scott-Hayward 2015).

4.4.2 GUI/REST API Security

In SDN controllers, there is always a possibility of manipulating the network state through controller Graphical User Interface. Accordingly, it essential to protect GUI by requiring authorization or authentication from users before accessing the controller via the GUI. For example, in OpenDayLight Controller, security is provided in which a username or password or both, are required to log into the controller Graphical User Interface. Similarly, in ONOS, GUI API Security is applied but without the requirement for authorization or authentication to access GUI. Instead, ONOS requires the IP address of the machine in which the controller is hosted (Scott-Hayward 2015).

4.5 Controller Security Services

Apart from protecting the control platform and interfaces, it is also important to introduce security services into SDN controllers. For example, many controllers have IDS/IPS integration, authorization and authentication, resource monitoring, and logging/security audit services. Introducing these techniques into a controller is useful for enhancing the security framework of the controller.

4.5.1 IDS/IPS Integration

Intrusion Detection Systems IDS) and Intrusion Prevention Systems (PSI) can be introduced into SDN Controller for detection and prevention of potential threats (Yan and Prehofer 2011). The IPS and IDS are implemented so that all traffics are directed via IPS. The IPS will evaluate the traffic then allow or block them based on the policy (Yan and Prehofer 2011). Additionally, traffic monitoring can be provided through an authentic installation of traffic counting at selected nodes in the network. Next, the associated controllers generate traffic statistics for detecting attacks. The detection is determined by assessing the extent to which the traffic statistics deviate from the standard or predetermined traffic baseline (Scott-Hayward 2015). This controller self-defending mechanism has been employed Ryu and ES-Flooding, controllers.

In ES-Flooding controller, for instance, IDS/IPS integration has been accomplished by use of SDN Security Actuator and SRI BotHunter applications (Porras et al. 2015). The BotHunter monitors the data traffic and identifies the consistency of the communication patterns with coordination-centric malware. Ones the communication pattern has been defined by BotHunter, another application called BHResponder checks the identified asset and decides whether it needs to be isolated from the system. Finally, the SE-Floodlight connects with the SDN Security Actuator to implement the isolation. The quarantine is accomplished by the SDN Actuating generating OpenFlow rules which will then redirect the suspicious traffic flows. (Porras et al. 2015).

4.5.2 Authentication and Authorization

In SDN Controller design, Authentication, Accounting, and Authorization (AAA) are important aspects which provide efficient Access Control to resources, users, and applications. Although some Controllers such as ONOS lack specific AAA implementation, these features are well-evident in other Controllers like ROSEMARY and SE-Floodlight. In ROSEMARY, for instance, AAA system is implemented in which individual applications are allowed access to particular resources of the controller. For privileged system calls, the authorization status of an application is assessed by the application's authorization module through investigating the signed key corresponding to the application.

However, controllers such as the ES-Floodlight, applications are authorized to access Controller resources during the procedures of authenticating the application. An application authentication process involves generating a runtime credential for unique identification of the application (Kreutz, Ramos, and Verissimo 2013). For each message produced by the application, a credential is added, and without the credential, the application cannot be allowed to run (Scott-Hayward 2015).

4.5.3 Resource Monitoring

In certain occasions, a single controller may have multiple functions and applications running in the control framework. When this situation occurs, all the running applications and functions must be monitored in regards to how the controller resources are used. Thus, resource monitoring is essential to ensure that no any one application or service excessively consume resources. Although different strategies are used to manage resources in various controllers, a resource manager is most preferred for controlling resource (such as CPU, file descriptor, and memory) utilization by different applications. The resource management by a resource manager consists of a resource table to manage to maximum resources to which applications are assigned.

Therefore, resource monitoring helps in managing the available controller resources, and this is important in different ways. First, it helps in monitoring of the system to determine anomalous behaviors and detect buggy or malicious applications (Kreutz et al. 2015). Secondly, proper resource management ensures that the controller resources support the maximum number of applications (Kreutz, Ramos, and Verissimo 2013).

4.6 Improving the Robustness of a Secure and Dependable SDN

In modern Controller design, emphases are shifting toward methodologies to address various threat vectors including those discussed in section 4.2 of this chapter. In this regard, (Kreutz, Ramos, and Verissimo 2013) presents some of the most efficient mechanisms which are useful for mitigating the various threats associated with SDN Controllers. Approaches such as replication, diversity, self-healing, and dynamic device association have been considered.

