

# **Trusted Communication in SDN OpenFlow Channel**

**Investigating Assumed Trusted Communication  
between SDN Controller and Switch and Possible Attack  
Scenarios**

Marika Haapajärvi

Master's Thesis

March 2017

School of Technology, Communication and Transport

Degree Programme in Information Technology

Cyber Security

Author Haapajärvi, Marika	Type of publication Master's Thesis	Date 20.3.2017 Language of publication: English
	Number of pages 99	Permission for web publication: yes
Title of publication <b>Trusted Communication in SDN OpenFlow Channel</b>		
Degree programme Master's Degree Programme in Information Technology, Cyber Security		
Supervisors Saharinen, Karo (JAMK) and Karjalainen, Mika (JAMK)		
Assigned by Miche, Yoan (Nokia Bell Labs)		
Abstract <p>Originally the subject of the research was Software Defined Networking (SDN) and particularly the threats towards the architecture and the SDN controller. However, during the research, it became evident that the protocol under research, OpenFlow itself had features which may turn into threats under the hands of a malicious actor outside the SDN network. The features may have vulnerabilities that may transform into threats should an attacker gain access to the SDN network.</p> <p>The verification of possible threat scenarios regarding the OpenFlow protocol vulnerabilities varied from configuring the SDN network to managing the traffic flows in the solution. Diverse SDN solution vulnerabilities and threats aspects were studied; from the history to the state-of-art of the technique and from the analysis of SW vulnerabilities according to vulnerabilities database to threat mitigations.</p> <p>The verification was not done in simulated environment but in an actual SDN network running on virtual resources with OpenDaylight SDN controller along with Open vSwitch virtual networking devices.</p> <p>The administrative tasks in overall have the greatest role in providing security in the SDN solution. To protect the SDN solution and mitigate the vulnerabilities, the SDN network should be configured according to secure communication principles with TLS. In case of connection failure, the SDN switch should be configured to reconnect to SDN controller and resume networking according to flow table rules. Also, the OpenFlow protocol features enabling unsecure communication in the SDN network, for example missing parameters in the flow entries or disabling networking statistics to prevent the monitoring of the SDN networking traffic, should be undeniably disabled.</p>		
Keywords/tags SDN Architecture, SDN Security, SDN Flow Table, OpenFlow Protocol and Threat Mitigation		
Miscellaneous		

Tekijä Haapajarvi, Marika	Julkaisun laji Opinnäytetyö, ylempi AMK  Sivumäärä 99	Päivämäärä 20.3.2017  Julkaisun kieli: Englanti  Verkkajulkaisulupa myönnetty: kyllä
Työn nimi <b>Luotettu kommunikaatio SDN OpenFlow -väylällä</b>		
Tutkinto-ohjelma Insinööri (YAMK), Kyberturvallisuus		
Työn ohjaajat Karo Saharinen (JAMK) ja Mika Karjalainen (JAMK)		
Toimeksiantaja Yoan Miche (Nokia Bell Labs)		
Tiivistelmä  <p>Tutkimuksen alkuperäinen aihe oli SDN-arkkitehtuuriin ja SDN-kontrolleriin kohdistuvat kyberuhat. Nopeasti tutkimuksen edetessä huomattiin, että OpenFlow-protokolla SDN-ratkaisussa sisälsi haavoittuvuuksia. Haavoittuvuuksia voi muodostua pahansuovan käyttäjän vaarantaessa SDN-ratkaisun kyberturvallisuuden hyökkäämällä sitä vastaan. Hyökkäyksen mahdollisuus avautuu SDN-verkkoa konfiguroitaessa ja vuotaulujen hallinnoinnin yhteydessä SDN OpenFlow -väylän komennoilla.</p> <p>Mahdollisten kyberuhkien varmistamiseen kuului SDN-verkon konfigurointi ja sen hallinnointi vuotaulujen avulla. Erilaisia katsantokantoja käyttäen tutkimusta tehtiin tutustumalla aikaisemmin havaittuihin haavoittuvuuksiin ja kyberuhkiin; siihen kuului SDN-arkkitehtuurin historiaan ja nykyiseen tilanteeseen tutustumista sekä haavoittuvuustietokantojen läpikäymistä ja kyberuhkien lieventämisen teoriaa.</p> <p>Kyberuhkien varmistaminen tehtiin oikeassa SDN-verkossa, ei simuloimalla. SDN-verkko konfiguroitiin virtuaalisiin resursseihin käyttäen OpenDaylight SDN -kontrolleria ja Open vSwitch -kytkimiä.</p> <p>SDN-arkkitehtuurin hallinnollisilla toimilla voidaan varmistaa sen kyberturvallisuus. Suojatakseen SDN-arkkitehtuurin ja lieventämällä siihen kohdistuvia haavoittuvuuksia, SDN-verkon hallintaliikenne tulee suojata TLS:llä. Jos tietoliikenneyhteys SDN-kontrollerin ja kytkimen välillä katkeaa, sen pitäisi palautua automaattisesti takaisin toimintakuntoon. Kytkimen pitäisi jatkaa pakettien edelleenlähettämistä vuotaulujen sääntöjen mukaisesti. Suojaamattomien toiminnallisuuksien käyttäminen, kuten tiettyjen parametrien määrittelemättä jättäminen vuotaulujen hallinnoinnissa ja SDN-tietoliikenteen monitoroinnin poistaminen käytöstä, pitäisi estää SDN-arkkitehtuurissa.</p>		
Avainsanat SDN arkkitehtuuri, SDN kyberturvallisuus, SDN vuotaulu, OpenFlow protokolla ja kyberuhkien lieventäminen		
Muut tiedot		

## Contents

Terms and Definitions .....	4
1 Introduction .....	5
1.1 Client .....	6
1.2 Parties Involved in the Research .....	7
1.3 Client Use Cases .....	8
1.4 Trend Threat Considerations .....	9
1.5 Practical Benefits .....	10
1.6 Research Methods .....	11
1.7 Objectives .....	11
2 Introduction to SDN .....	12
2.1 Brief History of SDN .....	12
2.2 Layers of Control Platforms and Their Scope .....	15
2.3 Flow Table .....	16
2.4 Security in SDN .....	19
3 Identified Threats Towards the SDN Controller .....	20
3.1 Introduction to the Threats Document .....	21
3.2 OpenFlow Protocol Weaknesses .....	22
3.3 Vulnerable SDN and Switch SW .....	23
3.3.1 Theory of Verification .....	26
3.3.2 SDN Network Statistics Monitoring .....	27
4 Verification of Threats .....	31
4.1 Threat Landscape Targets .....	31
4.1.1 SDN Controller and Switches .....	32
4.1.2 OpenFlow Channel Messages .....	32
4.2 Resources Needed in Verification .....	33
4.3 Verification Environment Configuration .....	35
4.3.1 Generic Verification Environment Configurations .....	35
4.3.2 SDN Controller Configuration .....	37
4.3.3 SDN Switch Configuration .....	38
4.3.4 SDN Network Configuration .....	38
4.3.5 Verification Tools .....	41
4.4 Verification Cases and Topologies .....	41
4.4.1 Reconnaissance capture .....	42

4.4.2	Statistics Packets Identification in The Control Channel.....	47
4.4.3	Lying to the SDN Controller.....	53
4.5	Analysis of Verification.....	62
5	Threat Mitigation Conclusions.....	63
5.1	Secure by Design.....	63
5.2	Computer and Operating System Security.....	64
5.3	OpenFlow Protocol Security Hardening.....	65
5.4	Layers of Security.....	66
5.5	Physical Security.....	67
5.6	Testing and Evaluating the Security.....	68
6	Discussion.....	68
6.1	Unsecure Default Settings.....	71
6.2	OpenFlow Protocol Exploitability.....	71
6.3	Mitigation.....	71
6.4	Networking Change Effects.....	72
6.5	Future Research.....	72
	References.....	73
	Appendices.....	77

## Figures

Figure 1.	Software Defined Infrastructure by HP. ....	14
Figure 2.	SDN Architecture (Op. cit. 13).....	15
Figure 3.	Flow table (Stanford University's OpenFlow Wiki 2010, chapter Flow Table Entries).....	17
Figure 4.	Flow script (Mahler 2014).....	18
Figure 5.	Flow table 0 rules and actions (Mahler 2014). ....	18
Figure 7.	Flow table 2 rules and actions (Mahler 2014). ....	19
Figure 6.	Flow table 1 rules and actions (Mahler 2014). ....	19
Figure 8.	OpenDaylight DLUX UI view on Nodes statistics. ....	29
Figure 9.	Node Connectors view in DLUX UI for SW4.....	30
Figure 10.	Flow entries example in the default flow table. ....	31
Figure 11.	OpenFlow version check command.....	32
Figure 12.	JAMK SDN testbed. ....	34
Figure 13.	Version info and pid from instance.properties -file of the Beryllium. ....	37
Figure 14.	DLUX modules.....	38
Figure 15.	Creating and verifying SDN controller and switch connection.....	39
Figure 16.	DLUX UI topology view of the SDN network.....	40
Figure 17.	Topology view for reconnaissance capture scenario. ....	42
Figure 18.	arp -a output from Kali host after MAC spoofing. ....	44

Figure 19. Switch properties.....	45
Figure 20. Changing switch fail mode to standalone and verification of command.....	45
Figure 21. Output of ovs-ofctl dump-ports -command.....	49
Figure 22. Ovs-ofctl dump-ports command output.....	50
Figure 23. Statistics request packet payload.....	50
Figure 24. Port statistics reply packet payload.....	51
Figure 25. DLUX UI output of SW4 statistics query.....	55
Figure 26. Contents of a new packet capture file created from a previous capture.....	58
Figure 27. Packet output in Scapy.....	59
Figure 28. Load payload to a variable rawload in scapy.....	60
Figure 29. Modify payload with Scapy.....	60
Figure 30. Statistics collection script.....	77
Figure 31. parse_stats.sh script.....	78

## Tables

Table 1. SDN controller and switch communication packets.....	33
Table 2. Generic verification environment settings.....	36
Table 3. SDN network and resource details.....	39
Table 4. Tools used in verification.....	41
Table 5. Command output and packet capture comparison, 3rd query.....	56

## Terms and Definitions

<b>CAM</b>	Content Addressable Memory.
<b>Flow rule</b>	An entry in a flow table with traffic forwarding instructions.
<b>Flow table</b>	A collection of traffic forwarding rules.
<b>SDN</b>	Software-Defined Networking
<b>SDI</b>	Software-Defined Infrastructure
<b>SDx</b>	Software-Defined Everything
<b>NBI</b>	North Bound Interface
<b>NFV</b>	Network Function Virtualization
<b>SBI</b>	South Bound Interface
<b>ODL</b>	OpenDaylight
<b>ODL Controller</b>	Open Daylight SDN Controller. Manages OpenFlow switches.
<b>OpenFlow</b>	SDN protocol standard for managing switches remotely from an SDN controller.
<b>Open vSwitch</b>	OpenFlow protocol compliant virtual switch for SDN architecture.
<b>Ovs-ofctl</b>	Command line interface on Open vSwitch virtual switch supporting and implementing OpenFlow protocol and its API.

## Thanks

Master's thesis instructor Yoan Miche, Specialist, from Nokia Bell Labs showed the wonderful world of research with innovative and inspirational guidance. Gabriel Waller, Research Department Manager, accepted the research to be done pro-bono. The benefits of this research for both parties hopefully pay off.

JAMK personnel; Karo Saharinen for instructing the master's thesis research. Janne Alatalo, Tuure Valo and Jarmo Viinikanoja for supporting in all kinds of dilemmas regarding the SDN Testbed.

And, special thanks to my common-law husband for supporting me throughout the studies, now there are two engineers in our family. That will be interesting...

# 1 Introduction

This master's thesis studies the threats towards the SDN architecture. The scope is especially in the possible OpenFlow protocol vulnerabilities which may be taken advantage of during the configuration of the SDN network and managing the traffic flows inside the SDN solution. The protocol vulnerabilities are discussed in detail in chapter 3.2 OpenFlow Protocol Weaknesses. Before going to the details of the protocol threats, a look back in history is taken on how the networking has evolved in the past two decades.

There has been a need for deploying and managing networks in an efficient and centralized way already in the traditional networks. The traditional networks have characteristics of being vendor-locked, proprietary and scattered, especially from the management point-of-view. The remedy for this puzzle was already drafted 20 years ago, in the minds of revolutionary researchers (Wetherall, Gutttag & Tennenhouse 1998, 1). At the time, the technological enablers were however missing (Feamster, Rexford & Zegura 2013, 5). The essence in the programmable networks is that networking resources are available via a programming interface. The interface, API, enables network management programmatically instead of complex physical changes and configurations (Feamster et al. 2013, 2 - 3). The details of programmable networks are highlighted in chapter 2.1 Brief History of SDN.

The programmable networks fit in the picture of today's networking needs again, as an easy and bundled way of managing networks from a centralized center. The technology trends form an overall digital mesh. Everything will be connected. The IoT and sensors augment the digital mesh (Gartner Top 10 Technology Trends 2016, 2016). These technology needs require more from the networking in connecting the resources together. The solution to the needs is virtualized, and shared resources such as cloud solutions are managed with the SDN architecture.

The malicious attackers have emerged in the networks connecting users, devices, sensors, services, information, data etc. The attackers have diverse reasons and motivation to attack. The attackers' objective is often in the financial gain, or they just may have malicious intents. Intelligence is another substantial cause for a cyber attack. (Slavov, Miche & Schneider 2016, 9.)



In the context of this master's thesis, a selection of threats is chosen to be verified. The selection is made according to Slavov et al. (2016, 22 - 34) considerations for the SDN specific threats. The document has identified different threats that range from the flow table manipulation to the network performance degradation. In between there are attacks via the forwarding plane, control network threats, i.e. the Southbound Interface (SBI) targeted attacks and attacks via the Northbound Interface (NBI). The research also discusses threats via virtualized environments and breach of domain isolation; however, those are not specifically addressed in this research. Nevertheless, the testbed for verification is entirely built in virtual resources since there may be some findings regarding that type of threats.

One interesting idea to verify is if it is possible to compromise the SDN controller via a virtual switch when asking for the flow table rules from it. Can the Data plane be compromised from a virtual switch? Also, it would be excellent to find and experiment with novel threats.

In addition to exploring the vulnerabilities in the SDN architecture, the threat landscape for the services running on the SDN orchestrated networks is drawn to understand cyber security. Also, an attacker mindset understanding is built to be used in ethical hacking.

## 1.1 Client

The client of this master's thesis is Nokia Bell Labs, and its organization Security Technologies. Security Technologies focus on research and standardization regarding the security technologies. The research topics are among others in SDN, machine learning and trusted boot.

This master's thesis and its results will provide input to Digile's DIMECC Cyber Trust project (DIMECC Cyber Trust Program n.d.). Cyber Trust is a consortium of industrial and research organizations collaborating in Security Technologies and Secure Management for Future Technologies project.

*DIGILE is a non-profit company operating between the public and private sectors with the goal of increasing know-how and developing the tools required by the Internet economy – and creating growth, jobs and new business in Finland through it. (Digile, about us 2016.)*

Cyber Trust project was set in motion as a major change in network architectures, protocols and management is expected. The change of networking architecture is expected to be based on the SDN and NFV. These architectures enable flexible provisioning, operation and management of the entire networking system securely. New business possibilities are also expected: e.g. multitenancy in data centers, which enables multiple tenants in the same SDN network to function in their own isolated network segments. (Cyber Trust Project Plan 2016, 1 – 3.) This brings scalability and green values to the networking in the form of deploying more resources based on the need and sharing the same resources with other tenants to allow scalability. Both the service provider and the tenant benefit from the solution as resources are optimally used with minimum and optimal investments. (What is SDN? n.d., chapter Benefits of SDN.)

Cyber Trust project's goals that are addressed in this master's thesis are related to SDN architecture security issues. The threat verification based on the Cyber Trust Threats document and possible new threats is expected to give input for continuous improvement of the project related testbed.

## 1.2 Parties Involved in the Research

The industrial organizations involved in the Cyber Trust project are Elisa Oyj, Nokia Networks, Nokia Bell Labs and Oy L M Ericsson Ab. Elisa Oyj has a project role as a business case coordinator and as a provider of mobile network B2B services. Nokia Networks and Nokia Bell Labs have an infrastructure provider role with a business interest especially in SDN and NFV. Oy L M Ericsson Ab has an interest as an infrastructure provider in the research of the new network challenges. (Cyber Trust Project Plan 2016, 7.)

The main research organizations involved in the project are Aalto University and JAMK University of Applied Sciences (JAMK). Aalto University acts as an academic research partner. JAMK's role is to act as a coordinator and a research partner that provides test bed hosting in the Realistic Global Cybersecurity Environment (RGCE). More information on the RGCE in chapter 4.2 Resources Needed in Verification. JYU (University of Jyväskylä), University of Oulu, Oulu Business School, University of Helsinki are in the supporting role in the business case. (Cyber Trust Project Plan 2016, 7.)

Digile's other services are related to the IoT Management (Digile IoT Management n.d.), Data to Intelligence (DIMECC Data to Intelligence n.d.) and Need for Speed (Digile Need for Speed n.d.). The IoT Management project description is following;

*Networking and Management research area aims solve technical issues which are important for the IoT above the physical networking layer. The important architecture issues include the infrastructure for device bootstrapping, end to end connectivity, monitoring and control capabilities. Also, in the research area we consider the infrastructure for network management and configuration, security, and object identification and discovery. (Digile IoT Management n.d.)*

The Data to Intelligence, D2I, handles big data, data processing technologies and user centric application in the security and customer intelligence to name a few of the seven project targets (DIMECC Data to Intelligence n.d.).

The Need for Speed, N4S, project concentrates on

*Digital resources are constantly available on-line, and for all to use. Increasingly, products and services are not developed by a single company but rather by a network of collaborating companies. This network of companies contributes to the ecosystem through different elements from both established and newly developed products, forming new, even more compelling offerings. (Digile Need for Speed n.d.)*

### 1.3 Client Use Cases

The client's main use case is for Data Centers (DC) providing Cloud services implemented with the SDN architecture. The DC solution may serve as a high-performance computing resource or as a web server. The high-performance computing resource can process Big Data efficiently. The Web server can provide a versatile platform for changing web usage and page creation: Web2.0 services for active web usage as opposed to previous passive content viewing like in the Web1.0 (Cormode & Krishnamurthy 2008, chapter 1 Introduction). The Semantic Web, Web3.0, would also benefit from the scalable and flexible networking. The SDN network may provide resources for data collecting processing power and data sharing (Dokulil, Tykal, Yaghob & Zavoral 2007, 1). The data processing especially requires scalable resources as it needs to find relations, i.e. interoperability between an actor, an event and for example, a location filtered with actor's common and unique preferences related to those variables (Shadbolt, Hall & Berners-Lee 2006, 1). The essence of semantic theory is "the logical connection of terms (that) establishes interoperability

between systems.” (Shadbolt et al. 2006, 1.) The processing of the gathered metadata and its output data is then relocated to suitable applications in the relevant devices (Dokulil et al. 2007, 2).

As learned earlier in chapter 1.1 Client the benefits in using the SDN architecture in Data Center (DC) solutions enable resource scaling by adding new virtual servers for better availability. When using the SDN, vendors cannot dictate to using their HW, configurations or protocols in the network management nor is the user bound to previous or future vendor solutions. This brings flexibility, cost savings, compatibility and freedom for the DC and related networking management.

Also, chapter 1.1 Client presents benefits for the SDN architecture tenant (e.g. a Cloud customer) in the form of scalability, cost reductions and green value add. According to the resource needs, a tenant can request scaling for the resources used. This is one part of the cost reductions as well as provision and de-provisioning of the resources according to the peak-times charged per usage i.e. pay-per-cost. The other cost reductions can be gained as no massive IT HW investments are no longer needed. As massive IT HW investments are not needed, the IT HW operational and the maintenance costs lower accordingly. By sharing the IT HW resources of the Cloud service provider green value is added to the tenant as well.

#### 1.4 Trend Threat Considerations

The Internet of Things (IoT) attached devices will require vast numbers of authentication services/servers when used in a secure manner (Lyne 2015, 2). The services are assumed to be running in the SDN operated resources. An attacker would try to compromise the authentication process to gain access to an IoT device. A compromised authentication service/server could lead to exploiting another trend threat, gaining access to personal records. (Lyne 2015, 2.) The personal records can be used as a negotiation advance in a ransomware scenario (Our Cybersecurity Predictions for 2016, 2015). The ransomware type of threats can be caused by an unpatched SW version in the target.

A web server could be running on the SDN orchestrated system. The web application threats enlarge the threat landscape significantly. As personal records are the threat trend target, the web server patching creates vulnerabilities which can enable injections, especially Cross-site Scripting, XSS, injection which can enable file and resource extraction

(OWASP Top 10, 9). In addition to the server patching, also plugins in browsers and servers (Content Management Servers, CMS, especially) enlarge the attack surface (Content Management Systems Security and Associated Risks 2013).

People may encounter spear-phishing as it is trending (Internet Security Threat Report 2016, 6) as well but how about the industry personnel's cyber security skills? The personal and human aspects require being on one's guard about the suspicious unknown content wedging from all around. There is a need for everybody to take responsibility in setting up and maintaining security and keeping up the security intelligence in the area (Op. cit. 47).

## 1.5 Practical Benefits

The research in this master's thesis context identifies the threats which hopefully help in finding possible flaws in the SDN architecture. The identified mitigations based on the verification results will help in preventing the threats from the user point-of-view in orchestrating the networking functions. For the developers, the usefulness may come in the form of using the threat findings as possible maintenance or security hardening ideas in the implementation phase. All these aspects come down to protecting the ICT security in all its aspects: authentication, integrity, confidentiality, authorization, availability and non-repudiation.

Currently it seems that only specific types of threats have been studied, like Denial-of-Service towards the SDN architecture; however, this master's thesis tries to make a more comprehensive threat identification and concentrates on those identified threats. Along the research and verification phases, new possible threats were found. Mitigations on the verified threats are suggested as a part of the research outcome.

This research will also carry out actual setup and verification of the identified threats to a certain degree within the timeframe constraints. The setup and verification is conducted in a virtualized environment with an SDN controller, OpenFlow switches and two Linux hosts, not simulating with mininet.

Considering the benefits of the research, there is a great number of existing documentation and information on the SDN basics, how it works, what different functional planes are, how the data flows are travelling in the network; however, the threats and mitigations to the threats have not been addressed, especially in the planned level of details as in this

master's thesis. The level of details includes the threat selection, verification setup and execution on the threat selection, the verification outcome analysis and the threat mitigation suggestions.

## 1.6 Research Methods

The meta-analysis is done in order to determine the state of the art of the SDN, SDN networking, security, threat landscape, weaknesses and possibilities in abusing the architecture. Also, traditional networking aspects in security, threats and weaknesses are considered if they are valid for the SDN architecture. However, the research mainly concentrates on the identified SDN security threats.

An academic discipline combines the research principles of sciences and humanities; the research is based on the library and written resources, and a selection of research based hypotheses is verified in the verification lab conditions. The hypotheses are applied to the real-life conditions in the form of threat mitigation suggestions.

The verification is carried out with the networking capture tools by carefully analyzing the results. The manual inspection is done in the laboratory conditions where the SDN Test Bed (more details in chapter 4.2 Resources Needed in Verification) is configured.

## 1.7 Objectives

The objective of this master's thesis is to understand the SDN architecture and deepen the knowledge on the security threats it may have. However, the main scope being the SDN, other related technologies and solutions are evaluated when in connection with the networking solutions or the services SDN provides (chapter 1.3 Client Use Cases). This is to help to gain understanding and overview of the technology field.

The Analysis of Current Software Defined Networking Threats (Slavov et al. 2016) identifies threats and vulnerabilities towards the SDN. Those identified threats and vulnerabilities, and the document itself will determine the threat selection to be verified in the scope of this master's thesis. Not all the threats are addressed in the verification part of this master's thesis.

