

Anastasia Shirokova

SECURITY OF SOFTWARE-DEFINED NETWORKS

Denial-of-service attack detection and mitigation

Bachelor's thesis
Degree programme in Information Technology

2019



South-Eastern Finland
University of Applied Sciences

Author (authors)	Degree	Time
Anastasia Shirokova	Bachelor of Engineering	December 2019
Thesis title		40 pages
Security of software-defined networks Denial-of-service attack detection and mitigation		
Commissioned by		
South-Eastern Finland University of Applied Sciences		
Supervisor		
Matti Juutilainen		
Abstract		
<p>Software-defined networking (SDN) is a recently developed architecture. This design brings programmability and efficiency into traditional networks. Awareness of potential SDN specific threats and complications is essential information for developers and engineers who intent to implement this network design for their business.</p> <p>The main goal of this thesis was to research the expanding field of software-defined networking and the security of this design. Narrowing down this comprehensive area, this work covered the reliability of SDN controllers against denial-of-service (DoS) attacks. The objective was to answer two questions after the practical part was conducted: Is SDN less secure because of its architecture compared to traditional networks? Are DoS attacks a big threat for SDN architecture or can they be easily mitigated?</p> <p>In the theory part, the attack surface of SDN and the available security solutions for these networks were studied. The purpose of the practical part was to illustrate a DoS attack against an SDN controller and to show possible methods to detect such types of attacks in SDN environment. To achieve this, the SDN topology was simulated using GNS3 software and the behaviour of the controller was observed under the DoS attack.</p> <p>This research showed that the existing techniques can minimize the impact of denial-of-service attacks in SDN environments. Moreover, with correctly configured monitoring and proper network design, the threats can be detected instantly and mitigated accordingly. While it is certainly challenging to successfully maintain secure SDN settings and the right knowledge base is required, using this design over traditional networks does not bring additional risks.</p>		
Keywords		
Software-defined networking, network security, GNS3, OpenDaylight, denial-of-service attacks		

CONTENTS

1	INTRODUCTION	5
2	SOFTWARE-DEFINED NETWORKS.....	6
2.1	History	7
2.1.1	From traditional networks to SDN	7
2.1.2	Start of SDN.....	9
2.2	Architecture	10
2.2.1	Application plane	11
2.2.2	Northbound interface	12
2.2.3	Control plane	13
2.2.4	Southbound interface.....	14
2.2.5	Data plane	15
2.3	Use cases of SDN	16
2.4	Security of SDN	18
2.4.1	Attack surface	19
2.4.2	Control plane security	20
3	DENIAL OF SERVICE	20
3.1	History	21
3.2	Techniques	22
3.3	DoS of SDN	23
3.3.1	Detection and mitigation solutions	24
4	IMPLEMENTATION.....	25
4.1	Simulation environment setup.....	25
4.2	Simulation of software-defined network	26
4.3	Simulation of DoS traffic	30
4.4	Findings and results.....	31

5	THE FUTURE OF SDN	33
6	CONCLUSIONS	34
	REFERENCES	36

1 INTRODUCTION

Today businesses focus a lot on continuous integration and development to get faster software delivery, to reduce the application development life-cycle and to simplify the troubleshooting of bugs. DevOps and Agile are the most used methodologies to achieve this. (Veritis Group Inc. 2019.) Moreover, these characteristics can be improved using software-defined networking (SDN) design, and therefore, engineers are more likely to choose this architecture for their network structure (Riveners 2016).

A new approach for network management has been introduced with the emerging of SDN. It can be distinguished from traditional networks by the following: the separation of control and data planes, existing overview of a complete network, APIs to manage the network and an efficient and automated orchestration of the services. This architecture brings numerous benefits, such as programmability, flexibility and a centralized management. (Zinner et al. 2013.) However, this creates new security concerns and challenges too. The centralized controller being the single point of management in SDN makes it an attractive target for DoS attacks. If an attacker is able to compromise the controller, he will get control over the whole network, thus, proper security solutions should be applied to detect and mitigate denial-of-service attacks. (Dridi & Zhani 2016.)