4.6.1 Replication

Replication is a crucial technique both for improving the dependability of SDN systems and enhancing the network security. Specifically, replicating controllers is useful for achieving a secure and dependable network (Kreutz, Ramos, and Verissimo 2013). For example, as illustrated in the figure below, the controller is replicated with three instances. Additionally, a mixed replication approach has been accomplished in this example by replicating application B to ensure that tolerance of both software and hardware accidents malicious faults.

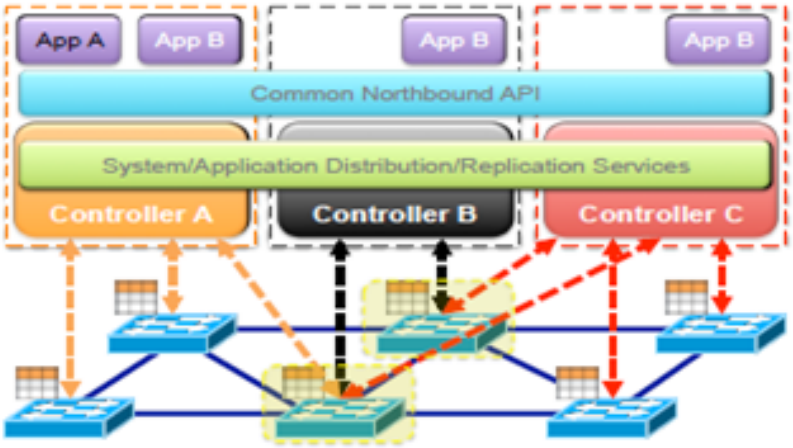

Figure 16. Secure and Dependable SDN

4.6.2 Diversity

Diversity is an important mechanism for enhancing the robustness of a secure SDN. Mostly, is necessary to replicate with diverse controllers for the avoidance of common mode faults (Kreutz, Ramos, and Verissimo 2013), including vulnerabilities and software bugs. For example, often, off-the-shelf Operating Systems from different families, are characterized by a limited number of intersecting vulnerabilities. Thus diversifying OS constraints the combined impact of attacks on common vulnerabilities (Garcia et al. 2014). In a Software-Defined Network environment, the same management program can run on different controllers. For simplification, a common abstraction for applications can be defined.

4.6.3 Self-healing Mechanisms

Reactive and proactive recoveries can restore the system to a healthy state and keep the network virtually functional by replacing the compromised components in the event of persistent adversary circumstances. However, for effective self-healing, it is necessary to replace the compromised components with new and diverse versions (Scott-Hayward, O'Callaghan, and Sezer 2013). In essence, diversity should be explored in the recovery process to strengthen the defense of the system against threats which target specific vulnerabilities in the network(Kreutz, Ramos, and Verissimo 2013).

4.6.4 Dynamic device association

There are situations in which a switch is associated with only one SND Controller. In such scenarios, the control plane of the particular switch will not tolerate faults. Thus, if the controller which is linked to the switch fails, there will be a failure in the control operations of the switch which means that the switch has to be associated with a different controller. So, to avoid such failures, it critical to ensure that each switch dynamically and securely associates with multiple controllers. In this case, the necessary security can be achieved through the use of threshold cryptography for detecting malicious controllers that could hinder attacks such as man-in-the-middle.

4.6.5 Trust between Controllers and Devices

Establishing a trust between controllers and devices is a crucial requirement for enhancing the trustworthiness of the overall control plane. T protect controllers; network devices should associate with controllers dynamically without causing unreliable relationships. A typical approach is to trust all controllers and the network devices until the trustworthiness of the Controller is substantially questionable (Scott-Hayward, O'Callaghan, and Sezer 2013). Additionally, the controllers should be set to report malicious or misbehaving controllers according to failure or anomaly detection algorithm. The malicious Controller should be automatically isolate when its trustworthiness falls below an unacceptable threshold.