The desirable outcome was met as new possible vulnerabilities were found in the course of conducting the verification. For those possible vulnerability findings, which may form into threats, mitigations are proposed based on the verification analysis.

## **2 Introduction to SDN**

In this chapter, the SDN history and present day are introduced. The overall architecture, main interfaces, layers of communication, interconnection between different modules are clarified along with the state-of-the-art research of the topic. Also, current limitations with regard to the SDN architecture and its security are studied.

### **2.1 Brief History of SDN**

The history of the SDN lies in the programmable and dynamically deployable networking. The active networks presented the network API concept where the custom functionalities can be applied to the networking resources (Feamster, Rexford & Zegura 2013, 1). There were two directions in the field at the time: capsule model and programmable networking device model. Also, according to Feamster et al. (2013, 1), there is a resemblance to the early telephony networks which had a clear separation of control and data planes. The separation simplified the network management and the deployment of new services. Briefly the principle of telephony networks, i.e. Public Switched Telephone Network (PSTN) was to split the telephony network into smaller manageable pieces, i.e. classes. Classes interconnected the different geographical areas and different devices in the telephony network.

Wetherall, Guttag & Tennenhouse (1998, 117) sketched an idea of an architecture of an active network toolkit which enables new protocol deployment automatically between the networking devices. The mobile code, i.e. general-purpose code techniques would be utilized to create a communication model which allows co-existence of the different networking protocols.

*The network service is flexible. Different applications are able to introduce new protocols into the network by specifying the routines to be executed at network nodes that forward their messages. Applications may customize network processing to suit their needs by pushing processing into the network - either processing that is traditionally performed at end-systems or novel kinds of processing that only make sense in the context of active networks. (Wetherall et al. 1998, 117 – 118.)*

The idea may have been too revolutionary and the active networks or early SDN for some reason did not pick up, the reason assumed to be the missing pushes in the technology and networking (Feamster et al. 2013, 5). Wetherall et al. (1998, 117, 127) identified difficulties in the network protocol change process, finding consensus on the new protocol, their interoperability and in the process deployment. Later on, Feamster et al. (2013, 2, 10) suggest remediation and catalyst being the progress in other related areas like distributed systems, operating systems and programming languages. The programmable network required programmable packet processing at high speed to evolve. Also, the use cases were not clear at the time.

For SDN to evolve from today on and become a solid and popular solution for networking, the use case need presented in history still persists, and as well as the customer's need for the technology. Mere networking management and packet forwarding with programmable APIs may not be enough but more networking service support may be required. The deep-packet inspection, transcoding of packets, programmable HW and Network Function Virtualization (NFV) could enable the wider networking services support according to Feamster et al. (2013, 11).

Originally the SDN was a Stanford University's OpenFlow project; however, now Feamster et al. (2013, 2) recognize a wide array of technologies as part of the SDN. Today the SDN is novel again. But, what does the abbreviation's SD-part mean, Software-Defined?

Longbottom (2015) refers it as decoupling software from hardware. Depending on the view point, What is Software-Defined Compute (2016) states that it is bringing the technology functions into the virtualized infrastructure. HP talks about the SDI, Software-Defined Infrastructure. In their definition when thinking about the SDN, it is one fundamental part



of the SDI along with computing, storage and facilities. The figure below (Figure 1) illustrates the solution (HP SDI for the future of Data Center. 2014, 13).

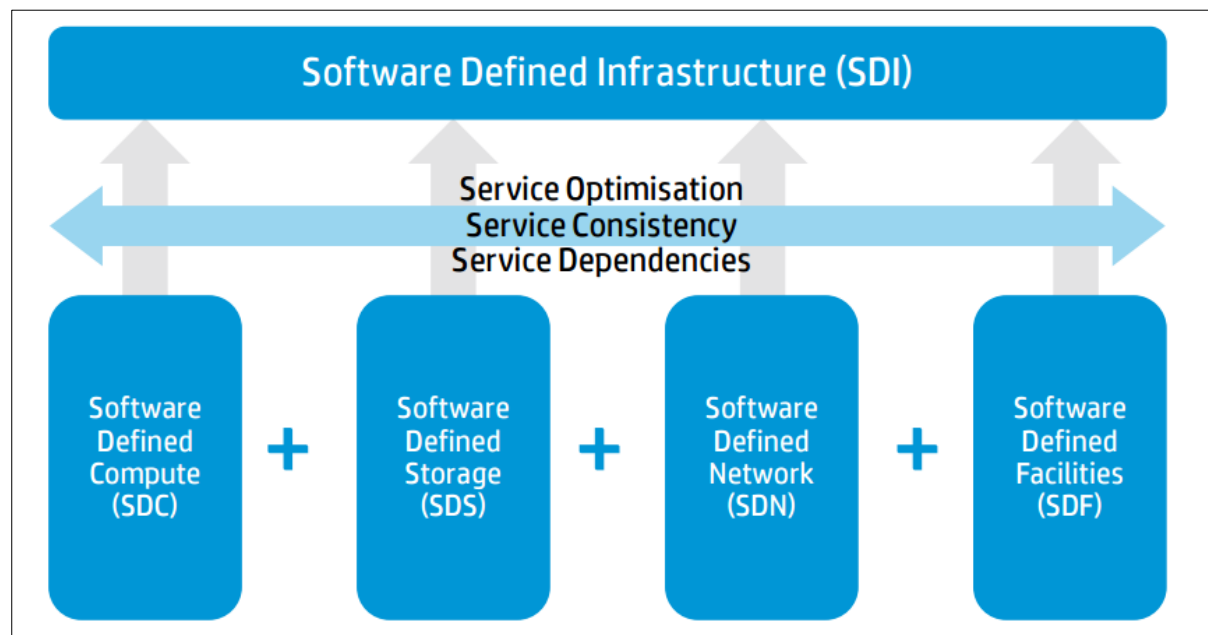


Figure 1. Software Defined Infrastructure by HP.

There is also related term SDE or SDx, Software-Defined Everything. That solution includes the SDN, Software-Defined Data Center, the SDDC, and Software-Defined Storage, the SDS. Including the facilities in the SDDC, which makes the difference of SDE/SDx with the SDI very minor or even non-existing (Matthews n.d.). The terms are overlapping as the SD in a sense is a new technological solution and an established terminology is not yet fully in place.

What is the SDN? In the SDI solution, the SDN enables networking between different SD objects. The SDN implements the core features of any SD object: architecture and planes of communication, main interfaces, and interconnection between the different modules.

The current limitations of the SDN architecture, and the OpenFlow protocol in particular, are considered here. The considerations are based on the technical restrictions of the OpenFlow protocol and their tendency to hinder the usage or making the solution vulnerable in the interest of making administrative tasks less complex or stressing the resources less. Some of these items are addressed in detail in the upcoming chapters and are especially related to forwarding rules, i.e. flow entry modifications and monitoring the traffic in the SDN network.

The latest specification for OpenFlow protocol, version 1.5.1, is used as a reference for the overall apprehension of the current state of the protocol implementation. The verification considerations are, however, done with devices supporting OpenFlow 1.3.5 as it is the version supported in the current virtual switch solution used, Open vSwitch. The limitations are considered between the latest OpenFlow release 1.5.1 to 1.3.5 as considering it as a realistic scenario that the latest version of the protocol may not be supported with the virtual switch solutions.

## 2.2 Layers of Control Platforms and Their Scope

The SDN architecture is a three-tier solution. The figure below represents the tiers (Figure 2). The SDN applications in the application plane communicate with the SDN controller via the North Bound Interface (NBI). The SDN controller is located in the Controller Plane. The SDN controller communicates with networking devices in the forwarding plane via the South Bound Interface (SBI). This represents the decoupling of the functionalities. (SDN Architecture 2014, 13.)

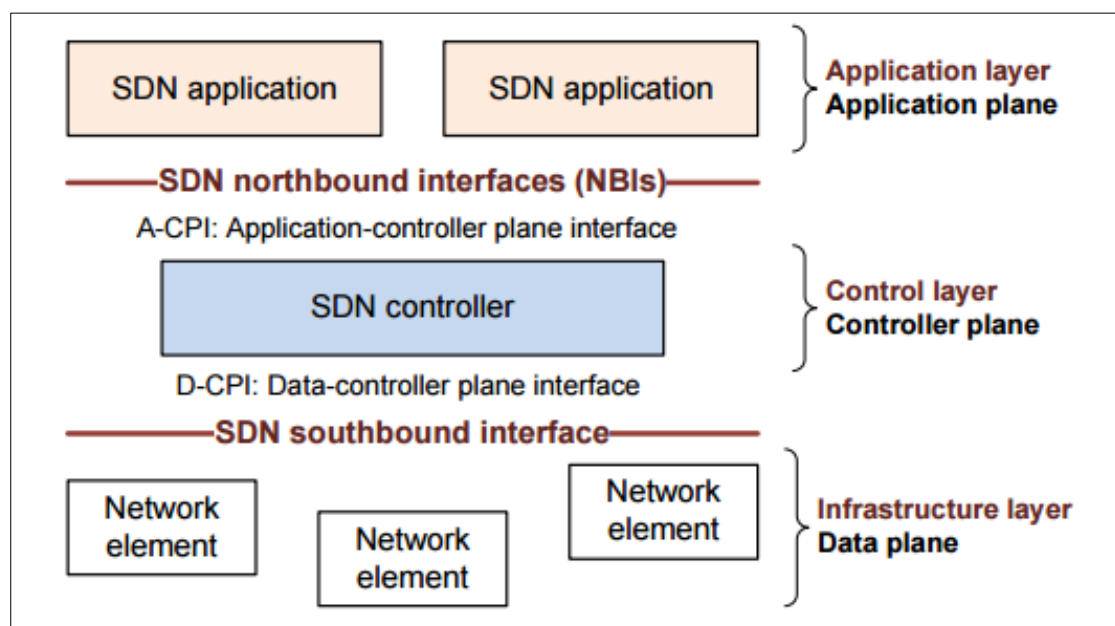


Figure 2. SDN Architecture (Op. cit. 13).

The decoupled architecture brings benefits to networking. The decoupling enables the device management centrally from the SDN controller, and the controller and the forwarding device need not to be on the same device. A device can be a physical or a virtual

networking device with the OpenFlow protocol compliance. The management is done via the open standard OpenFlow protocol. The versatile management enables the resource additions or reductions based on the traffic needs. The forwarding functions are conducted separately in the data plane. The decoupling also enables the applications to communicate directly with the SDN controller via API.

The benefits of the SDN include the architecture being “dynamic, manageable, cost-effective, and adaptable” (Software-Defined Networking (SDN) Definition n.d., chapter What is SDN). The dynamicity enables that new virtual devices can be added to the architecture for scalability purposes. The new enlarged entity can be managed centrally from the SDN controller. The possibility to use virtual devices and scaling them according to the usage needs makes the solution cost-effective. The vendor-neutrality in the devices and protocols makes the solution also adaptable.

## 2.3 Flow Table

This chapter describes the communication between the switch and the controller in the SDN architecture in the OpenFlow Channel and Control Channel. The significant messages in the transportation channel between the two entities are described.

*An OpenFlow channel is an interface between an OpenFlow switch and an OpenFlow controller. Control Channel includes one OpenFlow Channel per OpenFlow controller. (OpenFlow Switch Specification 1.3.5 2015, 11-12.)*

To understand the communication between the network elements and the SDN controller, the flow table format and its communication principles are clarified here.

*Flow table is a stage of pipeline. Pipeline is a set of linked flow tables for matching and forwarding packets. Flow table consists of flow entries. Flow entry is an element used to match and process packets. (OpenFlow Switch Specification 1.3.5 2015, 11-12.)*

A flow table consists of sequential flow entries, starting from 0. The flow entry contains the execution instructions for an action. The execution instruction can be an action, for example for packet forwarding or modifying. The processing pipeline is the path which a flow entry travels in the flow table based on the action set. The first three examples of the actions are self-explanatory in the below figure (Figure 3); however, the normal (packet) processing pipeline requires clarification. According to Fei (2014, 93), it refers to processing the packet as a Non-OpenFlow switch. This is the case of an action output being “forwarding packets to virtual ports”.

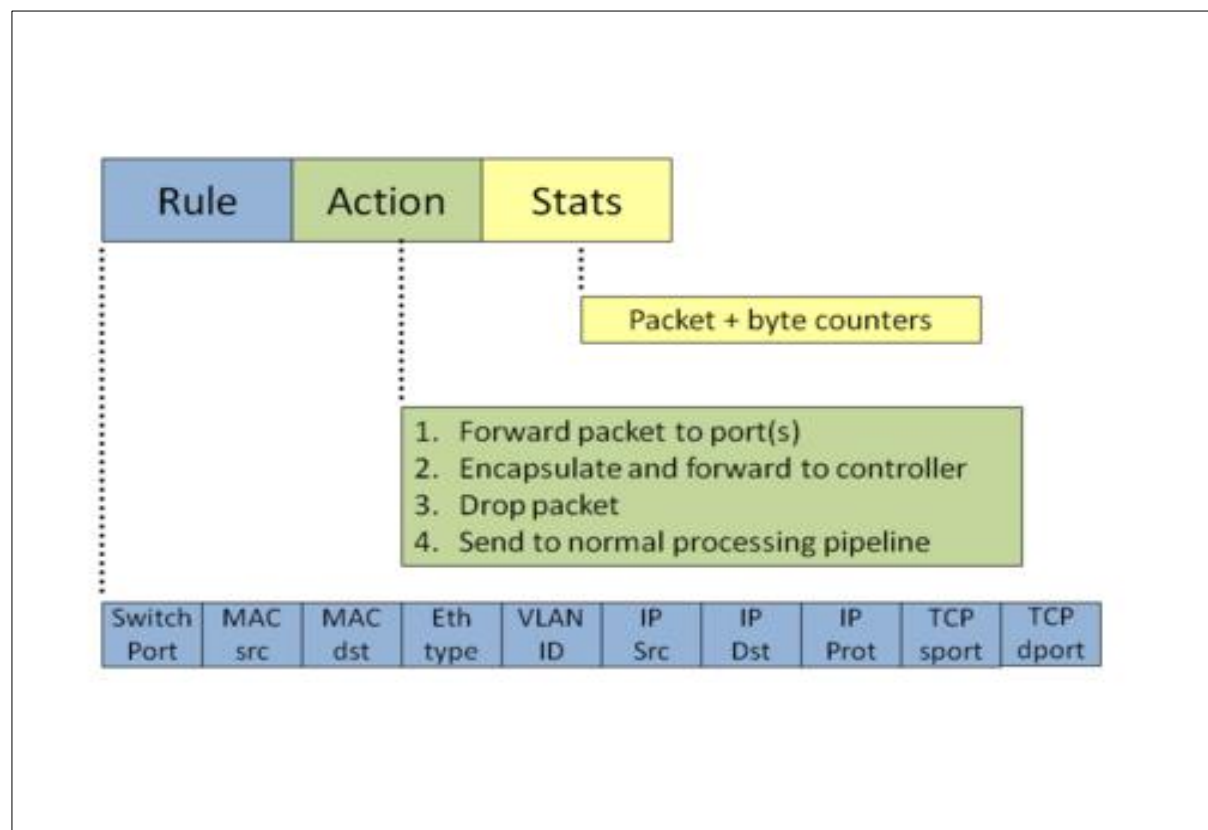


Figure 3. Flow table (Stanford University’s OpenFlow Wiki 2010, chapter Flow Table Entries).

The script `more_tables.txt` (Mahler 2014) in the example below (Figure 4) deploys flow tables and related actions to the mininet simulated virtual network; however, this example can be used to gain understanding of the flow table creation also in the realistic virtual device SDN controller networks.

```

mininet> sh more tables.txt
#table 0 - Access Control
table=0,ip,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=resubmit(,1)
table=0,arp,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=resubmit(,1)
table=0,icmp,nw_src=10.0.0.1,nw_dst=30.0.0.3,actions=resubmit(,1)
table=0,tcp,nw_src=10.0.0.1,nw_dst=30.0.0.3,tp_dst=80,actions=resubmit(,1)
table=0,ip,nw_src=30.0.0.3,actions=resubmit(,1)
table=0,priority=0,actions=drop

#table 1 - NAT
table=1,ip,nw_src=10.0.0.1,nw_dst=30.0.0.3,actions=mod_nw_src=5.5.5.5,resubmit(,2)
table=1,ip,nw_src=30.0.0.3,nw_dst=5.5.5.5,actions=mod_nw_dst=10.0.0.1,resubmit(,2)
table=1,priority=0,actions=resubmit(,2)

#table 2 forward/route
table=2,ip,nw_dst=10.0.0.1,actions=mod_dl_dst=00:00:00:00:00:01,output:1
table=2,ip,nw_dst=10.0.0.2,actions=mod_dl_dst=00:00:00:00:00:02,output:2
table=2,ip,nw_dst=30.0.0.3,actions=mod_dl_dst=00:00:00:00:00:03,output:3
priority=0,table=2,arp,nw_dst=10.0.0.1,actions=output:1
priority=0,table=2,arp,nw_dst=10.0.0.2,actions=output:2

```

Figure 4. Flow script (Mahler 2014).

The flow table 0 (Figure 5 below) allows all IP and ARP traffic in the 10.0.0.0/24 network. The Ping, i.e. ICMP packets are allowed from source 10.0.0.1 and destination 30.0.0.3 in the different subnet. Also the TCP connection to the port 80 from 10.0.0.1 is allowed to 30.0.0.3, which can be verified by issuing the curl command from 10.0.0.1 towards 30.0.0.3. The table 0 drops all the other packets, which can be verified by trying to ping from host 10.0.0.2 to destination 30.0.0.3. The suitable packets are forwarded to processing pipeline to the table 1, resubmitted to the next table by defining ,1 as a parameter.

```

mininet> sh more tables.txt
#table 0 - Access Control
table=0,ip,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=resubmit(,1)
table=0,arp,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=resubmit(,1)
table=0,icmp,nw_src=10.0.0.1,nw_dst=30.0.0.3,actions=resubmit(,1)
table=0,tcp,nw_src=10.0.0.1,nw_dst=30.0.0.3,tp_dst=80,actions=resubmit(,1)
table=0,ip,nw_src=30.0.0.3,actions=resubmit(,1)
table=0,priority=0,actions=drop

```

Figure 5. Flow table 0 rules and actions (Mahler 2014).

The flow table 1 (Figure 6 below) is acting as a network address translator: the IP packets from the 10.0.0.1 to the 30.0.0.3, their source address is changed to 5.5.5.5. When the IP traffic returns from the destination 30.0.0.3, the new source address for the 5.5.5.5 is set to be 10.0.0.1. These packets are forwarded to the processing pipeline to the flow table 2. The default action is to continue to the flow table 2.

The actual forwarding action takes place in the flow table 2. For example, IP packets to the destination 10.0.0.1 are forwarded via the port 1. The destination MAC address is also

```
#table 1 - NAT
table=1,ip,nw_src=10.0.0.1,nw_dst=30.0.0.3,actions=mod_nw_src=5.5.5.5,resubmit(,2)
table=1,ip,nw_src=30.0.0.3,nw_dst=5.5.5.5,actions=mod_nw_dst=10.0.0.1,resubmit(,2)
table=1,priority=0,actions=resubmit(,2)
```

Figure 6. Flow table 1 rules and actions (Mahler 2014).

changed in this action. The actions are respective in the first three entries in the flow table 2 (Figure 7 below). The latter 2 entries forward the ARP request to the 10.0.0.1 via port 1 and to the 10.0.0.2 via port 2. The actions are respective in the first three entries in the flow table 2. The latter 2 entries forward ARP request to 10.0.0.1 via port 1 and 10.0.0.2 via port 2.

```
#table 2 forward/route
table=2,ip,nw_dst=10.0.0.1,actions=mod_dl_dst=00:00:00:00:00:01,output:1
table=2,ip,nw_dst=10.0.0.2,actions=mod_dl_dst=00:00:00:00:00:02,output:2
table=2,ip,nw_dst=30.0.0.3,actions=mod_dl_dst=00:00:00:00:00:03,output:3
priority=0,table=2,arp,nw_dst=10.0.0.1,actions=output:1
priority=0,table=2,arp,nw_dst=10.0.0.2,actions=output:2
```

Figure 7. Flow table 2 rules and actions (Mahler 2014).

## 2.4 Security in SDN

The OpenFlow protocol security features are evaluated in this chapter. “The OpenFlow protocol does not implement security by itself, therefore the transport protocol or a protocol underneath it must provide security if it is needed.” (OpenFlow Switch Specification 1.3.5. 2015, 36.) In the SDN, the security is left on the administrators’ responsibility since there is an option: “The OpenFlow channel is usually encrypted using TLS, but may be run directly over TCP.” (OpenFlow Switch Specification 1.3.5. 2015, 31.) The before mentioned specification recommends to use the TLS version 1.2 or higher. It is the latest approved version of the secure communications over the Internet, and it is also suitable for the client/server solutions to maintain especially authentication, confidentiality and integrity in the communications (Dierks & Rescorla 2008, 1). The recommendation does not force the usage of the TLS.

The best features of the SDN architecture, such as centralized control and programmability, also make the architecture unsecure. According to Principles and Practices for Securing Software-Defined Networks (2015, 8-9), the centralized control may compromise the SDN

controller, which may endanger the entire networks' functionality. The programmability via the API enables the implementation of innovative applications but also leaves an open interface for the malicious applications.

The SDN controller implementations may have security enhancements which address the main issues of security: role-based authentication and security constraint enforcement. Hence, the next paragraph introduces briefly FortNOX and its successor SE-Floodlight and their solutions for these security constraints.

FortNOX is an extension for the NOX OpenFlow Controller. It implements the role-based authentication with three layers: the human administration with the highest priority, security application to further strengthen security policy due to identified malicious attack and non-security application with a lowest priority. The authentication is implemented with digital signatures. The security constraint enforcement is implemented with the conflicting flow rule detection, rule reduction and security policy synchronization (Porrás, Shinz, Yegneswarany, Fongy, Tysony & Guz 2012, 1, 3 - 4).

The SE-Floodlight is an extension to provide enhanced security to the SDN architecture with a different role-based authentication solution.

*Least Privilege feature enables OpenFlow applications to operate outside the controller process context. PACKET\_OUT Control: PACKET\_OUT messages produced by OpenFlow applications can now be restricted by administrators. Security Audit feature introduces a new OpenFlow audit subsystem that tracks all security-relevant events produced by the OpenFlow network stack. (SDN Security Suite 2015.)*

The security extension provides convincing security enhancements to secure the SDN architecture. The extensions could even be considered vital in using the SDN architecture and solutions.

### **3 Identified Threats Towards the SDN Controller**

In this chapter, the selection of threats to be verified is done. The selection is carefully reasoned based on the priority from the client's point-of-view and also considering the scope and resources in this research. The client has indicated that it would be in their interest to discover the possible SDN controller vulnerabilities, and if OpenFlow Channel and Control Channel can be compromised. This ties the whole SDN architecture to the

research, which in this case is the SDN network comprising of SDN controller and four virtual switches supporting the OpenFlow protocol. Their mutual communication is monitored to detect possible anomalies in the networking traffic caused by the threat simulations and the actual verifications.

### 3.1 Introduction to the Threats Document

Cyber Trust Collaboration is an ongoing project to make “Finland a leading country in Cyber Security”. In the scope of the project, programmable networks are in a significant role. The programmable networks as such are not a new thing as chapter 2.1 Brief History of SDN unveiled; innovative usage of the active networking was first crafted at the end of the 1990s, and its basis on the network management simplification by separating the control and data plane dates back to the early telephony networks.

The technology progress in the operating systems and programming in particular enabled the programmable networking as was discussed in chapter 1 Introduction. The new programmable networking may have unforeseen risks, and those possible risks are being assessed. If risks are identified, the treatment is planned according to those risks forming the vulnerabilities. The reference threat analysis does also risk treatment, i.e. mitigation of the possible vulnerabilities which may form into threats. This research complements the mitigations based on the verification outcome.