SDN keeps growing in popularity even more than was predicted just five years ago. Figure 1 represents the growing revenue of SDN, that was expected to be 3.2 billion USD according to the forecast of International Data Corporation made in 2015 (Dargue 2015.) However, the recent report by Market Research Future (2019) shows that the actual market of SDN is 8.8 billion USD in 2018 and is expected to reach 28.9 billion USD by 2023.

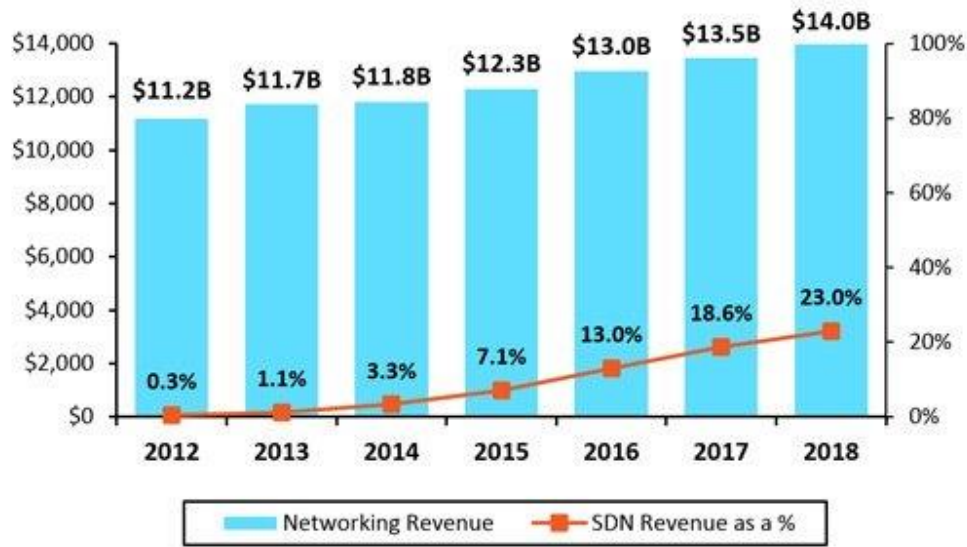


Figure 1. SDN and Enterprise networking revenue (Dargue 2015)

Chapter 2 of this thesis covers the structure of SDN, the OpenFlow and the business use cases for this network design. Moreover, this chapter provides an overview of the attack surface and the security solutions. Chapter 3 describes techniques of DoS attacks and their mitigation methods in SDN. In Chapter 4, a software-defined network is created in a virtual environment and a DoS attack is simulated to examine the performance of the controller. Furthermore, the results of the simulation are discussed. Finally, Chapter 5 and 6 summarize the project with the analysis of the future of SDN and the conclusions of this bachelor's thesis.

2 SOFTWARE-DEFINED NETWORKS

This section covers the background that is required to understand the concept of SDN. It starts with describing the initial demand in this approach from traditional networking perspective. Then, the design architecture is explained thoroughly, including application, control and data planes. Additionally, use cases that can be applied in the industry are defined together with the attack surface of SDN. Security of the control layer is outlined separately to gain more expertise required for the practical part of the thesis.

2.1 History

Network provisioning has not changed much for over 20 years. Speed and bandwidth have been noticeably enhanced, while in a nutshell the network kept the same structure and principles. The network settings are usually implemented in dedicated hardware, each piece of which is managed separately. This approach brings many difficulties in managing the systems, most importantly that it is unscalable, error-prone and not flexible.

Moreover, humanity could not predict the amount of traffic going through today's networks, and therefore, protocols in the TCP/IP model were not designed for the modern speed and amount of transmitted data. Eric Schmidt, CEO of Google in 2010, stated at the Techonomy conference that "there was five Exabytes of information created between the dawn of civilization through 2003, but that much information is now created every two days, and the pace is increasing" (Kirkpatrick 2010). If engineers could create the network protocols from scratch today, they would have developed them in a very different way.

After personal computers were introduced in the 1980s, the demand for the availability of resources increased. Thus, the operational part had to move to dedicated data centers (Bartels 2011). Separating client and server functionality brought improvements in system management and made it easier to distribute resources between users.