4.6.6 Trust between controllers and application software

In this case, a dynamic trust model should be used for a Controller or a software component which is presenting a changing behavior as a result of attacks or bugs (Yan and Prehofer 2011). In this article, the authors propose the usefulness of a model which supports automatic trust management for software systems (component based). It involves a holistic notation trust where the trustworthiness of a trustee is evaluated by a "trustor." The trustor observes the behavior of the trustee and also measures the trustee based on specific quality attributes such as reliability, availability, confidentially, maintainability, safety, and integrity. Thus, the model can be applied to assess the relationship between controllers and software application and identify malicious behaviors.

4.6.7 Security domains

Different types of systems use isolated security domain technique to secure the network from attackers. In operating systems, for example, user level applications are denied access to kernel level systems (Barth et al. 2008). The trick may involve the use of sandboxes for isolating the rendering engine from the browser kernel (Barth et al. 2008). In this way, the impact of most attacks will not penetrate past the rendering engine. So, the isolation protects the security of the browser kernel. In SDN Controllers, a security domain is achieved using mechanisms such as virtualization of sandboxing (Scott-Hayward, O'Callaghan, and Sezer 2013). With this design, an active isolation mode can be established using a well-defined interface which allows minimal communication and operation sets between the isolated domains.

4.7 Security Requirements for SDN Controllers

Notably, SDN controller needs to meet certain security requirements. Below is a discussion of some of the requirements for a secure SDN Controller

- Authentication: -SDN Controllers must be designed with authentication which is supportive to the existing credentials including those which are likely to be applied at the datacenters.

- Authorization: SDN controller must have an interface which supports the authorization of specific network resources and applications. Additionally, the interface must support the manipulation of authorizations.
- Facilities for Isolation: SDN controllers need to offer various utilities through which one application can be isolated from the others. This separation may be necessary where a malicious application has to be isolated to avoid compromising the security of the entire system.
- The interface of SDN controller needs to support the controller which is operating as a proxy in place of an application. In this case, the SDN controller must favor a method of associating tracking IDs such as an audit ID. This is critical when a proxy is acting on behalf of applications. It allows requests to correlate with the original application.
- The controller interface should offer an approach for applications and operators to enforce privacy.
- Delegating accessibility to subnet resources: The interface of the Controller needs to support this access to allow for the supply of new privacy and authorization constraints. When SDN provide this requirement, it will facilitate inter-organization use and support various security needs of debugging Use Case.
- The interfaces of the SDN Controllers must support the control of authorization for nested applications.
- SDN Controller interface must provide a way through which outer applications can learn about the policies associated with nested applications. This is necessary for information to be shared between instances.
- SDN Controller must support authorization accounting and auditing by allowing nested applications to authenticate on behalf of associated outer applications.

4.8 Recommendations for Future security improvements on SDN Controllers

It is evidently from this literature review that there is a need to provide more security enhancement on existing SDN Controllers. Specifically, additional security features are necessary for the current and future SDN controllers to be robust, secure, and resilient. In this

regard, the following are three design recommendation to improve security in SDN Controllers:

- Design based on software security principles: SDN controllers should be designed according to secure software design elements including privilege limitation, sensitive data encryption, and secure defaults (Chandrasekaran and Benson 2014). Additionally, controller's security should be tested using static analytic tools.
- Secure default controller setting: All SDN Controllers should employ safe mode boot processes to ensure that controllers are secure during the entire lifecycle of the system.
- Application Future-Proofing: Applications should be designed outside controllers to enhance transferability across the controllers.

5 Conclusion

Software-Defined Networking (SDN) is a new source of the networking standard, which makes a communication network programmable. Network operators can run their infrastructure more proficiently, supporting faster deployment of new services while enabling key features such as virtualization. This approach useful to wireless mobile networks that will not only help from the same features as in the wired case, on the other hand will also have a control on the distinct features of mobile deployments to push improvements even further. Software Defined Network Architecture and OpenFlow protocol comes up with a solution to current problems in computer networks. Though the new change in SDN made the basis for further research, important changes in computer network technology have not been in use yet. SDN has combined the most motivating and interesting ideas in one architecture that totally separates centralized logic of control layer from data layer. This group of layers opens the path for new changes and advances in computer network technology independently of device manufacturers.