The document has identified attacks via the forwarding plane, control network threats, i.e. Southbound Interface (SBI) targeted attacks and attacks via Northbound Interface (NBI). The scope of threat verification in this research is in the SBI targeted attacks. From the reference document’s selection of the threats, following threats are verified:

1. Traffic capture is done to gather information of the target
2. Impersonating an SDN agent, especially OpenFlow switch, to compromise the SDN controller via the control channel
  - 2.1. Threats towards virtual SDN switches; compromising the switch running on hypervisor and utilizing hypervisor’s vulnerabilities
  - 2.2. OpenFlow table manipulation can be used in impersonation
  - 2.3. Attacking the Integrity and Confidentiality of the Southbound Traffic; hijacking connection and take over a switch
  - 2.4. Attacks mounted via forwarding plane: attacker is the user or has gained access to a host in the forwarding network.
  - 2.5. Attacks from the Control Network, i.e. Control Channel; utilizing dedicated physical network



3. Configuration errors are utilized and also protocol implementation flaws to generate vulnerabilities in the target network.
  - 3.1. Protocol flaws i.e. missing security functions enable unsecure networking

The selection done by the client indicated interest priorities. Certain threats enable a next step in the attack avenue, such as hijacking a connection to a switch and via that manipulating the flow tables. Those combinations of the threats are formed into interesting and relevant verification scenarios.

### 3.2 OpenFlow Protocol Weaknesses

To assess different ways possible to compromise the SDN controller, the possible weaknesses of the SDN OpenFlow protocol are considered here. The hard timeout for flow entries enables expirations in the defined time regardless of the packets being directed towards the flow entry. The use of the timeouts becomes needful in case of a switch losing the connection to the SDN controller, and expired flow rules should be cleaned out. However, if the switch has been configured to function in the fail standalone mode, it reverts to the Ethernet switching and the reconnection to the SDN controller does not occur. This will leave the functionality of the specific part of the network non-functional. In case of misconfigured promiscuous mode of the switch, all the traffic traversing the virtual switch is seen.

The administrator of the SDN network is not enforced to configure secure communication. The usage of non-supporting TLS communication in the SDN network allows unauthenticated access. An attacker may exploit this feature and gain access to the SDN network.

A handshake message is a feature request from the SDN controller towards a switch. The handshake indicates for the SDN controller what are a switch's basic functionalities (OpenFlow Switch Specification 1.3.5. 2015, 76). Without a successful handshake, there will be a handshake failure. A handshake failure can also be caused by a version mismatch in an OpenFlow command (State Machine n.d). Without a successful handshake, communication session will not be established in the Control Channel between the SDN controller and a switch.

When setting up the communication between the SDN controller and a switch, the mode of recovering from the connection interrupt may make the SDN network vulnerable. If `ovs-ofctl` command `set-fail-mode` is used to set the handling of connection failure to `fail standalone` –option, the switch will not try to re-connect after a connection failure. It also gains full ownership of the flow tables and flow entries. If a switch is under an attacker's influence, the attacker may manage the flow tables.

The configuration errors may cause vulnerabilities when enabling the SDN features. For example, the `hard_timeout` defines how long a flow entry keeps the flow rules active (OpenFlow Switch Specification 1.3.5. 2015, 80). Setting the `hard_timeout` to zero means that the flow entry is permanent (Op. cit. 21) and in case of the SDN controller disconnect, the unnecessary or malicious flow entry may keep on forwarding traffic.

### 3.3 Vulnerable SDN and Switch SW

For verifying in theory, the possibility of compromising a networking device connected to the SDN controller, a vulnerable version of the OpenFlow switch and OpenDaylight (ODL) SDN Controller SW are researched. In the case of a virtual NW device, switch, also a vulnerable SW version may pose the device to a vulnerability or a threat. The virtual switch is a newish technology emerged alongside virtualization and hence, it can contain implementation faults resulting in vulnerable SW. The physical switch SW is assumed to have more built-in security enhancements, however, they still may contain fundamental vulnerabilities, such as default passwords. To be suitable for this scenario, they need to support the OpenFlow protocol.

The scenario is to evaluate if a data plane switch connected to the SDN control plane can distort the data flow without the SDN controller to recognize the faulty data flow. The ability of the SDN controller to detect the faulty data flow begins with observing the flow statistics on the switch and on the SDN controller. The flow statistics should be identical in the case of legit data flows, otherwise the flow statistics counters comparison on the SDN controller and on the compromised switch should differ.

This Vulnerable SDN and Switch SW scenario does not differentiate whether the switch is virtual or physical, and therefore considers vulnerabilities on a generic level for switches. The only requirement is the OpenFlow protocol support on a switch.

Most of the vulnerabilities in the Common Vulnerabilities and Exposures database regarding the SDN and data flows are Denial-of-Service type vulnerabilities. The contingent ways to enable the DoS include authentication, remote authentication and hijack of the authentication of the administrators, checking of the administrative authorization role based authorization, buffer overflows and malformed traffic. (SDN and Flow Vulnerabilities n.d.). The latest version of the OpenFlow Switch Specification version 1.5.1 is used as a reference here to get an overview of all versions' possible vulnerabilities.

Are the control plane and data plane properly separated? Are they running on different interfaces? Next, these dilemmas are addressed. "If a packet does not match a flow entry in a flow table, this is a table miss" (OpenFlow Switch Specification 1.5.1. 2015, 20.) In this scenario, a switch contacts the SDN controller for further instructions on where to forward a packet. Other reasons to cause similar communication, i.e. packet\_in message transport between the SDN controller and a switch, can be invalid time-to-live (TTL) of the message or a flow entry instruction action being meant to be output to the SDN controller. The control channel is used for this kind of communication between the switch and the SDN controller.

*If a valid Connection URI specifies TCP as the protocol, the TCP connection is initiated by the switch on startup to the controller, which is listening either on a user-specified TCP port or on the default TCP port 6653. Optionally, the TCP connection is initiated by the controller to the switch, which is listening either on a user-specified TCP port or on the default TCP port 6653. (OpenFlow Switch Specification 1.5.1 2015, 46.)*

The communication between the networking devices and the SDN controller goes via the OpenFlow Pipeline. The out-of-band connections vs. the in-band connections are considered here in the aspect of how feasible it is to compromise especially the switch-controller communication path.

*For out-of-band connections, the switch must make sure that traffic to and from the OpenFlow channel is not run through the OpenFlow pipeline. For in-band connections, the switch must set up the proper set of flow entries for the connection in the OpenFlow pipeline. (OpenFlow Switch Specification 1.5.1 2015, 36.)*

The table misses can be caused by an invalid time-to-live (TTL), by an OpenFlow version mismatch between the SDN controller and the switch or by a feature request failure. The TTL is a packet-in feature, and the OpenFlow version and the feature request are responses

to the packet-in requests. This request-reply pair is called a handshake. (OpenFlow Switch Specification 1.5.1 2015, 104.) The failures in the message format or request response may cause the need for using the communication channel. An attacker could exploit the need of using the control channel communication with malformed packets. A malformed packet may initiate the request on the control channel in a packet-in message and spoof the data. If queuing or per-flow meters (limit sent packet rate to the SDN controller) are not used, the DoS may happen, at least more easily (Op. cit. 104).

In the port configuration messages, a property for not sending the packet-in messages to a specific port can be set according to OpenFlow Switch Specification 1.5.1 (2015, 67);

```
OFPPC_NO_PACKET_IN = 1 << 6 /* Do not send packet-in msgs for port. */
```

This property of the packet could be utilized not to send a malicious packet to the SDN controller for further instructions on how to handle it.

During the reconnaissance phase if a target information with (as per OpenFlow Switch Specification 1.5.1. 2015, 42)

```
protocol:name or address:port (tcp:xxx.xxx.xxx:6653)
```

is discovered, Connection URI may be compromised.

How to utilize the physical switch's management port remotely? Assuming here that the local connections are not feasible, the remote connection should be set up. The remote connection may be set up already by the system administration to debug remotely in the possibly problematic situations. This way an attacker could exploit the management port connections in case that have gained the credentials for it. The OpenFlow compliant switch needs to be configured to listen to the SDN controller via a specific port. Connecting to the switch via the management port remotely using the current transport port 6653 (OpenFlow Switch Specification 1.5.1 2015, 42), it is assumed that the connection to control channel can be achieved.

Next, the usage of physical switches in the SDN Solution is considered. The CVE vulnerability database has an item with a description:

*Cisco Network Services (CNS) NetFlow Collection Engine (NFC) before 6.0 has an nfcuser account with the default password nfcuser, which allows remote attackers to modify the product configuration and, when installed on Linux, obtain login access to the host operating system. (Vulnerability Summary for CVE-2007-2282 2013.)*

It would collect the traffic statistics when the data flows entered or exited an interface in the SDN solution. This feature is not applicable for this scenario as it is an additional SW on top of the Cisco IOS collecting the IP network traffic.

The next task is to try to find even lower level exploits, i.e. vulnerable switch OS versions which could be utilized in this scenario. Juniper's JunOS has a vulnerability which can be exploited with hand-crafted telnet messages to compromise the entire switch.

*Due to a buffer overflow vulnerability in the SRX Series flow daemon (flowd) while processing certain crafted telnet protocol messages, a remote attacker may be able to execute arbitrary code leading to a complete compromise of the system. This issue only occurs if telnet pass-through authentication is configured on the firewall. (2013-10 Security Bulletin (CVE-2013-6013) 2013.)*

The SRX is a firewall, not a switch SW, hence, it is not applicable for this scenario.

The Cisco switches use the NX-OS operating system which has vulnerabilities defined in the CVE. The vulnerabilities can be found with the keyword "Cisco NX OS" (CVE Cisco NX-OS Security Vulnerabilities n.d.). The CVE-2016-1329 (Vulnerability Summary for CVE-2016-1329 2016) reveals hardcoded credentials on the IOS, which allows remote attackers to gain root privileges via telnet or SSH. The vulnerable IOS is of version 6.0., this version of operating system could be installed to the physical switch in the verification environment. The physical switches have this kind of critical vulnerabilities and if those are not handled, the attacker has a very high chance in succeeding to compromise a switch and along with that the other devices in the network.

In the theory of verification outline, a virtual switch is either based on the OpenFlow protocol or supports it. The virtual switches in the SDN Testbed use the Open vSwitch OpenFlow supporting images.

### 3.3.1 Theory of Verification

The search of unsecure versions of the SDN controller did not require special effort; it is in the hands of the SDN administrator to leave the solution unsecure or to make it secure. The strengthening of the communication protocol, such as TCP with TLS is entirely optional. The

unsecure SDN network can be easily configured by leaving out the additional security features.

Using the non-supporting TLS configuration, the SDN controller allows unauthenticated access to it. By default, the TLS is not enabled in the OpenDaylight Beryllium SDN controller, hence the default settings are used for the verification scenario to mimic a vulnerable target. After gaining the unauthorized access to the switch, the attacker could make changes to the configuration files without the SDN controller taking notice. The Vulnerability Summary for CVE-2012-4122 (2013) vulnerability enables to overwrite or create arbitrary files via the shell output redirection. This vulnerability could be exploited in this scenario.

At this point of research, it was realized that the OpenDaylight SDN controller version needs to support the OpenFlow version 1.3, which is due to already configured Open vSwitches and their versions in the SDN Test Bed environment. Hence, the suitable specification version 1.3.5 for the OpenFlow switches is used from now on as a reference.

### 3.3.2 SDN Network Statistics Monitoring

A switch in the SDN architecture has capabilities of providing statistics for a flow, a flow table, a port, and network devices, i.e. interfaces, a group and a queue. With the network traffic statistics, it is verified that the generated traffic is reaching its targets in the SDN architecture. The generated traffic is simple, such as connectivity checks between the hosts with ping.

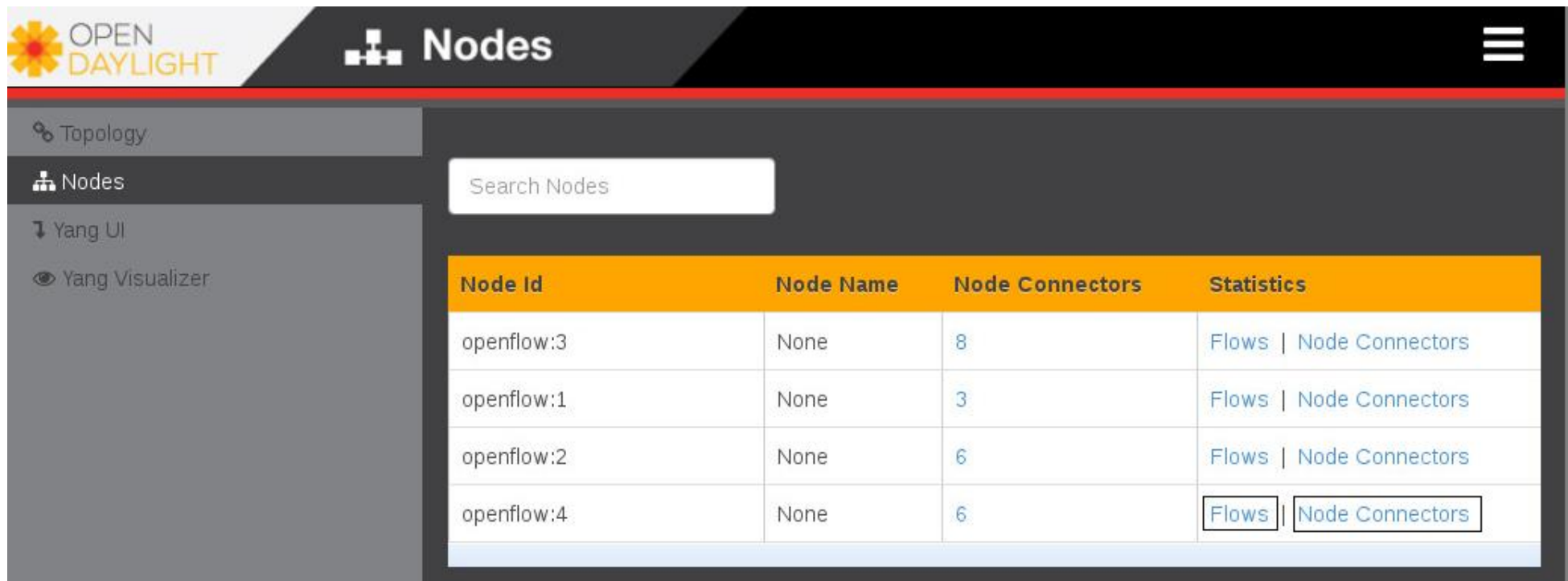
The group and queue statistics are not used. The group statistics are not used as the threat scenarios do not include multicast or flooding inside the SDN architecture (OpenFlow Switch Specification 1.3.5. 2015, 22). The queue statistics are not used as the Quality-of-Service packet feature is not set (Op. cit. 56).

The OpenFlow port level feature enables the collection of the detailed port statistics on a specified port from a certain switch. The other ways with different granularity to collect statistics are by flows or by flow tables. The flows present the statistics at the granularity level of packet and bytes statistics. The flow tables show statistics for all the tables with flow rules. The flow statistics also output the flow rules in a certain table. The summary on the packets and bytes is printed for the respective port, table or interface. The interface

statistics for the network devices associated with a switch are presented in the similar manner. (OpenFlow Switch Specification 1.3.5. 2015, 90, 93, 99.)

The statistics are collected in the SDN networking layer to verify the possible difference in the switch and in the SDN controller statistics counters. However, the granularity of the measurements is limited to the port based counters, the more accurate application layer measurements are not recorded in this phase. The granularity is based on van Adrichem, Doerr & Kuipers's (2014, 2) statement on using port based counters; "It lacks the ability to differ between different applications and traffic flows." This is where an SDN statistics application would bring remedy to in the form of collecting application specific networking traffic as mentioned in the assumptions when considering the scenarios for the use cases in chapter 1.3 Client Use Cases.

On the SDN controller side, DLUX UI, the graphical user interface, provides a view of the statistics. The statistics shown are port, flow table or flow based. The OpenDaylight's Model Drive Service Abstraction Layer (MD-SAL) module Nodes collects the networking and port statistics (Figure 8 on the next page). The Node ID is for a switch connected to the SDN network. The Node Connectors show the port statistics per switch (Figure 9 after the next page). The Node Connectors column shows the number of interfaces on the switch. It also maps ports to the interfaces and in the SDN network specific switch IDs. The flow table statistics for a specific switch (or node) can be seen with the option Flows. If no other than default table is defined, the graphical user interface will not show any statistics.



The screenshot displays the OpenDaylight DLUX UI interface for the Nodes statistics view. The top header features the OpenDaylight logo and the title "Nodes". A sidebar on the left contains navigation links: "Topology", "Nodes" (selected), "Yang UI", and "Yang Visualizer". A search bar labeled "Search Nodes" is positioned above the main content area. The main content area displays a table with the following data:

Node Id	Node Name	Node Connectors	Statistics
openflow:3	None	8	<a href="#">Flows</a>   <a href="#">Node Connectors</a>
openflow:1	None	3	<a href="#">Flows</a>   <a href="#">Node Connectors</a>
openflow:2	None	6	<a href="#">Flows</a>   <a href="#">Node Connectors</a>
openflow:4	None	6	<a href="#">Flows</a>   <a href="#">Node Connectors</a>

Figure 8. OpenDaylight DLUX UI view on Nodes statistics.



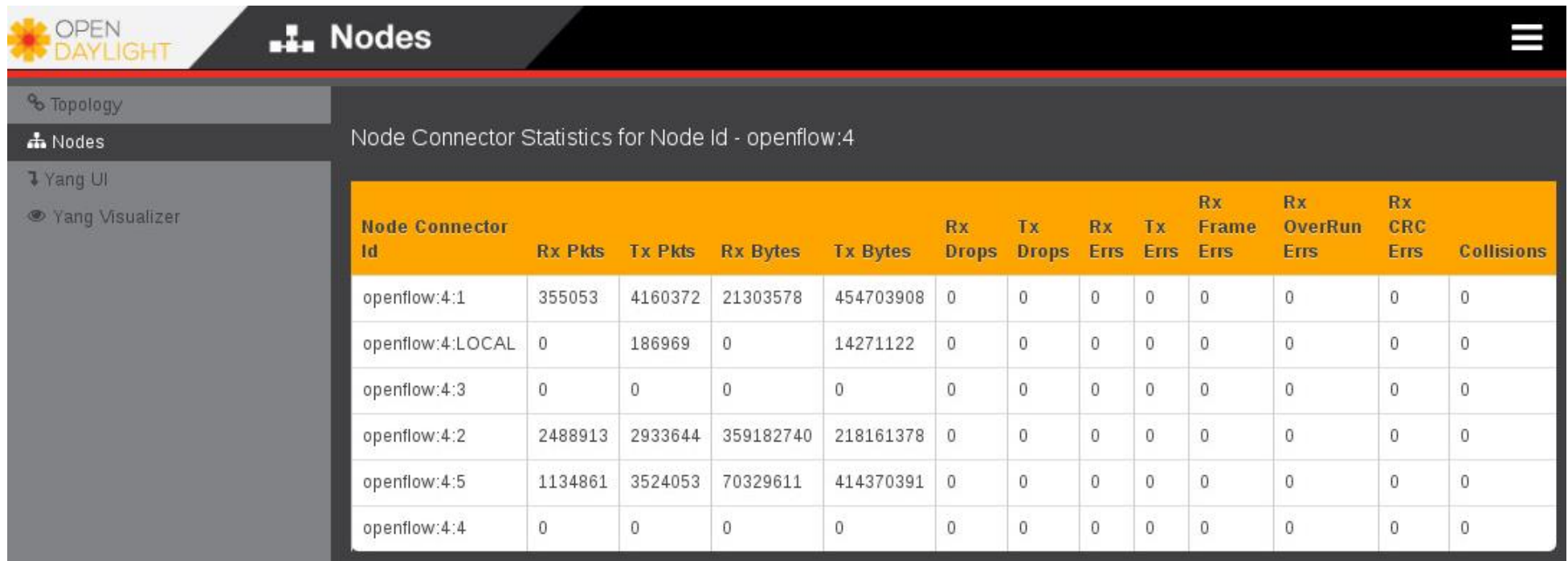


Figure 9. Node Connectors view in DLUX UI for SW4.

The command line tool for monitoring the OpenFlow Switches, `ovs-ofctl` (Open vSwitch Manual 2.6.90 n.d.) can be used to collect the statistics. The statistics are collected port, table or flow based. The different monitoring options can be used according to the list below. An example of the flow entries can be found in the figure below (Figure 10).

- Checks ports  
`show -O <openflow version> <bridge><if>`  
`show -O openflow13 of-switch LOCAL/eth1/eth2`
- Prints to the console statistics for each of the flow tables used by switch.  
`dump-tables -O <openflow version> <bridge>`  
`dump-tables -O openflow13 of-switch`
- Prints to the console all flow entries in switch's tables that match flows. If flows parameter is omitted, all flows in the switch are retrieved.  
`dump-flows -O <openflow version> <bridge>`  
`dump-flows -O openflow13 of-switch`
- Prints to the console statistics for network devices associated with switch. If `netdev` is specified, only the statistics associated with that device will be printed. `netdev` can be an OpenFlow assigned port number or device name, e.g. `eth0`.  
`dump-ports -O <openflow version> <bridge> <netdev>`  
`dump-ports -O openflow13 of-switch eth0`

```

root@sw1:~# ovs-ofctl dump-flows -O openflow13 of-switch
OFPST_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x2b00000000000000, duration=528110.616s, table=0, n_packets=1790143, n_
  bytes=138030722, priority=2,in_port=1 actions=output:2
  cookie=0x2b00000000000001, duration=528110.616s, table=0, n_packets=0, n_bytes=
  0, priority=2,in_port=2 actions=output:1,CONTROLLER:65535
  cookie=0x2b00000000000003, duration=528140.339s, table=0, n_packets=105629, n_b
  ytes=8978465, priority=100,dl_type=0x88cc actions=CONTROLLER:65535

```

Figure 10. Flow entries example in the default flow table.

## 4 Verification of Threats

In this chapter the threat landscape, verification environment and its configuration are described. The verification environment consists of the SDN Test Bed providing virtual resources for setting up the SDN controller and related networking devices. The environment configuration shows detailed configurations for the SDN architecture.

### 4.1 Threat Landscape Targets

The threat landscape targets consist of the SDN architecture related virtual or physical devices, SDN controller and switch(es) as well as the interfaces between them.

### 4.1.1 SDN Controller and Switches

The OpenFlow Switch Management Commands Command Line Interface, `ovs-ofctl` CLI, shows the supported version of the OpenFlow: 0x1:0x4 (Figure 11 below).

```
root@sw3:~# ovs-ofctl --version
ovs-ofctl (Open vSwitch) 2.3.0
Compiled Dec 19 2014 03:59:10
OpenFlow versions 0x1:0x4
```

Figure 11. OpenFlow version check command.

The release notes in the OpenFlow Switch Specification 1.5.1 2015 (p. 256 - 271) state that the codes mean support for the protocol versions 1.0 to 1.3.5. Hence, for the verification scenarios the version 1.3.5 OpenFlow Switch Specification is used as a reference. The chosen OpenDaylight SDN controller version, Beryllium, is backwards compatible with the OpenFlow protocol version 1.3.5. The multiple controller setup is not in the scope the master's thesis.

### 4.1.2 OpenFlow Channel Messages

The communication packets, controller-to-switch messages establish the connection between a controller and a switch in the SDN network. The example messages are listed in the next page (Table 1), and the communication direction is indicated to clarify the initiator of the message.

The handshake from the SDN controller requests identity and basic capabilities of a switch in its managed network. In reply, the switch shares its capabilities: flow, table, port, group and queue statistics. The capabilities also include reassembling the IP fragments and blocking looping ports. A Table Miss or invalid Time-To-Live field enable the sending of `PACKET_IN` messages towards the SDN controller. The switch has no rules where to forward such packets, since it is giving the control of the packets to the SDN controller. The `PACKET_OUT` is a response from the SDN controller to the switch for the `PACKET_IN` message (OpenFlow Switch Specification 1.3.5. 2015, 76, 107, 110).