2.1.1 From traditional networks to SDN

The first significant change in computing was the virtualization of computer resources. In 2002, VMware introduced ESX Server which was software to run several operating systems on only one hosting OS, while allowing operating systems to be used as a normal file. This has decreased expenses and improved performance. (VMWare 2018.) Moreover, it gave the opportunity to allocate the resources matching specific needs and to move or clone the machines between physical locations by effortlessly copying files containing an OS. According to

Kaspersky Lab US (2012), server virtualization was implemented by 69% of companies in the US.

The demand for computing power was growing extremely fast and it required companies to plan their resource usage by ordering the required physical network equipment in advance. Expanding the physical devices on premises brought several problems. Granted that physical infrastructure uses energy, space and other resources, it often sat idle before the upcoming demand and was not utilized in an efficient way. (Miller 2016.) In 2006, a noticeable part in network evolution was played by Amazon. The company launched its cloud platform, Amazon Web Services (AWS), providing its overabundant computer resources to retail users. Additional network, computational power and storage could be accessed straight over the Internet. (Amazon Web Services 2019.) Furthermore, AWS offered elastic computing services that allow the clients to rent virtual machines for their own applications and specific needs. Cloud technologies introduced the opportunity to pay only for the usage and made IT infrastructures more scalable and flexible. (Bohm et al. 2011.)

Multitenant data centers were the next stage in the efficiency of network design. Initially, the challenge was to create an isolated environment for each client and, at the same time, to efficiently allocate the resources of data centers. (Chao 2016.) Multiprotocol Label Switching (MPLS) protocol was used to design overlay networks above source networks and to keep data flow at layer 2 of the OSI model. Additionally, it allowed maintaining MAC and IP addresses of each virtual machine correctly during its physical relocation and holding the machines externally accessible and routable. (Guo et al. 2014.)

Network equipment, such as switches and routers, had configuration interfaces, but their functionality was still limited by the vendors that did not allow access to low level settings. At that moment, control and data planes were located together in one device. For instance, this meant that routing protocols had to be distributed between all devices to communicate routing. (Nadeau & Gray 2013.)

2.1.2 Start of SDN

In 2007, the first implementation of the OpenFlow (OF) protocol was introduced within the campus network of Stanford University. The researchers defined several causes for need for this technology. Traditional network equipment was mostly proprietary, with the functionality of routers available and switches being inflexible and relatively limited for researching purposes. Furthermore, the incompatibility of devices from different vendors was a concern. The existing open-source platforms were not sufficient in terms of performance. Thus, it was suggested to create the OpenFlow, the communication protocol between switches operating with only the data plane and the controller managing them. (McKeown et al. 2008, 69-74.)

After receiving support from the research community, Open Networking Research Center (ONRC) was founded in Stanford and UC Berkeley Universities. As a part of ONRC, Open Networking Lab (ON.Lab) was developing open source SDN platforms and software. In 2011, Open Networking Foundation (ONF) was started as a non-profit organization with financial support from the leading IT companies. The aim of ONF was to advertise development of SDN and to contribute to the technology standards, particularly the OpenFlow. In 2017, ONF and ON.Lab were merged together to combine their goals and resources (Open Networking Foundation 2016).

In 2007, in addition to research contributions, Martin Casado, Nick McKeown and Scott Shenker started Nicira company that began the commercial development of software-defined networks. Nicira Network Virtualization Platform (NPV) and Distributed Virtual Network Infrastructure (DVNI) were developed as a new approach in virtual networks with the use of SDN. The company was growing very fast and had several big companies as clients, such as eBay, AT&T and DreamHost. (Park 2013.) In 2012, when Nicira was acquired by VMWare, NSX was launched. It became the first network with SDN approach and the security virtualization platform of VMWare. That was the first big transaction that demonstrated the separate market and demand for SDN solutions. (Bort 2014.)

Realizing the increased attention towards SDN, many companies decided to contribute and to implement the technology in their products. For instance, according to Burt (2012), in 2011, Cisco added OpenFlow to their Nexus switches and Juniper configured OpenFlow protocol to JunOS SDK. Further, the OpenDaylight Project was started in 2013 by the Linux Foundation whose main delivery is the SDN controller. (Knorr 2013.)