SDN provides operators with high flexibility in controlling the network. Through SDN capabilities, managing the network has shifted from codifying functionalities in regards to low-level device configurations to building an application which facilitates the debugging and management of the system. Furthermore, SDN offers new techniques for solving long-standing problems such as routing challenges in the network while simultaneously permitting the application of security features including access control. Such capabilities are provided in SDN environment because SDN allows for the separation of the complexity of state distribution from network specification.

SDN is important in today's modern datacenters as it deals with big data, help in maintenance of cloud-based traffic, manage traffic towards many IP addresses and virtual machines, make the infrastructure scalable and agile and also handle the managing policy and security. SDN deals with new techniques for solving long-standing problems such as routing challenges in the network while simultaneously permitting the application of security features including access control. Such competencies are provided in the SDN environment because SDN allows for the separation of the complexity of state distribution from network specification.

It is evidently from the literature review done in this thesis we can conclude that there is a need to provide more security enhancement on existing SDN Controllers. Precisely, additional security features are necessary for the current and future SDN controllers to be robust, secure, and resilient. Design recommendation to improve security in SDN Controllers can be Design based on software security principles, Secure default controller setting and Application Future-Proofing. Precisely, SDN can deal with more protected network if it is properly secured. SDN has the ability to transform the network industry.

Bibliography

Arbettu, Ramachandra Kamath, Rahamatullah Khondoker, Kpatcha Bayarou, and Frank Weber. 2016. "Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers". In *Telecommunications Network Strategy and Planning Symposium (Networks), 2016 17th International,* 37–44. IEEE.

Barth, Adam, Collin Jackson, Charles Reis, TGC Team, et al. 2008. "The security architecture of the Chromium browser". *Technical report.*

Benamrane, Fouad, Mouad Ben Mamoun, and Redouane Benaini. 2017. "New method for controller-to-controller communication in distributed SDN architecture". *International Journal of Communication Networks and Distributed Systems* 19 (3): 357–367.

Bittman, Thomas J, George J Weiss, Mark A Margevicius, and Philip Dawson. 2013. "Magic quadrant for x86 server virtualization infrastructure". *Gartner, June.*

Botelho, Fábio, Alysson Bessani, Fernando MV Ramos, and Paulo Ferreira. 2014. "On the design of practical fault-tolerant SDN controllers". In *Software Defined Networks (EWSDN), 2014 Third European Workshop on,* 73–78. IEEE.

Chandrasekaran, Balakrishnan, and Theophilus Benson. 2014. "Tolerating SDN application failures with LegoSDN". In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks,* 22. ACM.

Cui, Laizhong, F Richard Yu, and Qiao Yan. 2016. "When big data meets software-defined networking: SDN for big data and big data for SDN". *IEEE network* 30 (1): 58–65.

"Ethernet OAM enabled OpenFlow Controller". 2011. Visited on September 12, 2017. `http://nrg.sara.nl/presentations/SC11-SRS-8021ag.pdf`.

Garcia, Miguel, Alysson Bessani, Ilir Gashi, Nuno Neves, and Rafael Obelheiro. 2014. "Analysis of operating system diversity for intrusion tolerance". *Software: Practice and Experience* 44 (6): 735–770.

Godanj, Igor, Krešimir Nenadić, and Krešimir Romić. 2016. “Simple example of Software Defined Network”. In *Smart Systems and Technologies (SST), International Conference on,* 231–238. IEEE.

Hares, Susan, and Russ White. 2013. “Software-defined networks and the interface to the routing system (I2RS)”. *IEEE Internet Computing* 17 (4): 84–88.

Hazboun, Elias. 2016. “The Interface to the Routing System”. *Network* 25.

Hu, Fei, Qi Hao, and Ke Bao. 2014. “A survey on software-defined network and openflow: From concept to implementation”. *IEEE Communications Surveys & Tutorials* 16 (4): 2181–2206.

Hu, Yannan, Wang Wendong, Xiangyang Gong, Xirong Que, and Cheng Shiduan. 2013. “Reliability-aware controller placement for software-defined networks”. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on,* 672–675. IEEE.

Keller, Eric, and Jennifer Rexford. 2010. “The" Platform as a Service" Model for Networking.” *INM/WREN* 10:95–108.