Table 1. SDN controller and switch communication packets.

Type	Initiator	Message
<b>HANDSHAKE</b>	Controller	Identity and capabilities: OFPT_FEATURES_REQUEST
	Switch	Identity and capabilities: OFPT_FEATURES_REPLY
<b>PACKET_IN</b>	Switch	Table miss: OFPR_NO_MATCH
	Switch	Invalid TTL: OFPR_INVALID_TTL
<b>PACKET_OUT</b>	Controller	Reply to PACKET_IN: OFPT_PACKET_OUT

## 4.2 Resources Needed in Verification

The infrastructure and environments for verification are provided by the Realistic Global Cyber Environment (RGCE) of JAMK University of Applied Sciences (JAMK) (Cyber Operation Environment – RGCE n.d.). The environment, RGCE, functions like the Internet: it has real structures and functionalities of the Internet starting from public IP addresses and geographical locations to the core services of the Internet (e.g. name services, update repositories, certificate infrastructure). It is an isolated and controlled environment, and it offers a risk-free environment for use of attacks, known vulnerabilities and real malware. It also simulates the real network traffic as the traffic generation software utilizes the same functional principles as botnets. The bots are controlled by a Web based User Interfaces and command servers for modeling the realistic user behavior and end user traffic on the Internet. (Op. cit.)

The Management-VM host runs Linux Debian 3.16.0-4-amd64. It provides a graphical user interface for the OpenDaylight's DLUX User Interface.

In the RGCE, there are virtual machines configured for the verification in the SDN Testbed (Figure 12 on the following page). One virtual machine acts as a target for the malicious attacks (Host2) and the other one as the attacker (Host3). The environment also includes an SDN controller implemented with the OpenDaylight (ODL) and virtual switches (SW1-4).

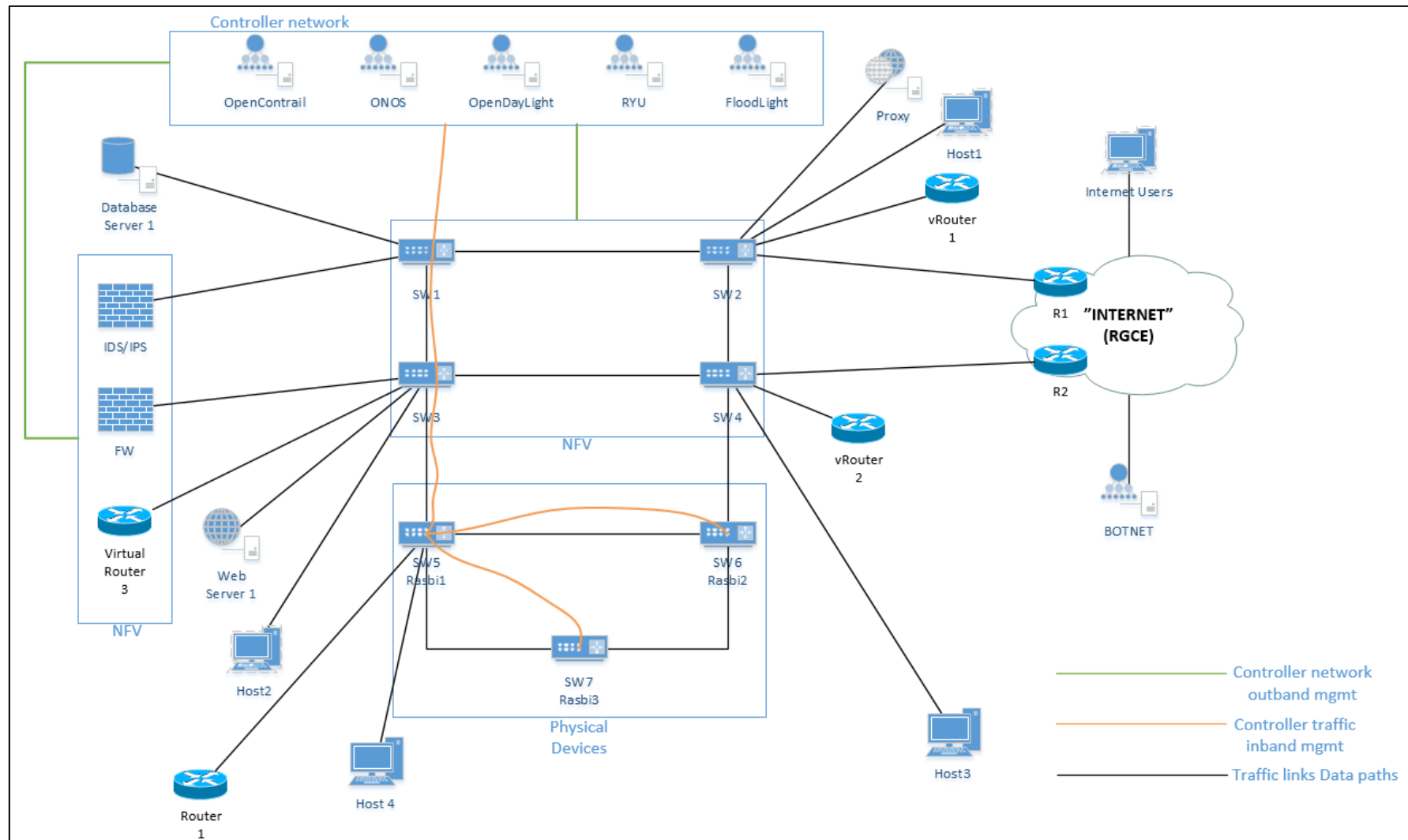


Figure 12. JAMK SDN testbed.

The verification concentrates on the SDN controller threats; however, the infrastructure for the verification includes a target and a host to ensure the communications and common malicious attacks with vulnerability and penetration testing tools; Kali Linux (Host3 in above Figure 12) and Metasploitable (Host2 in above Figure 12). Kali Linux is a penetration testing infrastructure:

*Kali Linux is a Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Kali contains several hundred tools aimed at various information security tasks, such as Penetration Testing, Forensics and Reverse Engineering. (What is Kali Linux 2016.)*

With Kali Linux's network scan tools and manifold penetration testing tool selection, the vulnerability of verification target, SDN architecture, is assessed.

Metasploitable is a penetration testing solution to simulate attacks with an extensive exploit database. Metasploitable also acts as an intentionally vulnerable target in the verification environment.

### 4.3 Verification Environment Configuration

The verification environment configuration will enable the threat scenario verifications. The verification scenarios aim is to find threats towards the SDN architecture. The threats may be very low level, i.e. protocol level enabled vulnerabilities. The protocol may allow vulnerable configurations to decrease the load on a switch by disabling the port statistics counters. However, the downside of enabling this feature may expose the switch to another vulnerability. In this case, SDN controller would not have knowledge of the port statistics and therefore, no full knowledge about the traffic in the SDN network.

The minimum-security configurations are intentionally enabled to highlight the vulnerabilities caused by the non-enforced security features.

The configurations cover generic verification environment configuration in the higher Linux OS level and also in more detailed level specifically for the SDN network and its devices.

#### 4.3.1 Generic Verification Environment Configurations

The generic settings of the entire verification environment are listed in the table below (Table 2). They will enable the basic functionalities to setup the verification environment.

The basic functionalities prevent the possible verification traffic to reach unintended targets, make use of the vulnerabilities of the OpenFlow protocol to compromise devices in the SDN architecture like switches and SDN controller, enable the OpenFlow statistics collection etc.

Table 2. Generic verification environment settings.

Action	Command
Terminate traffic to vulnerable target by disabling IP forwarding.	<code>echo 0 &gt; /proc/sys/net/ipv4/ip_forward</code>
Failure mode for switch in case of control channel disruption. Prevents switch from sending faulty flows towards the SDN controller. Can be used just in the opposite scenario to compromise the SDN controller.	<code>fail secure --mode</code> or <code>fail standalone --mode</code> <code>ovs-vsctl set-fail-mode &lt;bridge&gt; secure</code>
Configure flows with expiration time to prevent DoS (Not an interesting scenario as addressed in many papers.)	
Keys and certificates for TLS connection between switch and SDN controller <ul style="list-style-type: none"> <li>- Scenario: only controller side certificates are checked</li> <li>- Scenario: both the switch and controller certificates are checked</li> </ul>	
Verify TLS connection parameters to verify the actual version of TLS, cipher suite etc.	<code>openssl s_client</code>
Follow flows inside the flow table	
Optional: masquerading/spoofing; associating attacker's MAC address with another host's IP address to redirect messages to attacker	
OpenDaylight DayLight User Experience (DLUX) User Interface (UI) (OpenDaylight DLUX:DLUX Karaf Feature (n.d))	<p>Karaf to install networking modules, prior to accessing the DLUX</p> <ul style="list-style-type: none"> <li>- Relevant modules? restconf, odl I2 switch and switch to get started and get overview of all modules in the UI.</li> </ul> <p>Accessing the UI: <a href="http://localhost:8181/index.html">http://localhost:8181/index.html</a> (Default</p>

Action	Command
	password is admin/admin, changed immediately after first login)
Metasploit; add default gateway to connect to same network as Kali	route add default gw 203.0.113.1 eth0
Disable DHCP client on Kali to prevent unnecessary traffic during verification	Configure /etc/network/interfaces to use static configuration instead of DHCP. Disable dhclient.

### 4.3.2 SDN Controller Configuration

The OpenDaylight SDN controller Beryllium installation was chosen as it supports OpenFlow 1.0 and 1.3 which are the SDN Test Bed's compatible Open vSwitch OpenFlow versions.

The OpenDaylight SDN controller version info (Figure 13):

```
count = 1
item.0.name = root
item.0.loc = /home/user/opendaylight/distribution-karaf-0.4.2-Beryllium-SR2
item.0.pid = 6287
item.0.root = true
```

Figure 13. Version info and pid from instance.properties -file of the Beryllium.

To enable needed features in the SDN controller, installed features, i.e. plugins can be checked with the following command in Karaf:

```
feature:list -i
```

To verify which features are installed, referring to in below path is useful:

```
distribution/opendaylight-karaf/target/assembly/etc/org.apache.karaf.features.cfg
```

The features are installed on the need basis, and in the outset the following features were identified as needed features based on the OpenDaylight Features List (n.d).

- DLUX UI: graphical user interface for ODL. Feature name: odl-dlux-all, odl-dlux-core, odl-dlux-node
- YANG (controller components data structure): odl-dlux-yangui, odl-dlux-yangvisualizer
- OF-CONFIG: OF Switch communicating with OF, and configuration of OF data paths. Feature name in odl-of-config-all
- OF-PLUGIN: support for OF-enabled devices. Supported OF versions 1.0 and 1.3.2. Included in odl-config-all.
- L2Switch. Feature name odl-l2switch-switch-rest and ui.
- MdsAL APIDocs. Feature name odl-mdsal-all.
- REST API: odl-restconf-all



DLUX related modules in the Karaf UI (Figure 14 below):

```
opendaylight-user@root>feature:install odl-dlux-core odl-dlux-all odl-dlux-node
odl-dlux-yangui odl-dlux-yangvisualizer
```

Figure 14. DLUX modules.

Beryllium contains additional features to secure the SDN controller, in the outset, however, these features are not installed to identify new possible vulnerabilities due to the secure configuration disregard:

- CONTROLLER: multi-protocol controller infrastructure for multi-vendor networks.
- CONTROLLER-SHIELD: Controller security information for northbound application.
- GBP and Op-Flex: Application-centric policy model and group-based policy networking model.
- USC: encrypted communication between endpoints
- SNBI: automatic Controller discovery and secure IP connectivity
- DIDM: device specific configurations

#### 4.3.3 SDN Switch Configuration

For the compatibility purposes, the version of the OpenFlow protocol is queried on the switches with `ovs-ofctl` command:

```
ovs-ofctl -version
```

The output states the version to be 2.3.0; however, the most interesting information is the OpenFlow protocol supported versions so that the switches are compatible with OpenDaylight Beryllium distribution. The compatibility was verified in subchapter 4.1.1. SDN Controller and Switches.

#### 4.3.4 SDN Network Configuration

To setup the SDN network configuration, the IP address space is determined. It uses the internal network, 172.16.0.0/24. The SDN controller's and switches' interfaces are configured to that network to enable communication between the devices (Table 3 below).

Table 3. SDN network and resource details.

Resource	IP Address	Additional information
<b>OpenDaylight SDN controller</b>	eth0 172.16.0.15	ODL version Beryllium, runs on Debian Linux distro
<b>Open vSwitch SW1</b>	eth0 172.16.0.101	OF protocol v1.3.5 support
<b>Open vSwitch SW2</b>	eth0 172.16.0.102	OF protocol v1.3.5 support
<b>Open vSwitch SW3</b>	eth0 172.16.0.103	OF protocol v1.3.5 support
<b>Open vSwitch SW4</b>	eth0 172.16.0.104	OF protocol v1.3.5 support

The communication interface, Bridge (or datapath) between the SDN controller and the switches is named to *of-switch*. The figure below (Figure 15) describes the setting of the SDN controller IP address 172.16.0.15 to bridge of-switch. The connection type is TCP and the TCP port for the communication is the default port 6633 (OpenFlow Switch Specification 1.3.5. 2015, 161). The *is\_connected* value verifies that the connection is successful. The *target* is also verified in the bottom of the figure to be the correct one which was configured in the first command, tcp:172.16.0.15:6633.

```

@sw4:~# ovs-vsctl set-controller of-switch tcp:172.16.0.15:6633
@sw4:~# ovs-vsctl list controller
_uuid          : e5736583-0f52-4737-80e3-84f9b298b343
connection_mode : []
controller_burst_limit : []
controller_rate_limit : []
enable_async_messages : []
external_ids    : {}
inactivity_probe : []
is_connected    : true
local_gateway   : []
local_ip        : []
local_netmask   : []
max_backoff     : []
other_config    : {}
role            : master
status          : {last_error="Connection timed out", sec_since_connect="178", sec_since_disconnect="179", state=ACTIVE}
target          : "tcp:172.16.0.15:6633"

```

Figure 15. Creating and verifying SDN controller and switch connection.

The setup of of-switch bridge and communication between the devices in the SDN network are verified in the DLUX UI topology view of the SDN network (Figure 16 in the following page). All the configured OpenFlow switches are visible.

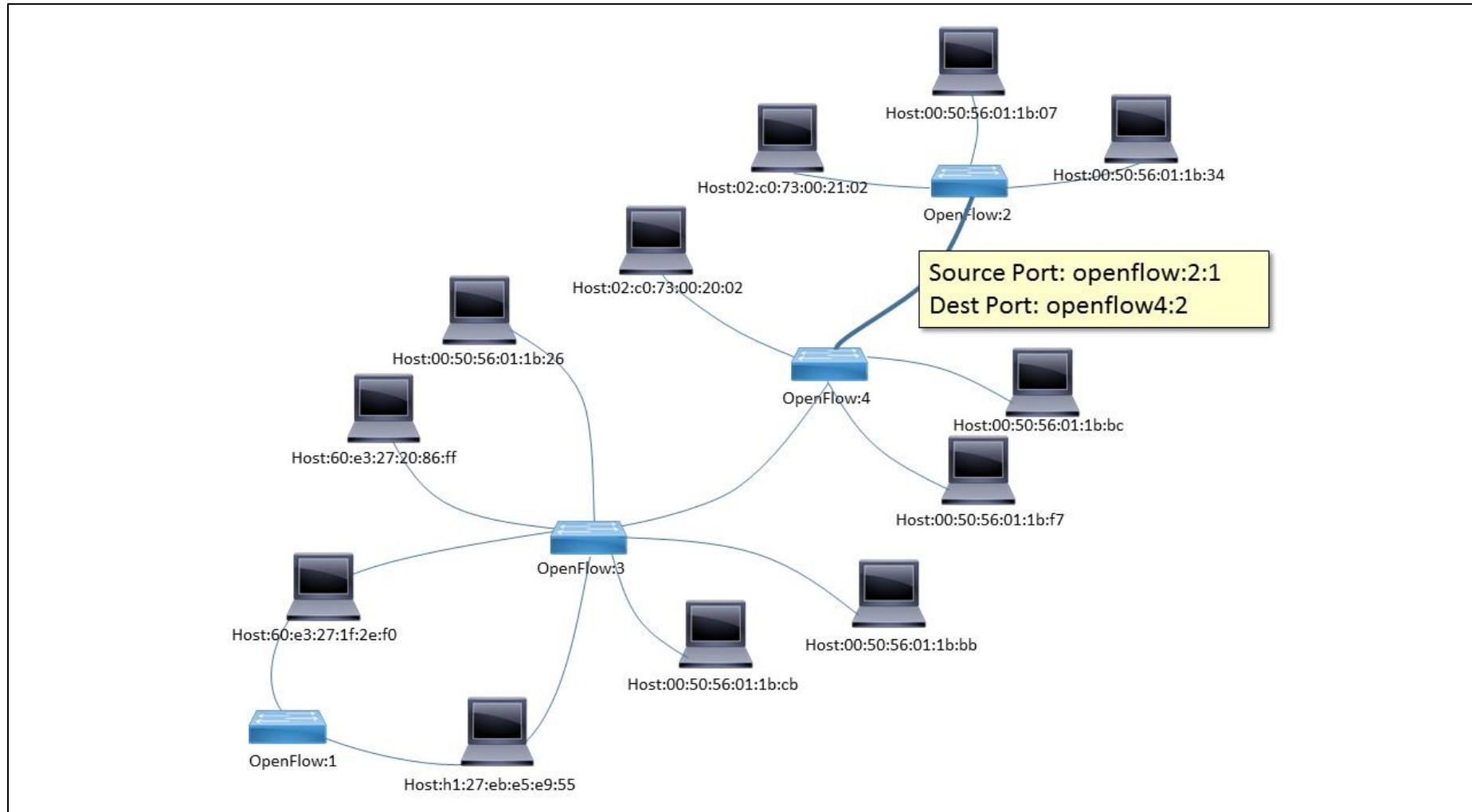


Figure 16. DLUX UI topology view of the SDN network.

#### 4.3.5 Verification Tools

The purpose of the verification tools is to capture networking traffic, distort it, modify it and resend it back to the SDN network (Table 4 below).

Table 4. Tools used in verification.

Tool	Version	Description
<b>EditCap</b>	1.12.1	Tools for extracting packet from wireshark pcap files
<b>Tcpreplay</b>	3.4.4	Replays networking traffic.
<b>Wireshark</b>	1.10.2	Network capture tool
<b>Macof</b>	2.4	MAC OverFlow tool.
<b>Scapy</b>	2.2.0	Packet reading, modification and sending tool i.e. packet manipulation tool.

#### 4.4 Verification Cases and Topologies

The verification part of the master's thesis concentrates on the SDN threats, not on the conventional networking threats which may have similarities. However, the conventional networking threats may suggest attack vectors applicable to the SDN architecture.

To confirm the verification results, logs and reports of the verification tools are gathered. The verification steps are carefully recorded to assure the verification case's reproducibility.

What is expected to be found during the verification? This master's thesis answers to the following questions:

- Is SDN controller vulnerable and how serious the possible vulnerabilities are?
- Is the selection of identified threats and attack vectors applicable to SDN controller?
- How to mitigate possible threats and attack vectors?

#### 4.4.1 Reconnaissance capture

In the reconnaissance capture the SDN architecture configuration has been setup according to chapter 4.3. Verification Environment Configuration and specifically its subchapters 4.3.2 SDN Controller Configuration and 4.3.4 SDN Network Configuration. The topology figure (Figure 17 below) illustrates the SDN network: SDN controller, switches, Kali host and management host interconnections and relation to RGCE. In the reconnaissance capture phase, Wireshark tool is used to capture the networking traffic. Without any generated traffic, there are, however, mainly ARP packets creating traffic. Also, some DHCP discovery and LLDP packets can be seen in the capture of 100 packets.

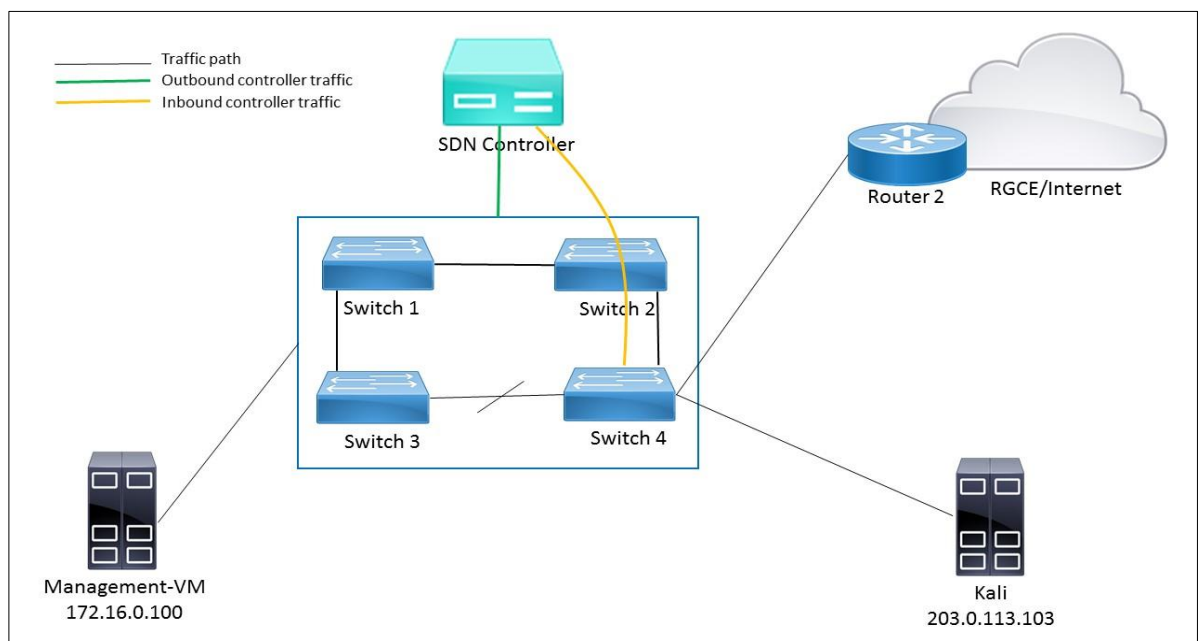


Figure 17. Topology view for reconnaissance

The target of the verification is to collect data which hopefully can be exploited to compromise the SDN controller. The first step is to try to extract information about the used OpenFlow version. Depending on the version used, some major vulnerabilities may be exploited (OpenFlow Switch Specification 1.5.1. 2015, chapter Release Notes).

ARP packets are of type broadcast from hosts/senders

- 172.16.0.101 SW1
- 172.16.0.102 SW2
- 172.16.0.103 SW3

Based on the IP addresses and MAC addresses in the packets, the sender can be united with the following devices with the help from the SDN controller's DLUX UI in the topology tab. SW3 OpenFlow:3:6 port has hosts connected to it which are configured to 172.16.1.0 network. The hosts try to reach the target 172.16.1.13 by sending broadcast ARP from addresses 172.16.1.101-103. Based on these broadcast ARPs coming from a different network, there is a misconfiguration in the network. That is something which does not, however, affect the outcome of the verification of the scenario.

An attacker could obtain the same data and make the same assumptions based on the broadcast ARPs. Based on this data, the attacker could try to obtain the connection to the switch. When obtaining the connection to the switch, it takes the attacker one step closer to compromise the SDN controller. Compromising the SDN controller as such is not even needed if access to a switch in the SDN network is gained. The outcome of verification will make this assertion evident as will chapter 4.4.3 Lying to the SDN Controller show.