2.2 Architecture

Centralized management is the main distinguishing feature of software-defined networks compared to traditional network design. In this part, the general design structure and the details of each layer are covered. Furthermore, the communication between the layers is explained.

The architecture is comprehensively divided into three layers, sometimes mentioned as planes: application, control and data. Data layer is also referred to as an infrastructure or forwarding layer. This separation provides a general framework of this network design. Figure 2 shows a logical view of SDN architecture visualized by ONF (2012).

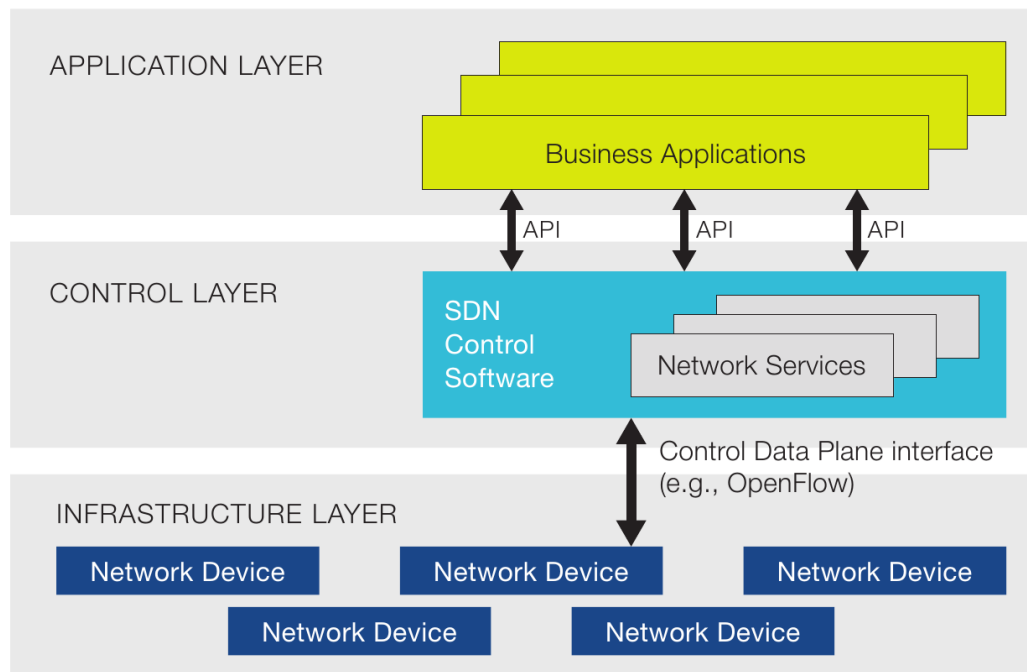


Figure 2. Software-defined network design (Open Networking Foundation 2012)

SDN combines applications that communicate with a controller using an API. The forwarding devices are separated from the control plane and get commands from the controller through the OpenFlow protocol. For comparison, control and data planes are combined in one device in traditional networks. (Nadeau & Gray 2013.)

2.2.1 Application plane

Applications on this plane are used by engineers to program and to maintain the network. Having in possession all network-related information, it provides the requirements to the lower layers. Moreover, the monitoring and security solutions are developed on this plane. (Open Networking Foundation 2012.)

According to Madhava (2018), the execution environment for applications can be internal and external as follows:

- The internal applications run inside the container of the controller. To be compatible with the OpenDaylight controller these applications should be written in Java and run within the Model-Driven Service Abstraction Layer (MD-SAL).
- The external applications can operate remotely and communicate with the controller using RESTful API. It gives the flexibility to use any programming language, for instance, Python, Bash or Java. However, this design is slower than the internal one.

Additionally, there are two types of application classes, reactive and proactive, as introduced by Brocade Communications Systems (2015):

- Reactive applications send the rules to the flow table once they receive a packet from a switch. Usually, for efficient callbacks and communication with the controller, such applications are developed to be internal.
- Proactive applications are triggered by external settings, such as link failures or data load changes, and not by received packets.

The internal reactive approach is well-suited for security applications because of the low latency and rapidity, whereas the proactive solution is commonly used to set up policy configurations and monitoring. Figure 3 compares the internal and external applications of the highest SDN layer.