Kirkpatrick, Keith. 2013. “Software-defined networking”. *Communications of the ACM* 56 (9): 16–19.

Kreutz, Diego, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2015. “Software-defined networking: A comprehensive survey”. *Proceedings of the IEEE* 103 (1): 14–76.

Kreutz, Diego, Fernando Ramos, and Paulo Verissimo. 2013. “Towards secure and dependable software-defined networks”. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking,* 55–60. ACM.

Lara, Adrian, Anisha Kolasani, and Byrav Ramamurthy. 2014. “Network innovation using openflow: A survey”. *IEEE communications surveys & tutorials* 16 (1): 493–512.

Liyanage, Madhusanka, Mika Ylianttila, and Andrei Gurtov. 2014. “Securing the control channel of software-defined mobile networks”. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a,* 1–6. IEEE.

McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. "OpenFlow: enabling innovation in campus networks". *ACM SIGCOMM Computer Communication Review* 38 (2): 69–74.

Medved, Jan, Robert Varga, Anton Tkacik, and Ken Gray. 2014. "Opendaylight: Towards a model-driven sdn controller architecture". In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a,* 1–6. IEEE.

Nunes, Bruno Astuto A, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. 2014. "A survey of software-defined networking: Past, present, and future of programmable networks". *IEEE Communications Surveys & Tutorials* 16 (3): 1617–1634.

"OF-Config-OpenFlow-Configuration-and-Management-Protocol". 2017. `http://searchsdn.techtarget.com/definition/OF-Config-OpenFlow-Configuration-and-Management-Protocol`.

"Open Flow Switch Specification". 2014. Visited on August 12, 2017. `https://www.opennetworking.org/sdn-resources/onf-specifications`.

"OpenContrail as SDN controller". 2017. `https://www.nanog.org/sites/default/files/3_Gorbunov_Opencontrail_As_Sdn_Controller.pdf`.

"opendaylight Controller". 2017. `https://www.sdxcentral.com/sdn/definitions/sdn-controllers/opendaylight-controller/`.

"OVSDB". 2017. `http://searchsdn.techtarget.com/definition/OVSDB-Open-vSwitch-Database-Management-Protocol`.

Pol, Ronald van der, S Boele, F Dijkstra, J Mambretti, J Chen, FI Yeh, M Savoie, B Ho, and L Sun. 2011. *Monitoring and Troubleshooting OpenFlow Slices with an Open Source Implementation of IEEE 802.1 ag.*

Porras, Phillip A, Steven Cheung, Martin W Fong, Keith Skinner, and Vinod Yegneswaran. 2015. "Securing the Software Defined Network Control Layer." In *NDSS.*

"Ryu Controller". 2017. `https://www.sdxcentral.com/sdn/definitions/sdn-controllers/open-source-sdn-controllers/what-is-ryu-controller/`.

Scott-Hayward, Sandra. 2015. “Design and deployment of secure, robust, and resilient SDN Controllers”. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on,* 1–5. IEEE.

Scott-Hayward, Sandra, Gemma O’Callaghan, and Sakir Sezer. 2013. “SDN security: A survey”. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For,* 1–7. IEEE.

“Software-Defined-Networking”. 2017. `https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/`.

Wallner, Ryan, and Robert Cannistra. 2013. “An SDN approach: quality of service using big switch’s floodlight open-source controller”. *Proceedings of the Asia-Pacific Advanced Network* 35:14–19.

Wang, Richard, Dana Butnariu, Jennifer Rexford, et al. 2011. “OpenFlow-Based Server Load Balancing Gone Wild.” *Hot-ICE* 11:12–12.

Yan, Zheng, and Christian Prehofer. 2011. “Autonomic trust management for a component-based software system”. *IEEE Transactions on Dependable and Secure Computing* 8 (6): 810–823.

Yoon, Bin Yeong, and Younghawa Kim. 2015. “Interworking model of transport SDN controllers”. In *Information Science and Security (ICISS), 2015 2nd International Conference on,* 1–3. IEEE.

Zhong, Xuxia, Ying Wang, Xuesong Qiu, and Wenjing Li. 2016. “FlowVisor-based cost-aware VN embedding in OpenFlow networks”. *International Journal of Network Management* 26 (5): 373–395.