The Link Layer Discovery Protocol (LLDP) packets are also travelling in the network. The purpose is to advertise a networking device's identity, capabilities and neighbors. Based on the system name field of the packet, openflow:4, assumptions can be made that the device is an OpenFlow switch. The information known by the network administrator states that openflow:4 is in fact the switch #4, SW4, which is the first forwarding device for Kali machine. The type of the switch can be concluded to be a VMWare device based on the MAC address information in the LLDP packet. The source MAC address information

*Vmware\_01:1b:d9 (00:50:56:01:1b:d9)*

states that switch is running on the VMWare, hence, it is most likely virtual switch. The port subtype value is 5:

*Port Subtype = Locally Assigned, Id=5*

which conforms to DLUX UI's topology view's figure that the Kali server is connected to the SW4's port 5.

Determining the IP address of the openflow:4 switch, i.e. SW4 is commenced by doing a broadcast ping to the broadcast address of the network. The broadcast ping with ping -b 255.255.255.255 command does not, however, reveal any new packet in the Wireshark

capture. Running the `arp -a` command after broadcast ping should combine the MAC address and IP address; however, not in this occasion as the SW4 is in a different network. The LLDP packets are multicast type packets, which have been seen also in another network in the layer 2.

In the SDN architecture and OpenFlow switches, a LLDP packet comes from SDN controller in `packet_out` message format and at that point a switch does not have any knowledge of the packet. In fact, an OpenFlow switch cannot by itself send, receive or process LLDP messages according to Pakzad, Portmann, Wee Lum & Indulska (2014, 3). Hence, the switch sends the packet to the SDN controller in `packet_in` message and the SDN controller can form network topology based on those messages including switch ID and ingress port number. By default, in the SDN architecture the discovery is done in regular intervals so that the tracing of LLDP packets is not restricted to initialization time of the SDN network.

The Netdiscover application of the Kali is used next in trying to determine the IP address of the SW4. Without parameters, it scans all the available and internal default networks. It uses only local network, and the device based the MAC in question is not found in this network. The tool did not provide information about the IP address of the SW4, not even when a virtual interface was configured in the same network as the SDN controller and the switches. This would indicate that the SDN network does not allow by default unauthorized devices to join the network, preventing this with flow table rules.

Using the administrator known IP address of the SW4 it was injected into the ARP table.

```
arp -s 172.16.0.104 00:50:56:01:1b:d9
```

The injection did not succeed as the network was unreachable. There is no network configuration in the Kali host for the network in question. In theory to get the IP address of LLDP sending device would be the first to spoof the MAC address on the KALI server. The MAC spoofing would require a virtual LAN configuration for the networking interface eth0. If the spoofing were done straight to the eth0, all traffic would stop. After the VLAN

```
kali:~# arp -a
? (172.16.0.104) at 00:50:56:01:1b:d9 [ether] PERM on eth0.103
```

Figure 18. `arp -a` output from Kali host after MAC spoofing.

configuration for the eth0 the injection was successful, and the entry was included in the attacker machine, i.e. in the Kali host ARP table (Figure 18).

At this point, also additional insecure features will be set to the OpenFlow Channel. By default, it has been observed that the Open vSwitch switch operates in fail secure mode (Figure 19 below).

```
@sw4:~# ovs-vsctl show
2a8b692a-a028-45cd-a349-1195e9cb3792
Bridge of-switch
  Controller "tcp:172.16.0.15:6633"
    is connected: true
    fail_mode: secure
```

Figure 19. Switch properties.

This mode is set to *fail standalone* (Figure 20 below). The change is expected to make the switch function as legacy Ethernet switch (OpenFlow Switch Specification 1.3.5 2015, 38). Starting from OpenFlow version 1.3.3. the fail standalone mode also means that “the switch owns the flow tables and flow entries”. (OpenFlow Switch Specification 1.5.1 2015, 268.)

```
@sw4:~# ovs-vsctl set-fail-mode of-switch standalone
@sw4:~# ovs-vsctl show
2a8b692a-a028-45cd-a349-1195e9cb3792
Bridge of-switch
  Controller "tcp:172.16.0.15:6633"
    is connected: true
    fail_mode: standalone
```

Figure 20. Changing switch fail mode to standalone and verification of command.

This standalone mode enables more autonomy and configuration possibilities on the switch regarding the flow tables management. It is also used as it is expected to be one the enablers in compromising the SDN controller.

A requirement for this state is also to disconnect the SDN controller connection. After disconnect, the switch should start function in the legacy mode and have more flow table management openings. The SDN controller disconnect is simulated by exiting the Karaf application on the SDN controller. In this way, it is experimented to see if MAC address overflow command *macof* has any effect on filling the CAM (Content Addressable Memory) tables of switch. Thus, in the possible case of CAM table overflow, the switch should start flooding all packets to network, and those could be captured by the attacker to hopefully



gain information about the network and its devices. However, the downside of this scenario is that it did not cause an overflow on the SDN controller and did not close the OpenFlow Channel towards the switch.

After sending the massive amount of 4 000 000 MAC overflow packets, identified as malformed packets, no abnormal behavior was seen in the network. The fail standalone mode is kept still on regardless of the fact that no effects were detected during the scenario. However, the traffic flooding after the CAM table overflow may concern only the new connections according to the MAC Flood Buffer Overflow Attack (2012). The site also advises to verify the traffic flooding after MAC flooding and verified successful clear text passwords visibility via FTP connection.

The MAC flooding with Macof tool is not a suitable stress procedure as the OpenFlow switches do not have MAC tables but flows. The networking traffic capture tool Wireshark on the Kali host sees the MAC overflow packets as malformed packets assumed to be forwarded to the SDN controller for further actions as the packet does not match any entries in the default flow table. The confirmation of malformed packets from the administrator point of view can be verified in the flow statistics. However, this procedure may not be suitable for determining the switch's IP address and it can be used to generate control channel traffic via an OpenFlow switch and the SDN controller in case that is needed. In this case, it has been proved that traditional networking stress tool does not operate correctly in the SDN network. Similar functionality can theoretically be achieved by overflowing the flow tables.

Still at this point the attacker has not compromised the SDN architecture or network.

DHCP Discovery packets are sent by the Raspberry devices based on the source MAC address in the DHCP discovery packet. Also, the Kali server sent the DHCP Discovery messages regardless of the static interface configuration. This reveals the type of other hosts in the network which can also be valuable information for an attacker trying to compromise the SDN controller. These DHCP messages create unnecessary and distracting traffic in the SDN network, hence for the upcoming verification cases the DHCP clients (dhclient process) need to be disabled from devices in the SDN network.

#### 4.4.2 Statistics Packets Identification in The Control Channel

The purpose of the identification of the statistic packets is to identify how the forwarding plane communication functions in the SDN network. If an attacker has access to a switch in the SDN network, there are many ways to exploit the solution. The next scenario tries to find a way to identify a packet in the Control Channel, use it maliciously to forge data and by that way make the SDN architecture vulnerable. By identifying a certain type of packet, an attacker could modify the captured packet's payload to gain intended malicious objective of lying to an SDN controller. The capture could give the attacker the format of the packets and, particularly, how to form the payload when an example packet has been identified. With that information, the attacker could create new packets based on the fields in the statistics command capture and use them maliciously. The attacker could also create invalid frames to enable Control Channel traffic between a switch and the SDN controller to sniff that traffic.

By identifying the correct OpenFlow packets, the attacker could tamper the packets to forge the packets themselves, make flow rules for these tampered packets to forward them to unintended destinations and make them inaccurate to hide malicious actions in the SDN network.

The SDN controller could query statistics to make sure that its flow table configurations are functioning as expected. This query could be malformed and actual statistics may not be the actual ones. The rogue switch could send inaccurate statistics for the SDN controller, and it does not have a way to know whether the statistics are correct or incorrect. The possibility to falsify statistics can open a path for other alterations of traffic in the SDN network.

Another purpose of the identification of the statistics packets is to enable packet tampering. Tampering the values inside the packets transporting statistics would create a situation where a rogue switch could send falsified values inside the statistics packets towards the SDN controller. The aim of tampered packets is to complicate and obfuscate the monitoring of the traffic what is going on between certain switches, which would enable lying to the SDN controller. The SDN controller does not currently have a way or tool to handle the statistics. The handling means evaluating the accuracy of the packet payload and taking actions based on inaccuracies of the SDN network traffic. The SDN controller

does not have means either to act upon if there was a mismatch in the packet counts; neither are there rules or instructions for the SDN controller how to handle the mismatches. Why does the statistics in generic need to be reliable? Because it is in the basis of networking to trust the traffic statistics, and the SDN controller blindly trusts the statistics response from a switch in the current SDN protocol implementation. One more purpose for a malicious intent could be that a switch sends packets to another destination than intended, possibly even to another tenant's network if possible.

The verification scenario expects that the attacker has gained access to the SDN network, especially on the switch. As an administrator, the traffic during the statistics query is observed and captured from the management server with a graphical desktop. The management server has an interface configured in the SDN network. To enable the traffic capture with the graphical user interface on the switch, SSH with X11 forwarding towards it as a root is commenced.

```
ssh -X <target>  
ssh -X root@172.16.0.104
```

The connection to switch is intentionally setup with in unsecure X11 forwarding as it enables the remote host to "access the local X11 display through the forwarded connection." (SSH Config Linux Man Page 2013.) The unsecure X11 forwarding also enables the possibility to do keystroke monitoring; however, this opportunity is not in the scope of this verification scenario. Using the insecure communications is also the worst-case scenario of poor security to enable attacks and shows that regardless of the solution considered very unsecure, it is in use still and could be exploited with malicious intent.

On the switch SW4, a query about port statistics is done with the command showing a snapshot of all ports statistics on the SW4.

```
ovs-ofctl dump-ports <netdev>  
ovs-ofctl dump-ports -O openflow13 of-switch
```

The figure below (Figure 21) is the output of `ovs-ofctl dump-ports`. It is used as a reference to find the correct packet in the traffic capture and possibly determine the payload of the packet as well. To see the payload in clear text would also help to determine that the data in the packet could be tampered with a suitable tool.

```

@sw4:~# ovs-ofctl dump-ports -0 openflow13 of-switch
OFPST_PORT reply (OF1.3) (xid=0x2): 6 ports
port 1: rx pkts=219207, bytes=13152818, drop=0, errs=0, frame=0, over=0, crc=
0
        tx pkts=2699503, bytes=269768582, drop=0, errs=0, coll=0
        duration=4482187.716s
port 4: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
        tx pkts=0, bytes=0, drop=0, errs=0, coll=0
        duration=4482187.718s
port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
        tx pkts=186969, bytes=14271122, drop=0, errs=0, coll=0
        duration=4482187.718s
port 2: rx pkts=1650815, bytes=226645964, drop=0, errs=0, frame=0, over=0, cr
c=0
        tx pkts=2175027, bytes=157612068, drop=0, errs=0, coll=0
        duration=4482187.717s
port 5: rx pkts=1069058, bytes=65273341, drop=0, errs=0, frame=0, over=0, crc
=0
        tx pkts=1993141, bytes=226340575, drop=0, errs=0, coll=0
        duration=4482187.721s
port 3: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
        tx pkts=0, bytes=0, drop=0, errs=0, coll=0
        duration=4482187.717s

```

Figure 21. Output of ovs-ofctl dump-ports -command.

The traffic is captured from the eth0 on the SW4. The packet info for the port statistics request-reply -pair is

*OFPT\_MULTIPART\_REQUEST* or  
*OFPT\_MULTIPART\_REPLY* for *OFPMP\_PORT\_STATS*

The request-reply pair was identified when monitoring the output capture while giving the ovs-ofctl command on the SW4.

Confirming the correct packet from the capture against command output is described next. The command output shows statistics for all ports on the SW4 (Figure 22 below).

```

@sw4:~# ovs-ofctl dump-ports -0 Openflow13 of-switch
OFPST_PORT reply (OF1.3) (xid=0x2): 6 ports
port 1: rx pkts=219297, bytes=13158218, drop=0, errs=0, frame=0, ov
tx pkts=2700420, bytes=269887460, drop=0, errs=0, coll=0
duration=4484023.530s
port 4: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0,
tx pkts=0, bytes=0, drop=0, errs=0, coll=0
duration=4484023.532s
port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, o
tx pkts=186969, bytes=14271122, drop=0, err
duration=4484023.532s
port 2: rx pkts=1651353, bytes=226732451, drop=0,
tx pkts=2175496, bytes=157649859, drop=
duration=4484023.531s
port 5: rx pkts=1069070, bytes=65274537, drop=0
tx pkts=1994136, bytes=2264636
duration=4484023.535s
port 3: rx pkts=0, bytes=0, drop=0, er
tx pkts=0, bytes=0, drop=0, er
duration=4484023.531s

```

Figure 22. Ovs-ofctl dump-ports command output.

The capture of network traffic shows two packets with info including PORT\_STATS, based on that a closer observation of the packets is done. The request part of the query is coming from the SDN controller towards the switch and the payload shows the details of the request packet's payload (Figure 23 below). The entire packet contents can be found in the Appendix 5 (Ovs-ofcli port statistics command capture, request packet, 1st query.).

Next is observed in detail the reply packet of the port statistics query. The figure (Figure 23) below shows the packet payload.

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REQUEST (18)
  Length: 24
  Transaction ID: 4256107
  Type: OFPMP_PORT_STATS (4)
  ▼ Flags: 0x0000
    .... 0 = OFPMPF_REQ_MORE: 0x0000
  Pad: 00000000
  Port number: OFPP_ANY (0xffffffff)
  Pad: 00000000

```

Figure 23. Statistics request packet payload.

The comparison of the ovs-ofcli command output and packet payload confirms that the output and packet payload with info OFPT\_MULTIPART\_REPLY, OFPMP\_PORT\_STATS are adequately uniform to make a conclusion that a certain command can be tied to a certain capture. The comparison was done in detail for the whole output and packet payload, here only described in detail for the port 1 for clarity. There are differences with values (blue boxes in Figure 24). The aim in this verification was to identify the correct packet and not investigate further the differences between transmit values. The second query verification cycle showed that the values are more uniform (Appendix 4 Ovs-ofcli port statistics command capture, request packet, 1st query). Later, in the subchapter 4.4.3 Lying to the SDN Controller, the results and a more careful packet inspection will prove that the statistics command output and capture are exactly uniform. To verify uniform values

```

▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 688
  Transaction ID: 4256107
  Type: OFPMP_PORT_STATS (4)
  ▶ Flags: 0x0000
  Pad: 00000000
  ▼ Port stats
    Port number: 1
    Pad: 00000000
    Rx packets: 219291
    Tx packets: 2700354
    Rx bytes: 13157858
    Tx bytes: 269878766
    Rx dropped: 0
    Tx dropped: 0
    Rx errors: 0
    Tx errors: 0
    Rx frame errors: 0
    Rx overrun errors: 0
    Rx CRC errors: 0
    Collisions: 0
    Duration sec: 4483885
    Duration nsec: 264000000
  ▶ Port stats
  ▶ Port stats
  ▶ Port stats
  ▶ Port stats
  ▶ Port stats

```

Figure 24. Port statistics reply packet payload.

requires going back chronologically in the long capture output to find the correct packet as the previously mentioned subchapter will describe.

Respectively, there is a possibility to view packet and bytes statistics for the flows. The finest granularity of the flow statistics can be reached by defining in or out port set to the flow rule. When defining the flow, for this command the flow translates into a specific flow entry field, such as `in_port` or `table`.

```
ovs-ofctl dump-flows <switch> <flow>
ovs-ofctl -O openflow13 dump-flows of-switch in_port=2
```

Without the flow definition, the output of command prints all the flow tables which have flow rules.

```
ovs-ofctl dump-flows <switch>
ovs-ofctl -O openflow13 dump-flows of-switch
```

To output the flow rules in a certain table, a flow parameter is defined within a table value pair. With default flow table this would result in the same output as the previous command.

```
ovs-ofctl dump-flows <switch> <flow>
ovs-ofctl -O openflow13 dump-flows of-switch table=0
```

With `dump-aggregate` command a summary on packets and bytes is printed for the respective port or table.

```
ovs-ofctl -O openflow13 dump-aggregate of-switch in_port=2
ovs-ofctl -O openflow13 dump-aggregate of-switch table=0
```

The packet types for the flow statistics are the same as for the port statistics in case a flow ID is given as `in_port` or `out_port` -value;

```
OFPT_MULTIPART_REQUEST and
OFPT_MULTIPART_REPLY for OFPMP_PORT_STATS
```

For the flow statistics, the packet request-reply -pairs are named as

```
OFPT_MULTIPART_REQUEST and
OFPT_MULTIPART_REPLY for OFPMP_FLOW
```

### 4.4.3 Lying to the SDN Controller

The main idea in this subchapter is to study the possibility to lie convincingly to an SDN controller. The objective of lying could be to obfuscate the SDN controller during data exfiltration by redirecting communication packets to other hosts than the intended ones. The purpose of this could be to modify packets in malicious intent and retransmit them back to the sender or relay them yet to another hosts. The main rationale is to effectively control switches or a subset of them without alerting the SDN controller monitoring capabilities.

If an attacker gains administrator access to one or several switches, there is a possibility to manipulate the flow tables and the rules in the flow entries, which gives the attacker full charge of the traffic sent via the switch. In this situation, the SDN controller is aware of the traffic rule changes. In addition to monitoring flow tables, the SDN controller can observe the port ingress and egress traffic on the switch (monitoring the traffic amount that pass through the interfaces). Based on the previous considerations, following assumptions are made:

- The SDN controller has a mechanism for interrogating the switches it controls, the statistics about the current traffic passing through
- The SDN controller is actively (at least at certain time intervals) sampling these statistics from the switches
- The monitoring that SDN controller does is performed somewhere above the it, for example by security applications residing on top of the Northbound APIs.
- Cross-referencing the traffic with the other switches it controls, to see if the traffic amounts that enter and leave the switches match the structure of the switch fabric
- Verifying whether the ingress and egress traffic statistics from the switches match each other. E.g. that the traffic statistics for Switch2, in the Figure, are matching that of Switches 1,3,4,5;
- As an attacker, one way to defeat this monitoring is then to convincingly lie to the SDN controller, to fool the monitoring, by giving credible traffic statistics to match as well as the possible values expected by the monitoring solution on top of the SDN controller.

In this work, the intention is not go as far as attack towards the SDN architecture, but merely try to prove that it is possible to actively lie to the SDN controller, given a compromised switch (or a set of them) within reasonable extend to prove malicious intentions on the compromised switch. Compromising a switch in the SDN network basically allows full management of the SDN controller as all the OpenFlow commands are available



via the switch. The SDN controller can be reached via SSH from the switch. If an attacker is able to configure their own management device in the SDN network, the SSH access is certainly available for all the SDN network devices as it is enabled by default.

The flow table modifications on a switch in the SDN architecture can be used as a form of lying to the SDN controller. The lying to the SDN controller can mean that in the flow table the destination of a certain packet is changed with flow rules. A specific protocol, port or IP address can be the variable in the flow rule (OpenFlow Switch Specification 1.3.5 2015, 63.) monitoring, the intention of which is to forward data to an unintended destination for tampering.

Making it harder for the SDN controller to fix possible errors in the SDN network traffic, creating a flow rule without proper ID can assist in that. The SDN controller cannot change a flow rule as it has a distinct identifier. By defining the flow rule with missing values, the attacker can ensure that the SDN controller cannot change the flow rule easily or at all. If the flow rule does not have a proper ID it cannot be easily removed because identifying the rule is impossible. Due to this incompatible rule, the removal of the whole default flow table or booting the SDN controller might be needed to resume to default behavior and default flow table.

The port statistics queries from a switch towards the SDN controller are clarified in section 4.4.2. Statistics Packets Identification. The switch sends a request to SDN controller for the statistics and gets a reply where the port statistics are in the packet payload. In this section, it is observed how the SDN controller is handling the request-reply -pair. The statistics request is sent from DLUX UI to query port statistics of SW4 (Figure 25 below). At the same time, the network capture is running to collect the communication between the two devices.

Node Connector Statistics for Node Id - openflow:4				
Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes
openflow:4:LOCAL	0	186969	0	14271122
openflow:4:1	235851	2878309	14151458	292408489
openflow:4:3	0	0	0	0
openflow:4:2	1753421	2267871	242874088	165022491
openflow:4:5	1077017	2180632	65884639	249367824
openflow:4:4	0	0	0	0

Figure 25. DLUX UI output of SW4 statistics query.

The corresponding reply-request -pair in the capture was identified based on the values shows in the DLUX UI for the openflow:4 device which is the SW4. The values were uniform. However, the latest port statistics packet capture chronologically was not the correct packet but the sixth when going back the capture. This clarifies the inaccuracy of values in subchapter 4.4.2 Statistics Packets Identification regarding the non-uniform values; the inspected packet was incorrect.

In the table below (Table 5), the command output in DLUX UI and packet capture is compared. It shows that the values are indeed consistent, there are no remaining packets within the calculations (command output minus packet capture values). The detailed reply packet data can be found in Appendix 9 (DLUX UI port statistics command reply capture, 3rd query).

Table 5. Command output and packet capture comparison, 3rd query.

			3rd query (Controller->Switch)			
			Rx packets	Rx bytes	Tx packets	Tx bytes
Port	1	Command output	235851	14151458	2878309	292408480
		Packet capture	235851	14151458	2878309	292408480
			0	0	0	0
Port	Local	Command output	-	-	186969	14271122
		Packet capture	-	-	186969	14271122
					0	0
Port	2	Command output	1753421	242874088	2267871	1655022491
		Packet capture	1753421	242874088	2267871	1655022491
			0	0	0	0
Port	5	Command output	1077017	65884639	2180632	249367842
		Packet capture	1077017	65884639	2180632	249367842
			0	0	0	0

Based on the capture data, it can be concluded that the request-reply pair has always the same direction: the SDN controller asks from a switch for the port statistics. This means that the SDN controller itself has no other means but to ask from the switch, it does not preserve values for comparison. This makes it harder for the SDN controller to know whether a switch lies to it or not about the statistics values. Based on the capture, the SDN controller makes the statistics queries frequently, not only by request (Appendix 9 Network capture during statistics query via DLUX UI, 3rd query). During the verification run, the capture is started and then in the DLUX UI the command is given. The actions take ca. 20 seconds. In this snapshot, the SDN controller sent altogether over 400 packets. Out of those, there were 19 statistics request-reply pairs. This highlights the continuous sending of port statistics queries by the SDN controller.

Now after confirming the statistics packet type and info, its contents and its accuracy, the next step is to enable lying to the controller. The next step is to modify statistics packet's destination in the flow table by a flow rule. The experiment will show if one or more rules are needed to implement the new destination forward.

The known information about the packet to be tampered by the attacker is the protocol type which is OpenFlow. The packet type is also known based on the detailed packet capture analysis in chapter 4.4.2 Statistics Packets Identification:

OFPT\_MULTIPART\_REQUEST or OFPT\_MULTIPART\_REPLY for OFPMP\_PORT\_STAT. It is yet to be evaluated which of the packets is to be re-directed, request or reply. When an SDN

controller sends a query about the statistics, should that query be forwarded to the attacker as a piece of information that the SDN controller is making such a request? Based on that request, the attacker would know that the reply packet needs to be modified to lie to the SDN controller, or that the reply sent should be tampered. But, does the attacker tamper all the packets? The assumption is made that the reply packets need to be modified. Otherwise when returning to the actual payload replies with different values than the modified ones, it may be more confusing and thus rise suspicion in case statistics are collected via the API for application purposes. Statistics collected via API would require an analysis and monitoring of the results, which is assumed to be the purpose of the statistics application, and hence, making a conclusion that all the statistics replies should be tampered starting from the takeover by the attacker.

The attacker would have generated or falsified the ongoing traffic; especially destination changes to another port and IP would create unexpected traffic values for the port statistics.

The SDN controller would be monitoring the output statistics of one switch to see that they match with the other switch's input statistics. Lying to the SDN controller in this scenario would occur in the way that the output statistics of SW4 port X towards the SW3 port X are tampered. Port Xs are the ports tied to the interfaces connecting the two switches in the SDN network.

To lie to the SDN controller, the flow table modifications can be done by defining the flow rules. The types of flow rule definitions can be adding, modifying or deleting. The flow rules are defined using the flow modification function (OpenFlow Switch Specification 1.3.5 2015, 43). When adding a flow rule, it can be dominant if not overlapping with other rules in the flow table. Overlap checking can also be disabled with flow modification flag `OFPPF_CHECK_OVERLAP` (Op. cit. 81). By setting the flag value to 1 it disables the checking of overlaps and adding new flow entries will overwrite the existing ones (Op. cit. 43). The attacker may use this feature to overwrite rules to make them work for malicious intents to obfuscate the SDN controller and provide inaccurate port statistic values.

The port statistics collection can be disabled by setting the flow modification flags `NO_PKT_COUNTS` and `NO_BYT_COUNTS` (OpenFlow Switch Specification 1.3.5 2015, 81). Resetting the counters can also work for malicious intents as the SDN controller will lose

the track of traffic traversing via the switches ports. Resetting the counters is one of the flow modification flags. It is called RESET\_COUNTS and when the flag is set, packet and byte counters for port statistics are cleared (Op. cit. 81). The attacker can lead on the SDN controller by resetting the counters and the SDN controller will not have any objections to that as it trusts the configurations and settings coming from the presumable reliable switch.

Managing the packet destination based on the packet info, i.e. changing the packet destination based on the OFPT\_MULTIPART\_REQUEST or OFPT\_MULTIPART\_REPLY values, is experimented next. The aim is to tamper the packet by changing the reply payload, i.e. statistics values. The packets can be altered with a suitable tool, such as Scapy. Editcap tool can be used to create new traffic from the existing capture files.

By defining a certain packet number, the Editcap tool usually removes the packet (Sharpe n.d.); however, now the intention is just to save specific packets to be re-transmitted to the SDN network and observe the impacts. With the parameter -r the packet selection can be reversed, and the defined packets will be saved to output file to be sent (Op. cit.) in the figure below (Figure 26).