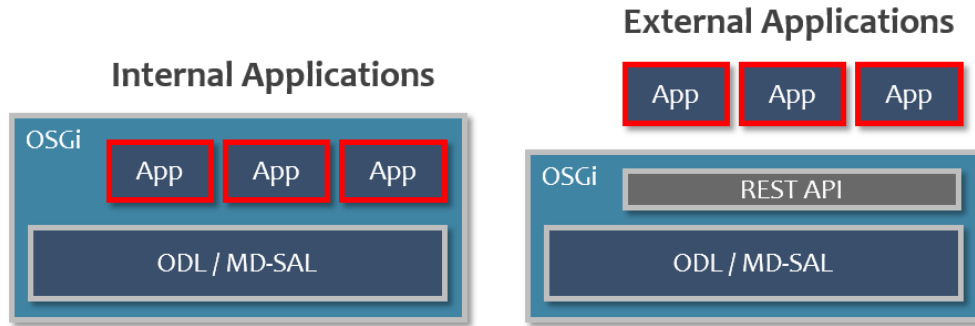


Figure 3. Internal and external application of SDN (Brocade Communications Systems 2015)

Various vendors build their own application plane solutions for different use cases. These applications can meet various business needs and they are actively used by network engineers. (Madhava 2018). For instance, on HPE SDN App Store, there are multiple applications available for network optimization, visualization and protection as well as load balancer application by different vendors (HP Inc. 2014).

2.2.2 Northbound interface

The area between application and control planes is named the Northbound interface (NBI). It is a programmatic control of abstracted network resources. Figure 4 shows the domains where the Northbound interface can be used.

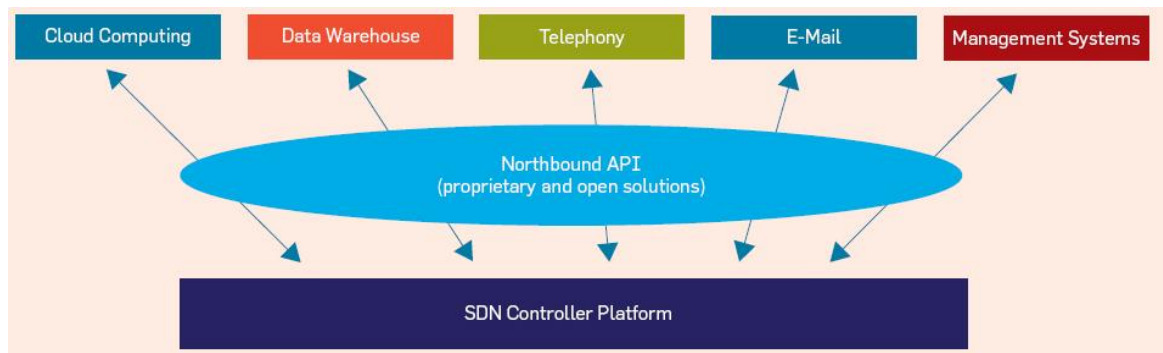


Figure 4. Northbound interface APIs

The NBI is a high-level application program interface (API), such as NETCONF or REST. It conceptualizes the infrastructure layer details and provides a method to

retrieve information from there. However, the development standard for the NBI does not exist yet. (Banse & Rangarajan 2015.)

A number of companies started the implementation of the NBI in their networking software. It is advantageous for big network enterprises to build their own NBI solutions, because it can be later integrated with their proprietary controller. For example, in the operating system Cisco IOS XE 16, NETCONF is provided to enable the integration with SDN. (Cisco 2018).

2.2.3 Control plane

The control plane is the center of SDN and is represented by an operation system and a controller. It is the abstraction layer that hides forwarding layer details from the application layer and translates the requirements from it to the forwarding layer. (Nadeau & Gray 2013.) Figure 5 presents the high-level overview of the SDN controller.