```
editcap -r <infile> <outfile> <packet#-packet#>
editcap -r dumpFlowsByFlowTable.pcapng 2port_stats_chop.pcapng 295-296
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.0.15	172.16.0.104	OpenFlow	92	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS
2	0.000178000	172.16.0.104	172.16.0.15	OpenFlow	756	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS

Figure 26. Contents of a new packet capture file created from a previous capture.

Before sending the packets, the identification of re-transmitted packets needs to be ensured. With parameter -a, a comment string can be added to a packet with certain frame number (Sharpe n.d.). Comment string is used to identify the correct packets in the new capture made during the sending of the generated capture.

```
editcap -a <1:comment> -a <2:comment> <infile> <outfile>
editcap -a 1:mod_request -a 2:mod_reply 2port_stats.pcapng_
2port_stats_comment.pcapng
```



The payload of the packet is modified next with the Scapy tool. First, the contents of the packet are read into variable *rawload* (Figure 28 below).

```
>>> rawload=p[0].getlayer(Raw).load
>>> rawload
'\x04\x12\x00\x18\x00[E\x9d\x00\x04\x00\x00\x00\x00\x00\xff\xff\xff\xff\x00\x00\x00\x00'
```

Figure 28. Load payload to a variable *rawload* in scapy.

The modification of the payload is done by assigning a new value to the *rawload* variable (Figure 29 below). The output of the variable is as modified.

```
>>> rawload="Modified Request"
>>> rawload
'Modified Request'
```

Figure 29. Modify payload with Scapy.

The changes are saved with write function, *wrpcap* writes files in pcap format. Variable *p* for packet contains the original packet with payload changes.

```
wrpcap("<path>",<data>)
>>> wrpcap("/home/administrator/2port_stats_modload.pcap",p)
```

The newly created pcap file with modified packet payload can be opened in Scapy with Wireshark (<data>) command.

```
>>> wireshark(p)
```

The packet could not be opened with Wireshark capture tool for further inspection due to an error notification (Wireshark tool pop-up error notification): "The capture file appears to be damaged or corrupt. (libpcap: IrDA capture has a packet with an invalid sll\_protocol field)". This bug has been identified and according to Scapy's Exported Linux Cooked-mode, The Capture Doesn't Open in Wireshark (2015) and there are restrictions in usage with Wireshark and IP pcap files:

*Please remember that Wireshark works with Layer 2 packets (usually called "frames"). So an Ether() header was added to the ICMP packets. Passing just IP packets (layer 3) to Wireshark will give strange results.*

This finding can be applied to TCP packets as well. There is a workaround to create an Ethernet header artificially to the packet (Op. cit.). These further actions to add the Ethernet header to TCP packets are not in the scope of the master's thesis, as the theory of modifying the packets can be adequately confirmed with previous example.

The SDN controller may have difficulties in deleting a flow rule if it has been originally defined with inadequate parameters. Making the removal hard or inconvenient with generic flow rules is also a way to restrict SDN controller's ability to delete a flow rule. To increase these hardships on the SDN controller side, no new flow table is created for the verification but the default flow table is used to add flow rules. The Openflow protocol allows adding flow rules without a cookie ID. In that case, the cookie will get a default value 0x0. There can be many rules with that same cookie, and if other fields are also generic it is difficult to direct the delete actions to a certain flow rule. The protocol also allows to define flows without priority field. It is one of the distinct identifiers in a flow rule. If it is not defined, the flow rule is difficult to delete without proper identifiers or distinct fields which would separate it from other flow rules.

In the worst-case scenario, the whole flow table with the misconfigured flow rule needs to be deleted to resume normal functionality. If the flow rule is in default flow table, usually flow table 0, the deletion would require to delete it. After deleting the default flow table there would be no flow tables. How would that affect the SDN network? To see the effects of deleting the default flow table, a trial run is executed and the consequences of removal are reviewed.

On the SW4 Ping is started towards the SDN controller to see the possible connection disconnects after deleting the default flow table 0. Network traffic capture on the switch is also started to monitor the traffic in the SDN network in detail. The flow table modification command with delete option is run on the switch. The command is for deleting a flow is according to Open vSwitch Manual 2.6.90 (n.d., chapter OpenFlow Switch Flow Table Commands):



```
ovs-ofctl del-flows <bridge> <flow>  
ovs-ofctl del-flows -O openflow13 of-switch 0
```

All the flows can be deleted by not defining the flow parameter (Op. cit.). There are no other flow tables defined currently but still the parameter 0 is given to the ovs-ofctl command to make sure of the deletion target. Defining a specific flow for the command does not succeed with the OpenFlow version 2.3.0 on the Open vSwitch. Hence adjusting the command and leaving out the flow definition. Now the command is successful and there are no flow tables defined any longer. This was verified with the dump-flows command. During the scenario, the ping did not stop running. The default flow table had very permissive flow rules hence the removal of it did not make an effect on the connectivity. In the network traffic capture, there was not anything out of ordinary. The query regarding the flows before executing the deletion showed the flows as expected and vice versa after deletion there were no payload in the flow table replies regarding the flow entries. Somehow disturbing is however the fact that communication between the SDN controller and switch continues in a very permissive manner and the OpenFlow commands can still be executed.

#### 4.5 Analysis of Verification

The purpose of this chapter is to summarize the feasibility of compromising the SDN controller. Compromising the SDN controller via a switch is possible in theory. Directly compromising the SDN controller does not necessarily require a more complex configuration than the first phase configuration: a host could be directly connected to a switch which is connected to Internet (RGCE). If an attacker gains the IP address of any networking device in the SDN architecture, it would also lead to the possibility of compromising the interface towards the SDN controller.

Compromising the SDN controller is not required to gain extensive management permissions and possibilities in an SDN network. A switch configured in the SDN network can add, modify and delete flow rules from a flow table without any specific reaction by an SDN controller. Deleting a default flow table did not make disturbance in the simple host reachability traffic with ping. But it was alarming at the same time as the traffic was not

affected; the networking continues without the SDN controller managing it with flow table rules.

If an attacker gains access to the switch, the access to see the Control Channel IP of the switch is possible. With `ovs-ofctl` CLI the attacker can request SDN controller connection details and see the SDN controller Control Channel IP. When this IP is available, then there are again attack possibilities towards the SDN controller. The attacker may experiment with password cracking tools to find service related credentials. Also, the web server running on the SDN controller and its provided DLUX UI is available for the attacker. If the administrator neglects to change default password, the attacker can gain full access to the SDN controller via DLUX UI. The default password is common knowledge in the configuration guidelines and thus, easily available and also the address of web service is included in the guidelines.

## 5 Threat Mitigation Conclusions

This chapter proposes actions to take before a vulnerability becomes a threat. The mitigations are evaluated based on the research and verification outcome of this master's thesis, which also addresses the actions to take to mitigate the possible threats when they may already have taken effect.

### 5.1 Secure by Design

The SDN architecture has possibilities to improve security, and security should be addressed already in the design phase. Security could be built-in by default. As discussed in chapter 2.4 Security in SDN, the basic security hardening features should be enforced. The transport protocol should be encrypted by default with TLS, and the encryption decision should not be left to the network administrator make. There is no question if the security is needed; as the OpenFlow Switch Specification 1.3.5 (2015, 36) states, it is needed.

The highly valuable features of SDN architecture such as centralized management and programmable API may pose severe threats. In the centralized management, a compromised SDN controller may make the whole network vulnerable and affect its functionality. The programmable API interface may leave open interfaces for an attacker to exploit. For these possible vulnerabilities, the security enhanced extensions of SDN, FortNox

and SE-Floodlight, present solutions which should be built-in for all SDN distributions. The role based authentication can have multiple layers, which by priority secure and allow access for different actors in SDN architecture. For human actors accessing the SDN controller settings, the highest level of priority in authentication is given. An application implemented with programmable API may have secure or in-secure actor status, and based on that, the priority of authentication is set to medium or low. The priority is then considered in the context of conflict resolution scheme by evaluating the effects of actions with regard to higher priority actor actions (Porras et al. 2012, 3). The low priority actors could be restricted outside the SDN controller process (SDN Security Suite 2015). Role-based authentication with layers according to FortNOX Secure SDN extension could be included in design phase.

The security enhanced SDN architecture solutions provide security extensions. There should not be extensions but built-in security features, which are enforced and the usage of SDN network is to be restricted without those security features.

## 5.2 Computer and Operating System Security

In this chapter computer and OS security aspects are considered as possibilities to harden the security of SDN architecture. SDN architecture itself has possibilities to be secure to a certain degree:

*If an attacker has found a host and its vulnerabilities, then it is likely that the flow created for the attack was successful (i.e. the attack comes from a trusted source). In this situation, SDN has little to do since its main operations are on the lower layers (2 and 3). (Abdalla 2014, 24.)*

The virtual devices in SDN architecture in this research are based on Linux distros and so the OS security facet is concentrating on Linux OS hardening possibilities. The basis of secure OS is patching OS, security hardening, additional security controls and testing according to Scarfone, Jansen & Tracy (2008b, 25). Out-of-date OS modules may contain vulnerabilities which can be exploited by attackers. Also, in the patching or update context the removal of unnecessary services, applications and network protocols and configuration of OS user authentication are a vital part of secure configuration of a device (Scarfone et al. 2008b, 26). Additional security controls at minimum should include anti-malware SW, Intrusion Detection (IDS) and Intrusion Prevention (IPS), firewalls and patch management

(Scarfone et al. 2008b, 30). With these tools, possible attacks are detected, their impacts are mitigated and prevented in an up-to-date device. Testing these security measures gives proof of their implementation rationale and also reveals possible additional security measure needs in case of faults found during testing or simulated attacks.

### 5.3 OpenFlow Protocol Security Hardening

The OpenFlow protocol has clearly good intentions on leaving possibilities for implementation to affect the system configurations quite extensively. The unfortunate outcome of these chances of making administrative tasks easier or stressing the SDN system less, is making the solution vulnerable.

The usage of non-supporting TLS SDN controller allows unauthenticated access. There should be no option for configuring the SDN network without TLS enabled (Openflow Protocol Library:User Guide n.d.). The enabling of secure communications does not require many steps: key certificate generation or using exemplary keys, uncommenting the TLS-tag in the OpenFlow protocol library configuration file and configuring the virtual machine to use those generated or exemplary keys. Public and private key pairs can be created with `ovs-pki` command for the SDN controller and switches (OpenDaylight OpenFlow Plugin: TLS Support n.d.). The chain of trust consists of intermediate PKCS12 SDN controller key store and switch key store containing private and public keys. The key stores are made available in OpenDaylight installation folder (Op. cit.). Next the OpenFlow protocol library configuration file is modified as discussed earlier in this paragraph. Now the system should be ready for secure communication.

The failures in configuration commands and queries may be caused by OpenFlow version mismatch. The administrator needs to be fully aware of the version of resources in SDN network, otherwise no configurations take effect. The handshake message is a feature request from the SDN controller towards a switch. The handshake states the SDN controller what the switch's basic functionalities are (OpenFlow Switch Specification 1.3.5. 2015, 76). Without a successful handshake, there will be a handshake failure which prevents communication session initiation in the Control Channel between the SDN controller and a switch.

When setting up the communication between the SDN controller and a switch, the mode of recovering from the connection interrupt may make the SDN network vulnerable. “In fail-standalone mode, the switch owns the flow tables and flow entries” (OpenFlow Switch Specification 1.3.5. 2015, 174). When the parameter is not defined when configuring the communication between the SDN controller and the switch with command *ovs-ofctl set controller*, the default value, i.e. the fail secure mode is a safer option. With the default parameter, the switch tries to reconnect to the SDN controller in a connection failure situation. During the reconnect the packets towards the SDN controller are dropped and flow entries expire gracefully after their flow entries timeouts have passed. In the fail standalone mode, the switch starts to act as a legacy switch and has total control over the flow tables. (Op. cit. 38.)

The configuration error possibility is present and can cause vulnerabilities when enabling features. When modifying flows, there are potential ways to affect the SDN networking negatively. Hard timeout is one parameter of flow modification actions. *Hard\_timeout* defines along with *idle\_timeout* how long a flow entry keeps flow rules active (OpenFlow Switch Specification 1.3.5. 2015, 80). Setting the *hard\_timeout* to zero means that the flow entry is permanent (Op. cit. 21) and in case of SDN controller disconnect the unnecessary or malicious flow entry may keep on forwarding traffic.

The administrator’s level of technical knowledge is vital to keep up-to-date, and security policy may help with enforcing to plan and implement secure configurations.

The enumeration of flow modifications flags shows possibilities in configuring flows without collecting statistics for packets or bytes. Also, creating overlapping flow entries may be possible if the check for the feature *CHECK\_OVERLAP* is disabled. A malicious user may take advantage of *RESET\_COUNTS* and obfuscate the SDN controller by resetting a counter for packets and bytes. The counters for packets (*NO\_PKTS\_COUNT*) and bytes (*NO\_BYT\_COUNTS*) in the switch can be disabled, and when the SDN controller requests for statistics, there is none. (OpenFlow Switch Specification 1.3.5. 2015, 81.)

## 5.4 Layers of Security

There are many possibilities to carry out security attacks. The ways can be considered as layers depending on the different attack vectors reaching different levels of the system. The

following layered security example refers to Shenk (2013, 5). An authorized usage of assets remains as the main control target. However, there are often privilege issues overlooked regardless of continuous addressing of the issue. A basic authorization assurance such as a default password and strong password usage and missing privilege levels, i.e. groups, is still often dismissed.

The secure configuration on networking devices, laptops and mobile etc. devices requires security hardenings to be done. The level of device hardening should start already from the boot sector by enabling disk encryption to prevent unauthorized access to data and hardening the boot loader with authentication. The Operating System level security hardening can be reached by actively taking care of patching, disabling unnecessary services and user account, and also making sure that only authorized processes are running in the system. The physical ports and networking interfaces prevent unauthorized access with adequate authentication.

Previously only two layers were discussed in detail, and multiple items were already identified to address in layered security. While layers are expected to slow down the attacker, monitoring and thus detecting the attack on different layers is crucial. Detection may come afterwards; however, if it can be done in reasonable timeframe after the attack, the attacker steps may still be available, for example in the logs.

## 5.5 Physical Security

However virtualized and software-defined, SDN architecture runs on physical HW resources. The HW resources should be protected with layers of security. According to Hobbs (2016), it is good to consider the security perimeter concept in order to build protective guards around the possibly vulnerable target: site perimeter, building perimeter, building interior and the target itself. All the layers together aim at preventing unauthorized access to target and also a theft or injure of the hedged item.

The threats towards the hedged item, in this case the SDN architecture, may consist of physical tampering of the networking devices or breaking cables between the devices. Hence, the cables should be protected and preferably the SDN architecture related devices stored behind locked doors if physically available.

The physical locks may also hinder malware installations via external devices. In case of an external device attached to device, the BIOS settings may help in preventing to run the malware. In the BIOS settings, the booting from and running external devices can be disabled (Protect Your Information from Physical Threats n.d, chapter Software and Settings Related to Physical Security). Another layer of security in the SDN ensemble can be added by defining secure password for BIOS as well.

## 5.6 Testing and Evaluating the Security

A testing and evaluation conducted in continuous or regular intervals ensure the functionality of the security controls.

*(Security) Testing involves hands-on work with systems and networks to identify security vulnerabilities and can be executed across an entire enterprise or on selected systems. (Scarfone, Souppaya, Cody & Orebaugh 2008a, 15).*

The vulnerability scanning and penetration testing tools are used in security testing. They will indicate possible vulnerabilities in a controlled environment that could be exploited by attackers. Testing also gathers input to evaluate the compliance of security hardening actions such as patch management and password policy according to Scarfone et al. (2008a, 15).

## 6 Discussion

The discussion chapter gathers the tenets learned during the research and verification phases of this master's thesis. While going back to the beginning of the research, there was only an interest to learn more about the newish networking technology. Today, when the master's thesis is closing to its end in the form of this discussion, a lot more is known about the topic. The master's thesis research has deepened and built a solid networking knowledge base. Also many questions have risen, or those could be called further research ideas; they will be addressed in the later chapter 6.5. Future Research.

Where does the ultimate feeling of understanding a subject or its fundamentals come from? Frail sensations of it in the course of the research may suggest that it seems to be curiosity and prowess to question the purpose of a statement or a solution in this case. A

solution refers here to a feature, and the courage in questioning a feature in the OpenFlow protocol which could leave the architecture vulnerable.

When considering the identified threats by Slavov et al. (2016), to start off the threat landscape is wide and the legacy networking threats apply to the SDN solutions as well. The identified threats are on a higher level than which this master's thesis concentrated on. The OpenFlow protocol implementation and configuring the SDN network had possibilities to make the SDN solution secure, and also it gave a chance to make it vulnerable. The latter possibility was exploited in this master's thesis' verification scenarios.

The vulnerability exploits started already in setting up the control channel between the SDN controller and a switch. The secure communication settings with TLS were disregarded and switch connectivity recovery actions were disabled. These unsecure configurations were used because it was possible. There were also ways to make the SDN flow table and its rules difficult to modify or remove. That possibility was utilized when predicting the attack vectors an attacker may have and in preventing the attacker to exploit them. The details of flow table configurations are discussed later in chapter 6.2 OpenFlow Protocol Exploitability. The discussion especially targets to the Lying to the SDN Controller -scenario aspect.

The SDN controller trusts the replies it receives from a switch. Hence the attack vector of compromising the SDN controller was prospective via a switch in the SDN solution. If an attacker gains access to a switch, there is no question in compromising the SDN controller. It will be possible as they have communication channel configured between the SDN controller and switch(es). The communication channel is especially vulnerable if the TLS is not configured for it. The trust mentioned in the beginning of this paragraph also refers to the fact that is not necessary to compromise the controller as it is enough to compromise a switch. The switch was capable of running OpenFlow commands via CLI, those commands are the same as SDN controller uses to manage the SDN network.

The essence of the SDN architecture being the centralized control, it may enable compromising the SDN controller. That then may endanger the functionality of the entire network. Programmability via an API enables the implementation of innovative applications; however, it also leaves an open interface for the malicious applications.



The attacker mindset understanding was used in ethical hacking during the verification phase. The purpose was to find fixes for the possible vulnerabilities, not to exploit them. Also, the ethical hacking was made use of by trying to prevent the exploits by considering mitigations for the them.

This research may have useful input for the SDN Test Bed which was an outcome of the Cyber Trust project's earlier phase. The SDN Test Bed administrators could benefit from the actual usage experience of a non-simulated SDN network, usage of the OpenDaylight controller, its enhanced features in GUI and its ability to provide actual networking administration. The GUI features may not yet provide all the configuration possibilities available, but it can be concluded that the GUI is stable enough for its current intended usage. The YANG UI part of the GUI is unfortunately still relying on RESTful API commands which are unnecessarily complex to administer the network. The topology view, however, is especially informative providing an overview of the SDN network's networking devices and hosts connected to those networking devices. The OpenFlow implementation in the Open vSwitches (version 2.3. 0), in the form of CLI, is more versatile for the network administration than the GUI at the moment.

To answer the questions whether the SDN controller is vulnerable, yes, it is. It trusts too much in the communication with a switch in the SDN solution. A switch can use the OpenFlow protocol commands in similar manner as an SDN controller as was observed in the Lying to the Controller scenario and especially in its flow configuration part. How serious those SDN vulnerabilities are? Very serious in the hand of a malicious user. The malicious user has a way to manage the SDN network entirely with the OpenFlow commands on a switch. Mitigations to these vulnerabilities can be done with secure design, security hardenings for the OS, processes and SW and OpenFlow protocol hardenings as was outlined in chapter 5 Threat Mitigation Conclusions.

In the following chapters, the most intriguing findings regarding the vulnerabilities are discussed.

## 6.1 Unsecure Default Settings

The SSH is enabled by default, and this becomes an issue if an attacker gains access to the SDN network. However, the credentials need to be cracked prior to the usage of SSH communications but it is assumed to be fundamentally easy for an experienced attacker.

In the architecture, there are no security features enabled by default. The packets are transmitted unencrypted in the SDN network. There is a security module available; however, as it is not mandatory or default module in the distro, an administrator may decide not to configure it. The reasons for not configuring the security module could be ignorance, difficulty or plain malicious intent due to grudge or possible financial gain.

## 6.2 OpenFlow Protocol Exploitability

In this master's thesis, the different aspect in finding threats was to drill down the SDN switch and OpenFlow protocol specification and trying to find possible ways to exploit their features. Suitable features regarding the flow statistics in exploiting them in malicious intent are CHECK\_OVERLAP, RESET\_COUNTS, NO\_PKT\_COUNTS, NO\_BYT\_COUNTS in the ofp\_flow\_mod\_flags enumeration. The aim in configuring these is to confuse the SDN controller by not checking overlapping flow entries, resetting counters during active networking and not collecting packet or byte count of a flow entry. Similar obfuscating configurations can additionally be done by not updating the status of flows in case of expiration or deletion. That can be done by setting the flag for OFPFF\_SEND\_FLOW\_REM to zero. (OpenFlow Switch Specification 1.3.5 2015, 81.)

## 6.3 Mitigation

The security starts with careful planning of the security needs and policies and from engagement to provide security to all the personnel. "Many server security and performance problems can be traced to a lack of planning or management controls." (Scarfone et al. 2008a, 17.) When planning the security policies, information storage, services, user privileges and authentication, traffic filtering, skilled personnel and physical security aspects are considered. In a company, there should be clear division of responsible persons and responsibilities on the security functions implementing the security policies.

## 6.4 Networking Change Effects

The future of networking will include many more devices (IoT and related sensors), applications and machine-to-machine connections needing resources. The networking bandwidth requirements will increase as there will be multi-vendor multi-device multi-network distributed environment requirements.

## 6.5 Future Research

In the future research, especially the attack scenarios could be further studied. With more intense concentration on a specific scenario, an attack would be expected to succeed. The success in attacking the SDN network would complement this research greatly.

Implementing and experimenting with a statistics application would bring alive the original research dilemma on how the lying to the SDN controller actually goes. This master's thesis' results could be utilized in that research to tackle the already known obstacles.

The enablers of unsecure SDN network, especially the administrative configurations, flow and statistics collection findings could be used in the further security enhancements of the SDN protocol and architecture.

An interesting research aspect would also be to investigate the resource restrictions and experiment how the memory strain could affect the related processes. For example, if memory is low on a virtual resource, how well will the Open vSwitch process on a switch handle that situation and will it terminate spontaneously and what are the effects of that action. By defining and setting the thresholds to monitor the memory usage and acting upon those threshold events would complement the resource strain scenario as well.

## References

- 2013-10 Security Bulletin (CVE-2013-6013). 2013. Junos: SRX flowd core due to buffer overflow while processing telnet protocol. Vulnerability Database. Juniper Networks. Referenced 17.6.2016.  
<https://kb.juniper.net/InfoCenter/index?page=content&id=JSA10594&actp=search>
- Abdalla, T. 2014. Software-Defined Networking and its Security. Master's Thesis. Aalto University Library. <https://aaltodoc.aalto.fi/handle/123456789/14388>
- Content Management Systems Security and Associated Risks. 2013. Alert Database. US-CERT. Referenced 1.12.2016. <https://www.us-cert.gov/ncas/alerts/TA13-024A>
- Cormode, G. & Krishnamurthy, B. 2008. Key differences between Web 1.0 and Web 2.0. <http://www.ojphi.org/ojs/index.php/fm/article/view/2125/1972>
- CVE Cisco NX-OS Security Vulnerabilities. N.d. Vulnerability Database. Common Vulnerabilities and Exposures CVE. Referenced 15.7.2016.  
[https://www.cvedetails.com/vulnerability-list/vendor\\_id-16/product\\_id-17908/Cisco-Nx-os.html](https://www.cvedetails.com/vulnerability-list/vendor_id-16/product_id-17908/Cisco-Nx-os.html)
- Cyber Operation Environment – RGCE. N.d. Web publication. Jyväskylä, Finland: Jyväskylä Security Technology. Referenced 12.6.2016. <http://jyvsectec.fi/en/cyber-environment/>
- Cyber Trust Project Plan. 2016. Confidential project plan. Digile.
- Dierks, T. & Rescorla, E. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. IETF Trust. Referenced 30.9.2016. <https://tools.ietf.org/pdf/rfc5246.pdf>
- Digile, about us. N.d. Web publication. Digile. Referenced 15.8.2016.  
<http://digile.fi/en/about-us/>
- Digile, IoT Management. N.d. Web publication. Digile. Referenced 15.8.2016.  
<http://www.internetofthings.fi/program.html>
- Digile, Need for Speed. N.d. Web publication. Digile. Referenced 15.8.2016.  
<http://www.n4s.fi/en/>
- DIMECC Cyber Trust Program. N.d. Web publication. DIMECC, Digile. 14.12.2016.  
<http://cybertrust.fi/>
- DIMECC Data to Intelligence. N.d. Web publication. DIMECC, Digile 13.12.2016.  
<http://www.datatointelligence.fi/>
- Dokulil, J., Tykal, J., Yaghob, J. & Zavoral, F. 2007. Semantic Web Infrastructure. IEEE. <https://janet.finna.fi>
- Feamster, N., Rexford, J. & Zegura, E. 2013. The Road to SDN: An Intellectual History of Programmable Networks. <https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf>
- Fei, H. 2014. Network Innovation through OpenFlow and SDN: Principles and Design. Boca Raton, Florida, USA; CRC Press. <https://janet.finna.fi>

Gartner Top 10 Technology Trends 2016. 2016. Web publication. Gartner. 14.11.2016.  
<http://www.gartner.com/newsroom/id/3143521>

HP SDI for the Future of the Data Center. 2015. Web publication. Hewlett-Packard.  
Referenced 15.9.2016. <http://www.slideshare.net/HewlettPackardEnterprise/hp-softwaredefined-infrastructure-for-the-future-of-the-data-center>

Internet Security Threat Report. 2016. Web publication. Symantec. 15.10.2015.  
<https://www.symantec.com/security-center/threat-report>

Longbottom, C. 2015. Software defined everything: Throwing more software into the mix.  
Web publication. Computer Weekly. Referenced 14.9.2016.  
<http://www.computerweekly.com/feature/Software-defined-everything-Throwing-more-software-into-the-mix>

Lyne, J. 2014. Security Threat Trends 2015. Web publication. Sophos. Referenced 12.8.2016.  
<https://www.sophos.com/threat-center/medialibrary/PDFs/other/sophos-trends-and-predictions-2015.pdf>.

MAC Flood Buffer Overflow Attack. 2012. Online Guideline. Break stuff majorly. Referenced 6.10.2016. <http://breakstuffmajorly.blogspot.fi/2012/05/mac-flood-buffer-overflow-attack.html>

Mahler, D. 2014. OpenFlow with multiple Flow Tables. Video publication. Referenced 30.8.2016. <https://www.youtube.com/watch?v=TD5wmoD7XOE>

Matthews, M. N.d. Are You Ready for Software-Defined Everything? Web publication. Wired.com. Referenced 10.10.2016. <https://www.wired.com/insights/2013/05/are-you-ready-for-software-defined-everything/>

Open vSwitch Manual, ovs-ofctl, version 2.5.90. N.d. Online Manual. Open vSwitch. Referenced 28.6.2016. <http://openvswitch.org/support/dist-docs/ovs-ofctl.8.txt>

OpenDaylight DLUX:DLUX Karaf Feature. N.d. Online Guideline. OpenDaylight Wiki. Referenced 11.7.2016.  
[https://wiki.opendaylight.org/view/OpenDaylight\\_DLUX:DLUX\\_Karaf\\_Feature](https://wiki.opendaylight.org/view/OpenDaylight_DLUX:DLUX_Karaf_Feature)

OpenDaylight Features List. N.d. Online Guideline. OpenDaylight. Referenced 14.7.2016.  
<https://www.opendaylight.org/opendaylight-features-list>

OpenDaylight OpenFlow Plugin: TLS Support. N.d. Online Guideline. OpenDaylight.org. 15.10.2015.  
[https://wiki.opendaylight.org/view/OpenDaylight\\_OpenFlow\\_Plugin:\\_TLS\\_Support#Create\\_.26\\_Sign\\_private.2Fpublic\\_key\\_certificates](https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:_TLS_Support#Create_.26_Sign_private.2Fpublic_key_certificates)

Openflow Protocol Library:User Guide. N.d. Online Guideline. OpenDaylight.org. Referenced 15.10.2015.  
[https://wiki.opendaylight.org/view/Openflow\\_Protocol\\_Library:User\\_Guide#Configuration](https://wiki.opendaylight.org/view/Openflow_Protocol_Library:User_Guide#Configuration)

OpenFlow Switch Specification, Version 1.3.5. 2015. Palo Alto, CA, USA: ONF Open Networking Foundation. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.5.pdf>

OpenFlow Switch Specification, Version 1.5.1. 2015. Palo Alto, CA, USA: ONF Open Networking Foundation. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf>

Our cybersecurity predictions for 2016. 2015. Web publication. Sophos. 2.9.2016. <https://blogs.sophos.com/2015/12/11/our-cybersecurity-predictions-for-2016/>

OWASP Top 10. 2013. Online Guideline. OWASP. Referenced 2.9.2016. [https://www.owasp.org/images/f/f8/OWASP\\_Top\\_10\\_-\\_2013.pdf](https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf)

Pakzad, F., Portmann, M., Wee Lum, T. & Indulska, J. 2014. Efficient Topology Discovery in Software Defined Networks. Brisbane, Australia: School of ITEE, The University of Queensland. IEEE. <https://janet.finna.fi>

Porras, P., Shinz, S., Yegneswarany, V., Fongy, M., Tysony, M. & Guz, G. 2012. A Security Enforcement Kernel for OpenFlow Networks. SRI International. Referenced 30.9.2016. <http://www.csl.sri.com/users/vinod/papers/fortnox.pdf>

Principles and Practices for Securing Software-Defined Networks. Version 1.3.4. 2015. Online Guideline. Palo Alto, CA, USA: ONF Open Networking Foundation. Referenced 30.9.2016. [https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Principles\\_and\\_Practices\\_for\\_Securing\\_Software-Defined\\_Networks\\_applied\\_to\\_OFv1.3.4\\_V1.0.pdf](https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Principles_and_Practices_for_Securing_Software-Defined_Networks_applied_to_OFv1.3.4_V1.0.pdf)

Protect your information from physical threats. N.d. Web publication. Security in a box. Referenced 22.11.2016. <https://securityinabox.org/en/guide/physical>

Scapy's Exported Linux Cooked-mode Capture Doesn't Open in Wireshark. 2015. Online Guideline. Stackoverflow.com. Referenced 15.11.2016. <http://stackoverflow.com/questions/27831427/scapys-exported-linux-cooked-mode-capture-doesnt-open-in-wireshark>

Scarfone, K., Souppaya, M., Cody, A. & Orebaugh, A. 2008a. Technical Guide to Information Security Testing and Assessment. Gaithersburg, MD, USA: National Institute of Standards and Technology (NIST). <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf>

Scarfone, K., Jansen, W. & Tracy, M. 2008b. Guide to general server security. Gaithersburg, MD, USA: National Institute of Standards and Technology (NIST). <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-123.pdf>

SDN and Flow vulnerabilities. N.d. Web publication. Common Vulnerabilities and Exposures. Referenced 17.6.2016. <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=sdn+flow>

SDN Architecture. 2014. Palo Alto, CA, USA: ONF Open Networking Foundation. Referenced 15.9.2016. [https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR\\_SDN\\_ARCH\\_1.0\\_06062014.pdf](https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf)

SDN Security Suite. 2015. Web publication. OpenFlowSec.org. Referenced 30.9.2016. <http://www.openflowsec.org/SDNSuite.html>

Shadbolt, N., Hall, W. & Berners-Lee, T. 2006. The semantic web revisited. IEEE. <https://janet.finna.fi>

- Sharpe, R. N.d. Editcap Man Page. Online manual. Referenced 10.11.2016.  
<https://www.wireshark.org/docs/man-pages/editcap.html>
- Shenk, J. 2013. Layered Security: Why it Works? SANS Institute Infosec Reading Room. Referenced 12.6.2016. <https://www.sans.org/reading-room/whitepapers/analyst/layered-security-works-34805>
- Slavov, K., Miche, Y. & Schneider, P. 2016. An Analysis of Current Software Defined Networking Threats. Espoo, Finland: Cyber Trust Consortium, Nokia and Ericsson Security Research.
- Software-Defined Networking (SDN) Definition. N.d. Web publication. Palo Alto, CA, USA: Open Networking Foundation (ONF). Referenced 2.9.2016.  
<https://www.opennetworking.org/sdn-resources/sdn-definition>
- SSH Config Linux Man Page. 2013. Online manual. BSD. Referenced 21.10.2016.  
[https://linux.die.net/man/5/ssh\\_config](https://linux.die.net/man/5/ssh_config)
- Stanford University OpenFlow Wiki. 2010. Wiki publication. Referenced 12.8.2016.  
<http://yuba.stanford.edu/cs244wiki/index.php/Overview>
- State Machine. N.d. Flowgrammable. Referenced 14.12.2016.  
<http://flowgrammable.org/sdn/openflow/state-machine/>
- van Adrichem, N. L. M., Doerr, C. & Kuipers F. A. 2014. OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks. Delft, The Netherlands: IEEE. <https://janet.finna.fi>
- Vulnerability Summary for CVE-2007-2282. 2013. Vulnerability Database. National Cyber Awareness System. Referenced 17.6.2016.  
<https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-2282>
- Vulnerability Summary for CVE-2012-4122. 2014. Vulnerability Database. National Cyber Awareness System. Referenced 18.11.2016.  
<https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-4122>
- Vulnerability Summary for CVE-2016-1329. 2016. Vulnerability Database. National Vulnerability Database NVD. Referenced 15.7.2016.  
<https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-1329>
- Wetherall, D., Gutttag, J. & Tennenhouse, D. 1998. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. IEEE. <https://janet.finna.fi>
- What is Kali Linux?. 2016. Web publication. Offensive Security. Referenced 22.8.2016.  
<http://docs.kali.org/introduction/what-is-kali-linux>
- What is SDN? N.d. Web publication. SDxCentral. Referenced 14.9.2016.  
<https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/>
- What is Software Defined Compute? N.d. Web publication. SDx Central. Referenced 14.9.2016. <https://www.sdxcentral.com/sdn/definitions/what-is-software-defined-compute/>

## Appendices

### Appendix 1. REST API Commands

#### SW4 port2 where a host is connected to:

<http://172.16.0.15:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:4/node-connector/openflow:4:2/opendaylight-port-statistics:flow-capable-node-connector-statistics>

#### With authentication with *curl* command:

`curl -i -u admin:admin -X GET http://172.16.0.15:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:4/node-connector/openflow:4:2/opendaylight-port-statistics:flow-capable-node-connector-statistics`

### Appendix 2. Statistics collection scripts.

The script (`get_stats.sh`) is used to collect statistics from all the ports of a switch (Figure 30). The relevant data is parsed with additional script, `parse_stats.sh`. Output file is `all_ports.stats_$(date)`.

```
#!/bin/bash -x
i=0
date=`date +%Y-%m-%d_%H:%M:%S`
while [ $i -lt 100 ]
do
    echo $i
    ovs-ofctl -O openflow13 dump-ports of-switch >> all_ports.stats_$(date)
    i=$(( $i + 1 ))
    echo $i
done
```

Figure 30. Statistics collection script.

`Parse_stats.sh` script (below Figure 31) requires a port as parameter to parse correct port statistics. Output filename of `get_stats.sh` is given as a parameter and also the port which the statistics are gone thru in detail, LOCAL or numerical port.



```

#!/bin/bash -x
date=`date +%Y-%m-%d_%H:%M:%S`

if [ "$1" = "" ] || [ "$2" = "" ];
then
    echo "Give filename and port as a parameter for the script."
    exit
fi

if [ "$2" = "LOCAL" ];
then
    cat $1 | grep -A 2 "$2" | grep "rx pkts" | awk -F '=' '{print $2}' | awk
-F ',' '{print $1}' > rx
    cat $1 | grep -A 2 "$2" | grep "tx pkts" | awk -F '=' '{print $2}' | awk
-F ',' '{print $1}' > tx
    paste rx tx | column -s $'\t' -t > summary_port$2_$date
else
    cat $1 | grep -A 2 "port $2" | grep "rx pkts" | awk -F '=' '{print $2}'
| awk -F ',' '{print $1}' > rx
    cat $1 | grep -A 2 "port $2" | grep "tx pkts" | awk -F '=' '{print $2}'
| awk -F ',' '{print $1}' > tx
    paste rx tx | column -s $'\t' -t > summary_port$2_$date
fi

```

Figure 31. parse\_stats.sh script.

### Appendix 3. Ovs-ofcli command output for port statistics and network capture comparison

			1st query				2nd query			
			Rx packets	Rx bytes	Tx packets	Tx bytes	Rx packets	Rx bytes	Tx packets	Tx bytes
Port	1	Command output	219297	13158218	2700420	299887460	231849	13911338	2835562	287013065
		Packet capture	212291	13157858	2700354	269878766	231849	13911338	2835566	287013405
			7006	360	66	30008694	0	0	-4	-340
Port	Local	Command output	-	-	186969	14271122	-	-	186969	14271122
		Packet capture	-	-	186969	14271122	-	-	186969	14271122
					0	0			0	0
Port	2	Command output	1652353	226732451	1069070	157649859	1728907	239011984	2245636	163249051
		Packet capture	1651314	226726052	2175463	157647203	1728909	239012154	2245638	163249221
			1039	6399	-1106393	2656	-2	-170	-2	-170
Port	5	Command output	1069070	652745537	1994064	226454603	1075186	65745489	2135714	243871430
		Packet capture	1069070	652745537	1994136	226454603	1075186	65745489	2135718	243871770
			0	0	-72	0	0	0	-4	-340

#### Appendix 4. Ovs-ofcli port statistics command capture, request packet, 1<sup>st</sup> query.

No.	Time	Source	Destination	Protocol	Length	Info
141	12.011973000	172.16.0.15	172.16.0.104	OpenFlow	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS

Frame 141: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0

Interface id: 0 (eth0)

Encapsulation type: Ethernet (1)

Arrival Time: Oct 21, 2016 11:02:22.298288000 EEST

[Time shift for this packet: 0.000000000 seconds]

Epoch Time: 1477036942.298288000 seconds

[Time delta from previous captured frame: 0.003754000 seconds]

[Time delta from previous displayed frame: 0.003754000 seconds]

[Time since reference or first frame: 12.011973000 seconds]

Frame Number: 141

Frame Length: 90 bytes (720 bits)

Capture Length: 90 bytes (720 bits)

[Frame is marked: False]

[Frame is ignored: False]

[Protocols in frame: eth:ethertype:ip:tcp:openflow:openflow\_v4]

[Coloring Rule Name: TCP]

[Coloring Rule String: tcp]

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Destination: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Address: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

....0. .... = LG bit: Globally unique address (factory default)

....0. .... = IG bit: Individual address (unicast)

Source: Vmware\_01:22:4a (00:50:56:01:22:4a)

Address: Vmware\_01:22:4a (00:50:56:01:22:4a)

Address: Vmware\_01:22:4a (00:50:56:01:22:4a)

....0. .... = LG bit: Globally unique address (factory default)

....0. .... = IG bit: Individual address (unicast)

Type: IP (0x0800)

Internet Protocol Version 4, Src: 172.16.0.15 (172.16.0.15), Dst: 172.16.0.104 (172.16.0.104)

Version: 4

Header Length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))

0000 00.. = Differentiated Services Codepoint: Default (0x00)

....00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)

Total Length: 76

Identification: 0x4261 (16993)

Flags: 0x02 (Don't Fragment)

0... .... = Reserved bit: Not set

.1.. .... = Don't fragment: Set

..0. .... = More fragments: Not set

Fragment offset: 0

Time to live: 64

Protocol: TCP (6)

Header checksum: 0x9fb3 [validation disabled]

[Good: False]

[Bad: False]

Source: 172.16.0.15 (172.16.0.15)

Destination: 172.16.0.104 (172.16.0.104)

[Source GeolIP: Unknown]

[Destination GeolIP: Unknown]

Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 44823 (44823), Seq: 2014, Ack: 31321,  
Len: 24

Source Port: 6633 (6633)  
Destination Port: 44823 (44823)  
[Stream index: 0]  
[TCP Segment Len: 24]  
Sequence number: 2014 (relative sequence number)  
[Next sequence number: 2038 (relative sequence number)]  
Acknowledgment number: 31321 (relative ack number)  
Header Length: 32 bytes

.... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)  
000. .... .... = Reserved: Not set  
...0 .... .... = Nonce: Not set  
... 0... .... = Congestion Window Reduced (CWR): Not set  
... .0.. .... = ECN-Echo: Not set  
... ..0. .... = Urgent: Not set  
... ...1 .... = Acknowledgment: Set  
... .... 1... = Push: Set  
... .... .0.. = Reset: Not set  
... .... ..0. = Syn: Not set  
... .... ...0 = Fin: Not set

Window size value: 1452  
[Calculated window size: 1452]  
[Window size scaling factor: -1 (unknown)]

Checksum: 0xd8df [validation disabled]  
[Good Checksum: False]  
[Bad Checksum: False]

Urgent pointer: 0

Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps  
No-Operation (NOP)

Type: 1  
0... .... = Copy on fragmentation: No  
.00. .... = Class: Control (0)  
...0 0001 = Number: No-Operation (NOP) (1)

No-Operation (NOP)  
Type: 1  
0... .... = Copy on fragmentation: No  
.00. .... = Class: Control (0)  
...0 0001 = Number: No-Operation (NOP) (1)

Timestamps: TSval 360922630, TSecr 1120897422  
Kind: Time Stamp Option (8)  
Length: 10  
Timestamp value: 360922630  
Timestamp echo reply: 1120897422

[SEQ/ACK analysis]  
[Bytes in flight: 24]

OpenFlow 1.3  
Version: 1.3 (0x04)  
Type: OFPT\_MULTIPART\_REQUEST (18)  
Length: 24  
Transaction ID: 4256107  
Type: OFPMP\_PORT\_STATS (4)

Flags: 0x0000  
.... .... .... ..0 = OFPMPF\_REQ\_MORE: 0x0000  
Pad: 00000000  
Port number: OFPP\_ANY (0xffffffff)  
Pad: 00000000

## Appendix 5. Ovs-ofcli statistics command capture, reply packet, 1<sup>st</sup> query.

The output of statistics command 1<sup>st</sup> query is in text, in subchapter 4.4.2 Statistics Packets Identification.

```
No.   Time      Source      Destination  Protocol Length Info
     142 12.012147000 172.16.0.104 172.16.0.15  OpenFlow 754  Type:
OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS
```

```
Frame 142: 754 bytes on wire (6032 bits), 754 bytes captured (6032 bits) on interface 0
Interface id: 0 (eth0)
Encapsulation type: Ethernet (1)
Arrival Time: Oct 21, 2016 11:02:22.298462000 EEST
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 1477036942.298462000 seconds
[Time delta from previous captured frame: 0.000174000 seconds]
[Time delta from previous displayed frame: 0.000174000 seconds]
[Time since reference or first frame: 12.012147000 seconds]
Frame Number: 142
Frame Length: 754 bytes (6032 bits)
Capture Length: 754 bytes (6032 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:tcp:openflow:openflow_v4]
[Coloring Rule Name: TCP]
[Coloring Rule String: tcp]
Ethernet II, Src: Vmware_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware_01:22:4a (00:50:56:01:22:4a)
Destination: Vmware_01:22:4a (00:50:56:01:22:4a)
Address: Vmware_01:22:4a (00:50:56:01:22:4a)
....0. .... = LG bit: Globally unique address (factory default)
....0 .... = IG bit: Individual address (unicast)
Source: Vmware_01:1b:3d (00:50:56:01:1b:3d)
Address: Vmware_01:1b:3d (00:50:56:01:1b:3d)
....0. .... = LG bit: Globally unique address (factory default)
....0 .... = IG bit: Individual address (unicast)
Type: IP (0x0800)
Internet Protocol Version 4, Src: 172.16.0.104 (172.16.0.104), Dst: 172.16.0.15 (172.16.0.15)
Version: 4
Header Length: 20 bytes
Differentiated Services Field: 0xc0 (DSCP 0x30: Class Selector 6; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
1100 00.. = Differentiated Services Codepoint: Class Selector 6 (0x30)
....00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
Total Length: 740
Identification: 0x5e21 (24097)
Flags: 0x02 (Don't Fragment)
0... .... = Reserved bit: Not set
.1.. .... = Don't fragment: Set
..0. .... = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (6)
Header checksum: 0x809b [validation disabled]
[Good: False]
[Bad: False]
Source: 172.16.0.104 (172.16.0.104)
Destination: 172.16.0.15 (172.16.0.15)
[Source GeolP: Unknown]
```

```

[Destination GeolIP: Unknown]
Transmission Control Protocol, Src Port: 44823 (44823), Dst Port: 6633 (6633), Seq: 31321, Ack: 2038,
Len: 688
  Source Port: 44823 (44823)
  Destination Port: 6633 (6633)
  [Stream index: 0]
  [TCP Segment Len: 688]
  Sequence number: 31321 (relative sequence number)
  [Next sequence number: 32009 (relative sequence number)]
  Acknowledgment number: 2038 (relative ack number)
  Header Length: 32 bytes
  .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    ... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 1... = Push: Set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
  Window size value: 1444
  [Calculated window size: 1444]
  [Window size scaling factor: -1 (unknown)]
  Checksum: 0x5b6e [validation disabled]
    [Good Checksum: False]
    [Bad Checksum: False]
  Urgent pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    No-Operation (NOP)
      Type: 1
        0... .... = Copy on fragmentation: No
        .00. .... = Class: Control (0)
        ...0 0001 = Number: No-Operation (NOP) (1)
    No-Operation (NOP)
      Type: 1
        0... .... = Copy on fragmentation: No
        .00. .... = Class: Control (0)
        ...0 0001 = Number: No-Operation (NOP) (1)
    Timestamps: TSval 1120897423, TSecr 360922630
      Kind: Time Stamp Option (8)
      Length: 10
      Timestamp value: 1120897423
      Timestamp echo reply: 360922630
  [SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 141]
    [The RTT to ACK the segment was: 0.000174000 seconds]
    [Bytes in flight: 688]
OpenFlow 1.3
  Version: 1.3 (0x04)
Type: OFPT_MULTIPART_REPLY (19)
  Length: 688
  Transaction ID: 4256107
  Type: OFPMP_PORT_STATS (4)
  Flags: 0x0000
    .... .... .... ..0 = OFPMPF_REQ_MORE: 0x0000
  Pad: 00000000
  Port stats