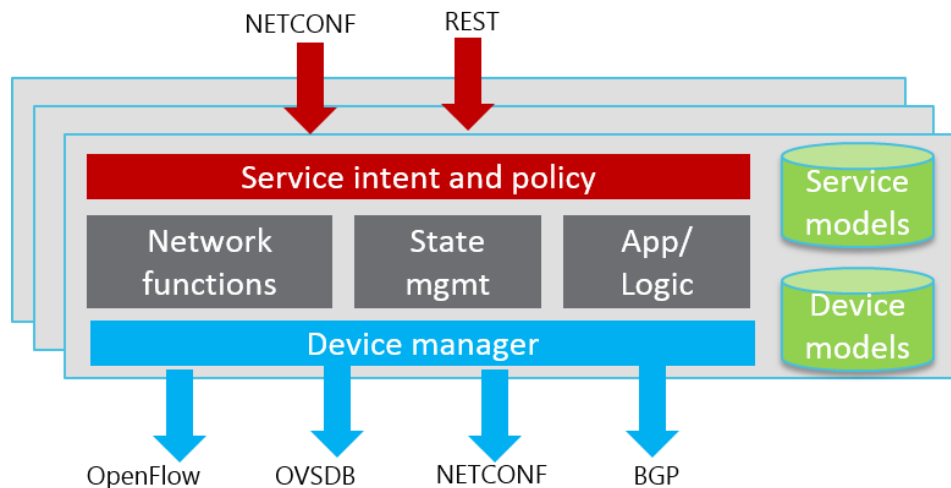


Figure 5. High-level SDN controller design

The operation system in SDN is the infrastructure consisting of the monitoring, access and resource management of the whole network, not only a single node. The role of the controller is to rule the data layer and to orchestrate a network. The controller consists of the interface for network applications, the network-wide state management layer and the communication to the data plane layer. (Hu et

al. 2017.) Moreover, it possesses the whole overview and topology of the network that helps to optimize packet flow. For instance, with this approach, the implementation of a mesh topology is possible on the layer 2 of the OSI model without IP routing involved. (Nadeau & Gray 2013.)

In the SDN world, Westbound and Eastbound API terms are commonly used. Westbound API is applied as a communication channel between the controllers of different network domains. The purpose of the Eastbound API is interacting with non-SDN planes, for instance, MPLS. (Zinner et al. 2013.)

The most common controller today is the OpenDaylight controller. It is a project under Linux foundation with the first release 'Hydrogen' launched in 2014. Since then, the development has never been stopped and the project keeps evolving with several releases each year. The latest version 'Sodium' was published in September 2019. The controller is open source and executed on Java Virtual Machine. (Knorr 2013.)

2.2.4 Southbound interface

The Southbound interface (SBI) is a logically centralized control of network resources. It is a low-level interface between network devices and a controller. There are several protocols of this interface for network device configuration, such as OpenFlow, NETCONF and SNMP. To programmatically change network devices from the application layer, other protocols should be used, for instance, OF-Config or NETCONF. (Azodolmolky 2013.)

OpenFlow is the most popular protocol used for the Southbound interface that is used in some implementations of SDNs between switches and the controller. OpenFlow does not change the configuration of network devices. Instead, it modifies the forwarding of traffic through the network devices. It is used to program traffic flows that go via routers and switches. To achieve this, three crucial components should be included in the OpenFlow switch: The flow table, the secure channel and the OpenFlow protocol. The secure channel is a

communication interface between the controller and the switch, usually it is a TLS asymmetrical encryption. (Kandoi 2015.)

A flow table is filled with rules sent by a controller. The traffic control in SDN is based on the flow and not on separate packets. A rule is specified for the first packet and then it is applied to the rest of the packets in the same flow (Lee & Chiueh 2014). The packets for each flow are defined based on port, MAC address and IP address. To illustrate this, Figure 6 shows the structure of an OpenFlow table row.