```

Port number: 1  
Pad: 00000000  
Rx packets: 219291  
Tx packets: 2700354  
Rx bytes: 13157858  
Tx bytes: 269878766  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4483885  
Duration nsec: 264000000

Port stats

Port number: 4  
Pad: 00000000  
Rx packets: 0

Rx bytes: 0

Tx bytes: 0  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4483885  
Duration nsec: 266000000

Port stats

Port number: OFPP\_LOCAL (0xffffffe)  
Pad: 00000000  
Rx packets: 0  
Tx packets: 186969  
Rx bytes: 0  
Tx bytes: 14271122  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4483885

Duration nsec: 266000000

Port stats

Port number: 2  
Pad: 00000000  
Rx packets: 1651314  
Tx packets: 2175463  
Rx bytes: 226726052  
Tx bytes: 157647204  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0

Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4483885  
Duration nsec: 265000000

## Port stats

Port number: 5  
Pad: 00000000  
Rx packets: 1069070  
Tx packets: 1994064  
Rx bytes: 65274537  
Tx bytes: 226454603  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
    Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4483885  
Duration nsec: 269000000

## Port stats

Port number: 3  
Pad: 00000000  
Rx packets: 0  
Tx packets: 0  
Rx bytes: 0  
Tx bytes: 0  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4483885  
Duration nsec: 265000000

Appendix 6. Ovs-ofcli port statistics command output, 2<sup>nd</sup> query.

```
@sw4:~# ovs-ofctl dump-ports -0 Openflow13 of-switch
OFPST_PORT reply (OF1.3) (xid=0x2): 6 ports
port 1: rx pkts=231849, bytes=13911338, drop=0, errs=0, frame=0, over=0, crc=0
      tx pkts=2835562, bytes=287013065, drop=0, errs=0, coll=0
      duration=4741383,602s
port 4: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
      tx pkts=0, bytes=0, drop=0, errs=0, coll=0
      duration=4741383,604s
port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
      tx pkts=186969, bytes=14271122, drop=0, errs=0, coll=0
      duration=4741383,604s
port 2: rx pkts=1728907, bytes=239011984, drop=0, errs=0, frame=0, over=0, crc=0
      tx pkts=2245636, bytes=163249051, drop=0, errs=0, coll=0
      duration=4741383,603s
port 5: rx pkts=1075186, bytes=65745489, drop=0, errs=0, frame=0, over=0, crc=0
      tx pkts=2135714, bytes=243871430, drop=0, errs=0, coll=0
      duration=4741383,607s
port 3: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
      tx pkts=0, bytes=0, drop=0, errs=0, coll=0
      duration=4741383,603s
```



## Appendix 7. Ovs-ofcli port statistics command capture, 2<sup>nd</sup> query.

No.	Time	Source	Destination	Protocol	Length	Info
160	15.511742000	172.16.0.104	172.16.0.15	OpenFlow	754	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS

### Packet comments

Frame 160: 754 bytes on wire (6032 bits), 754 bytes captured (6032 bits) on interface 0  
 Interface id: 0 (eth0)  
 Encapsulation type: Ethernet (1)  
 Arrival Time: Oct 24, 2016 10:34:07.652404000 EEST  
 [Time shift for this packet: 0.000000000 seconds]  
 Epoch Time: 1477294447.652404000 seconds  
 [Time delta from previous captured frame: 0.000171000 seconds]  
 [Time delta from previous displayed frame: 0.000171000 seconds]  
 [Time since reference or first frame: 15.511742000 seconds]  
 Frame Number: 160  
 Frame Length: 754 bytes (6032 bits)  
 Capture Length: 754 bytes (6032 bits)  
 [Frame is marked: False]  
 [Frame is ignored: False]  
 [Protocols in frame: eth:ethertype:ip:tcp:openflow:openflow\_v4]  
 [Coloring Rule Name: TCP]  
 [Coloring Rule String: tcp]

Ethernet II, Src: Vmware\_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware\_01:22:4a (00:50:56:01:22:4a)  
 Destination: Vmware\_01:22:4a (00:50:56:01:22:4a)  
 Address: Vmware\_01:22:4a (00:50:56:01:22:4a)  
 ....0. .... = LG bit: Globally unique address (factory default)  
 ....0 .... = IG bit: Individual address (unicast)  
 Source: Vmware\_01:1b:3d (00:50:56:01:1b:3d)  
 Source: Vmware\_01:1b:3d (00:50:56:01:1b:3d)  
 Address: Vmware\_01:1b:3d (00:50:56:01:1b:3d)  
 ....0. .... = LG bit: Globally unique address (factory default)  
 ....0 .... = IG bit: Individual address (unicast)

Type: IP (0x0800)

Internet Protocol Version 4, Src: 172.16.0.104 (172.16.0.104), Dst: 172.16.0.15 (172.16.0.15)  
 Version: 4  
 Header Length: 20 bytes  
 Differentiated Services Field: 0xc0 (DSCP 0x30: Class Selector 6; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))  
 1100 00.. = Differentiated Services Codepoint: Class Selector 6 (0x30)  
 ....00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)  
 Total Length: 740  
 Identification: 0xae3 (60131)  
 Flags: 0x02 (Don't Fragment)  
 0... .... = Reserved bit: Not set  
 .1.. .... = Don't fragment: Set  
 ..0. .... = More fragments: Not set  
 Fragment offset: 0  
 Time to live: 64  
 Protocol: TCP (6)  
 Header checksum: 0xf3d8 [validation disabled]  
 [Good: False]  
 [Bad: False]  
 Source: 172.16.0.104 (172.16.0.104)  
 Destination: 172.16.0.15 (172.16.0.15)  
 [Source GeoIP: Unknown]  
 [Destination GeoIP: Unknown]

Transmission Control Protocol, Src Port: 44823 (44823), Dst Port: 6633 (6633), Seq: 38961, Ack: 2621, Len: 688

Source Port: 44823 (44823)  
 Destination Port: 6633 (6633)  
 [Stream index: 0]  
 [TCP Segment Len: 688]  
 Sequence number: 38961 (relative sequence number)  
 [Next sequence number: 39649 (relative sequence number)]  
 Acknowledgment number: 2621 (relative ack number)  
 Header Length: 32 bytes  
 .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)  
 000. .... .... = Reserved: Not set  
 ...0 .... .... = Nonce: Not set  
 .... 0... .... = Congestion Window Reduced (CWR): Not set  
 .... .0.. .... = ECN-Echo: Not set  
 .... ..0. .... = Urgent: Not set  
 .... ...1 .... = Acknowledgment: Set  
 .... .... 1... = Push: Set  
 .... .... .0.. = Reset: Not set  
 .... .... ..0. = Syn: Not set  
 .... .... ...0 = Fin: Not set  
 Window size value: 1444  
 [Calculated window size: 1444]  
 [Window size scaling factor: -1 (unknown)]  
 Checksum: 0x5b6e [validation disabled]  
 [Good Checksum: False]  
 [Bad Checksum: False]

Urgent pointer: 0

Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps  
 No-Operation (NOP)  
 Type: 1  
 0... .... = Copy on fragmentation: No  
 .00. .... = Class: Control (0)  
 ...0 0001 = Number: No-Operation (NOP) (1)  
 No-Operation (NOP)  
 Type: 1  
 0... .... = Copy on fragmentation: No  
 .00. .... = Class: Control (0)  
 ...0 0001 = Number: No-Operation (NOP) (1)  
 Timestamps: TSval 1185273762, TSecr 425298935  
 Kind: Time Stamp Option (8)  
 Length: 10  
 Timestamp value: 1185273762  
 Timestamp echo reply: 425298935  
 [SEQ/ACK analysis]  
 [This is an ACK to the segment in frame: 159]  
 [The RTT to ACK the segment was: 0.000171000 seconds]  
 [Bytes in flight: 688]

OpenFlow 1.3

Version: 1.3 (0x04)  
 Type: OFPT\_MULTIPART\_REPLY (19)  
 Length: 688  
 Transaction ID: 5097194  
 Type: OFPMP\_PORT\_STATS (4)  
 Flags: 0x0000  
 .... .... .... 0 = OFPMPF\_REQ\_MORE: 0x0000  
 Pad: 00000000  
 Port stats  
 Port number: 1

Pad: 00000000  
Rx packets: 231849  
Tx packets: 2835566  
Rx bytes: 13911338  
Tx bytes: 287013405  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4741390  
Duration nsec: 618000000

## Port stats

Port number: 4  
Pad: 00000000  
Rx packets: 0  
Tx packets: 0  
Rx bytes: 0  
Tx bytes: 0  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4741390  
Duration nsec: 620000000

## Port stats

Port number: OFPP\_LOCAL (0xffffffe)  
Pad: 00000000  
Rx packets: 0  
Tx packets: 186969  
Rx bytes: 0  
Tx bytes: 14271122  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0

Rx overrun errors: 0

Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4741390  
Duration nsec: 620000000

## Port stats

Port number: 2  
Pad: 00000000  
Rx packets: 1728909  
Tx packets: 2245638  
Rx bytes: 239012154  
Tx bytes: 163249221  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0

Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4741390  
Duration nsec: 619000000

Port stats

Port number: 5  
Pad: 00000000  
Rx packets: 1075186  
Tx packets: 2135718  
Rx bytes: 65745489

Tx bytes: 243871770

Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4741390  
Duration nsec: 623000000

Port stats

Port number: 3  
Pad: 00000000  
Rx packets: 0  
Tx packets: 0  
Rx bytes: 0  
Tx bytes: 0  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4741390  
Duration nsec: 619000000

## Appendix 8. DLUX UI port statistics command reply capture, 3<sup>rd</sup> query.

```
No.   Time      Source      Destination  Protocol Length Info
 288 36.678984000 172.16.0.104 172.16.0.15  OpenFlow 754  Type:
OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS
```

```
Frame 288: 754 bytes on wire (6032 bits), 754 bytes captured (6032 bits) on interface 0
Interface id: 0 (eth0)
Encapsulation type: Ethernet (1)
Arrival Time: Oct 25, 2016 09:20:48.863171000 EEST
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 1477376448.863171000 seconds
[Time delta from previous captured frame: 0.000220000 seconds]
[Time delta from previous displayed frame: 0.000220000 seconds]
[Time since reference or first frame: 36.678984000 seconds]
Frame Number: 288
Frame Length: 754 bytes (6032 bits)
Capture Length: 754 bytes (6032 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:tcp:openflow:openflow_v4]
[Coloring Rule Name: TCP]
[Coloring Rule String: tcp]
Ethernet II, Src: Vmware_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware_01:22:4a (00:50:56:01:22:4a)
Destination: Vmware_01:22:4a (00:50:56:01:22:4a)
Address: Vmware_01:22:4a (00:50:56:01:22:4a)
....0. .... = LG bit: Globally unique address (factory default)
....0 .... = IG bit: Individual address (unicast)
Source: Vmware_01:1b:3d (00:50:56:01:1b:3d)
Address: Vmware_01:1b:3d (00:50:56:01:1b:3d)
....0. .... = LG bit: Globally unique address (factory default)
....0 .... = IG bit: Individual address (unicast)
Type: IP (0x0800)
Internet Protocol Version 4, Src: 172.16.0.104 (172.16.0.104), Dst: 172.16.0.15 (172.16.0.15)
Version: 4
Header Length: 20 bytes
Differentiated Services Field: 0xc0 (DSCP 0x30: Class Selector 6; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
 1100 00.. = Differentiated Services Codepoint: Class Selector 6 (0x30)
  ....00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)
Total Length: 740
Identification: 0x8082 (32898)
Flags: 0x02 (Don't Fragment)
 0... .... = Reserved bit: Not set
 .1.. .... = Don't fragment: Set
 ..0. .... = More fragments: Not set
Fragment offset: 0
Time to live: 64
Protocol: TCP (6)
Header checksum: 0x5e3a [validation disabled]
 [Good: False]
 [Bad: False]
Source: 172.16.0.104 (172.16.0.104)
Destination: 172.16.0.15 (172.16.0.15)
[Source GeoIP: Unknown]
[Destination GeoIP: Unknown]
Transmission Control Protocol, Src Port: 44823 (44823), Dst Port: 6633 (6633), Seq: 92441, Ack: 5577,
Len: 688
Source Port: 44823 (44823)
```

```

Destination Port: 6633 (6633)
[Stream index: 0]
[TCP Segment Len: 688]
Sequence number: 92441 (relative sequence number)
[Next sequence number: 93129 (relative sequence number)]
Acknowledgment number: 5577 (relative ack number)
Header Length: 32 bytes
.... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
  000. .... .... = Reserved: Not set
  ...0 .... .... = Nonce: Not set
  .... 0... .... = Congestion Window Reduced (CWR): Not set
  .... .0.. .... = ECN-Echo: Not set
  .... ..0. .... = Urgent: Not set
  .... ...1 .... = Acknowledgment: Set
  .... .... 1... = Push: Set
  .... .... .0.. = Reset: Not set
  .... .... ..0. = Syn: Not set
  .... .... ...0 = Fin: Not set
Window size value: 1444
[Calculated window size: 1444]
[Window size scaling factor: -1 (unknown)]
Checksum: 0x5b6e [validation disabled]
  [Good Checksum: False]
  [Bad Checksum: False]
Urgent pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  No-Operation (NOP)
    Type: 1
      0... .... = Copy on fragmentation: No
      .00. .... = Class: Control (0)
      ...0 0001 = Number: No-Operation (NOP) (1)
  No-Operation (NOP)
    Type: 1
      0... .... = Copy on fragmentation: No
      .00. .... = Class: Control (0)
      ...0 0001 = Number: No-Operation (NOP) (1)
  Timestamps: TSval 1205774065, TSecr 445799227
    Kind: Time Stamp Option (8)
    Length: 10
    Timestamp value: 1205774065
    Timestamp echo reply: 445799227
[SEQ/ACK analysis]
  [This is an ACK to the segment in frame: 287]
  [The RTT to ACK the segment was: 0.000220000 seconds]
  [Bytes in flight: 688]
OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 688
  Transaction ID: 5365034

Type: OFPMP_PORT_STATS (4)
  Flags: 0x0000
  .... .... .... ..0 = OFPMPF_REQ_MORE: 0x0000
  Pad: 00000000
  Port stats
    Port number: 1
    Pad: 00000000
    Rx packets: 235851

```

Tx packets: 2878309  
Rx bytes: 14151458  
Tx bytes: 292408489  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4823391  
Duration nsec: 829000000

## Port stats

Port number: 4  
Pad: 00000000  
Rx packets: 0  
Tx packets: 0  
Rx bytes: 0  
Tx bytes: 0  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0

Duration sec: 4823391

Duration nsec: 831000000

## Port stats

Port number: OFPP\_LOCAL (0xffffffe)  
Pad: 00000000  
Rx packets: 0  
Tx packets: 186969  
Rx bytes: 0  
Tx bytes: 14271122  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0

Duration sec: 4823391

Duration nsec: 831000000

## Port stats

Port number: 2  
Pad: 00000000  
Rx packets: 1753421  
Tx packets: 2267871  
Rx bytes: 242874088  
Tx bytes: 165022491  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0

Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4823391  
Duration nsec: 830000000

Port stats

Port number: 5  
Pad: 00000000  
Rx packets: 1077017  
Tx packets: 2180632  
Rx bytes: 65884639  
Tx bytes: 249367824  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4823391  
Duration nsec: 834000000

Port stats

Port number: 3  
Pad: 00000000  
Rx packets: 0  
Tx packets: 0  
Rx bytes: 0  
Tx bytes: 0  
Rx dropped: 0  
Tx dropped: 0  
Rx errors: 0  
Tx errors: 0  
Rx frame errors: 0  
Rx overrun errors: 0  
Rx CRC errors: 0  
Collisions: 0  
Duration sec: 4823391  
Duration nsec: 830000000



## Appendix 9. Network capture during statistics query via DLUX UI, 3<sup>rd</sup> query.

Here is a clip from the 30-page packet capture output to show the format of flow related packets, OFPMP -packets sample.

No.	Time	Source	Destination	Protocol	Length
14	0.674914	172.16.0.15	172.16.0.104	TCP	66

### Info

6633 → 44823 [ACK] Seq=496 Ack=7257 Win=1432 Len=0 TSval=445790226 TSecr=1205765063

Frame 14: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 496, Ack: 7257, Len: 0

No.	Time	Source	Destination	Protocol	Length
15	0.674923	172.16.0.104	172.16.0.15	OpenFlow	386

### Info

Type: OFPT\_MULTIPART\_REPLY, OFPMP\_TABLE

Frame 15: 386 bytes on wire (3088 bits), 386 bytes captured (3088 bits) on interface 0

Ethernet II, Src: Vmware\_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware\_01:22:4a (00:50:56:01:22:4a)

Internet Protocol Version 4, Src: 172.16.0.104, Dst: 172.16.0.15

Transmission Control Protocol, Src Port: 44823, Dst Port: 6633, Seq: 7257, Ack: 496, Len: 320

[3 Reassembled TCP Segments (6112 bytes): #12(2896), #13(2896), #15(320)]

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
16	0.681838	172.16.0.15	172.16.0.104	OpenFlow	82

### Info

Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_GROUP\_DESC

Frame 16: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 496, Ack: 7577, Len: 16

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
17	0.681884	172.16.0.104	172.16.0.15	OpenFlow	82

**Info**

Type: OFPT\_MULTIPART\_REPLY, OFPMP\_GROUP\_DESC

Frame 17: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0

Ethernet II, Src: Vmware\_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware\_01:22:4a (00:50:56:01:22:4a)

Internet Protocol Version 4, Src: 172.16.0.104, Dst: 172.16.0.15

Transmission Control Protocol, Src Port: 44823, Dst Port: 6633, Seq: 7577, Ack: 512, Len: 16

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
18	0.688011	172.16.0.15	172.16.0.104	OpenFlow	90

**Info**

Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_GROUP

Frame 18: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 512, Ack: 7593, Len: 24

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
19	0.688053	172.16.0.104	172.16.0.15	OpenFlow	82

**Info**

Type: OFPT\_MULTIPART\_REPLY, OFPMP\_GROUP

Frame 19: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0

Ethernet II, Src: Vmware\_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware\_01:22:4a (00:50:56:01:22:4a)

Internet Protocol Version 4, Src: 172.16.0.104, Dst: 172.16.0.15

Transmission Control Protocol, Src Port: 44823, Dst Port: 6633, Seq: 7593, Ack: 536, Len: 16

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
20	0.688841	172.16.0.15	172.16.0.104	OpenFlow	90

**Info**

Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_METER\_CONFIG

Frame 20: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 536, Ack: 7609, Len: 24

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
21	0.688881	172.16.0.104	172.16.0.15	OpenFlow	82

**Info**

Type: OFPT\_MULTIPART\_REPLY, OFPMP\_METER\_CONFIG

Frame 21: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0

Ethernet II, Src: Vmware\_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware\_01:22:4a (00:50:56:01:22:4a)

Internet Protocol Version 4, Src: 172.16.0.104, Dst: 172.16.0.15

Transmission Control Protocol, Src Port: 44823, Dst Port: 6633, Seq: 7609, Ack: 560, Len: 16

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
22	0.689692	172.16.0.15	172.16.0.104	OpenFlow	90

**Info**

Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_METER

Frame 22: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 560, Ack: 7625, Len: 24

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
23	0.689746	172.16.0.104	172.16.0.15	OpenFlow	82

**Info**

Type: OFPT\_MULTIPART\_REPLY, OFPMP\_METER

Frame 23: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0

Ethernet II, Src: Vmware\_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware\_01:22:4a (00:50:56:01:22:4a)

Internet Protocol Version 4, Src: 172.16.0.104, Dst: 172.16.0.15

Transmission Control Protocol, Src Port: 44823, Dst Port: 6633, Seq: 7625, Ack: 584, Len: 16

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
24	0.726402	172.16.0.15	172.16.0.104	TCP	66

**Info**

6633 → 44823 [ACK] Seq=584 Ack=7641 Win=1452 Len=0 TSval=445790239 TSecr=1205765067

Frame 24: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 584, Ack: 7641, Len: 0

No.	Time	Source	Destination	Protocol	Length
25	3.674709	172.16.0.15	172.16.0.104	OpenFlow	122

**Info**

Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_FLOW

Frame 25: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 584, Ack: 7641, Len: 56

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
26	3.674845	172.16.0.104	172.16.0.15	OpenFlow	826

**Info**

Type: OFPT\_MULTIPART\_REPLY, OFPMP\_FLOW

Frame 26: 826 bytes on wire (6608 bits), 826 bytes captured (6608 bits) on interface 0

Ethernet II, Src: Vmware\_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware\_01:22:4a (00:50:56:01:22:4a)

Internet Protocol Version 4, Src: 172.16.0.104, Dst: 172.16.0.15

Transmission Control Protocol, Src Port: 44823, Dst Port: 6633, Seq: 7641, Ack: 640, Len: 760

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
27	3.674978	172.16.0.15	172.16.0.104	TCP	66

**Info**

6633 → 44823 [ACK] Seq=640 Ack=8401 Win=1452 Len=0 TSval=445790976 TSecr=1205765814

Frame 27: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 640, Ack: 8401, Len: 0

No.	Time	Source	Destination	Protocol	Length
28	3.677495	172.16.0.15	172.16.0.104	OpenFlow	90

**Info**

Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_PORT\_STATS

Frame 28: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 640, Ack: 8401, Len: 24

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
29	3.677652	172.16.0.104	172.16.0.15	OpenFlow	754

**Info**

Type: OFPT\_MULTIPART\_REPLY, OFPMP\_PORT\_STATS

Frame 29: 754 bytes on wire (6032 bits), 754 bytes captured (6032 bits) on interface 0

Ethernet II, Src: Vmware\_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware\_01:22:4a (00:50:56:01:22:4a)

Internet Protocol Version 4, Src: 172.16.0.104, Dst: 172.16.0.15

Transmission Control Protocol, Src Port: 44823, Dst Port: 6633, Seq: 8401, Ack: 664, Len: 688

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
30	3.679190	172.16.0.15	172.16.0.104	OpenFlow	90

**Info**

Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_QUEUE

Frame 30: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 664, Ack: 9089, Len: 24

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
31	3.679232	172.16.0.104	172.16.0.15	OpenFlow	82

**Info**

Type: OFPT\_MULTIPART\_REPLY, OFPMP\_QUEUE

Frame 31: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0

Ethernet II, Src: Vmware\_01:1b:3d (00:50:56:01:1b:3d), Dst: Vmware\_01:22:4a (00:50:56:01:22:4a)

Internet Protocol Version 4, Src: 172.16.0.104, Dst: 172.16.0.15

Transmission Control Protocol, Src Port: 44823, Dst Port: 6633, Seq: 9089, Ack: 688, Len: 16

OpenFlow 1.3

No.	Time	Source	Destination	Protocol	Length
32	3.680218	172.16.0.15	172.16.0.104	OpenFlow	82

**Info**

Type: OFPT\_MULTIPART\_REQUEST, OFPMP\_TABLE

Frame 32: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface 0

Ethernet II, Src: Vmware\_01:22:4a (00:50:56:01:22:4a), Dst: Vmware\_01:1b:3d (00:50:56:01:1b:3d)

Internet Protocol Version 4, Src: 172.16.0.15, Dst: 172.16.0.104

Transmission Control Protocol, Src Port: 6633, Dst Port: 44823, Seq: 688, Ack: 9105, Len: 16

OpenFlow 1.3