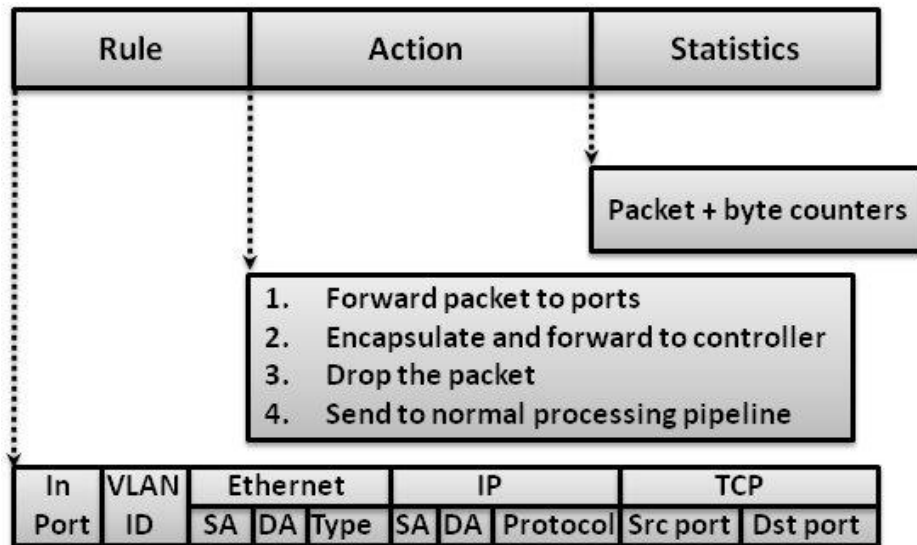


Figure 6. Flow table structure of an OpenFlow switch (Azodolmolky et al. 2011)

OpenFlow protocol has started as a research project in the University. Currently, it is maintained by the Open Networking Foundation. Nevertheless, they do not provide the standards for the Westbound and Eastbound interfaces nor for the Northbound interface. (Nadeau & Gray 2013.)

2.2.5 Data plane

The fundamental objective of the data plane is traffic forwarding and processing, including de-encapsulation, encryption, matching MAC addresses and changing source and destination IP addresses. Devices of this plane are routers, layer 2 and layer 3 switches. (Nadeau & Gray 2013.)

In comparison with Ethernet switches, which contain the table and have an integrated controller, an SDN switch only has data forwarding functionality and can be programmed with OpenFlow protocol by a controller. SDN switches are simpler and therefore faster. Figure 7 shows the structure of the OpenFlow switch. Each OpenFlow switch has a flow table that is filled in based on the information from the controller. If the switch does not have a matching flow rule for a packet, it buffers a packet and then sends a request OFPT_PACKET_IN message a rule for it from the controller. After, the response OFTP_FLOW_MOD message is sent back by a controller. OFTP_FLOW_MOD message consists of the action to perform and a timeout for which to keep the rule in the flow table. (Kandoi 2015.)

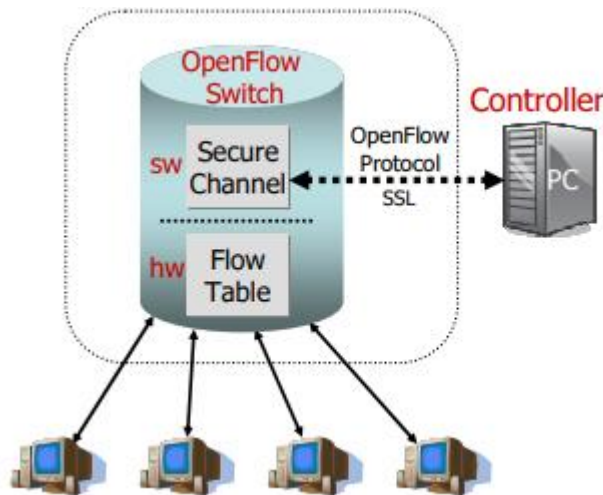


Figure 7. Main components of OpenFlow switch (McKeown et al. 2008, 69-74.)

Since vendors want to be up-to-date with SDN technologies, nowadays many commercial switches are OpenFlow-enabled. This means that on top of traditional switching infrastructure there is an SDN traffic steering available. To illustrate this, Cisco OpenFlow agent can be enabled on Cisco Nexus 3000 and 9000 series switches. The switches are configured by this agent with OpenFlow databases and interfaces (Cisco 2018.)

2.3 Use cases of SDN

Automation, agility and having an outline of a network topology are several advantages that SDN brings to network design. This section discusses how

enterprises can use this to improve or to plan their infrastructure. The most common use cases are discussed here, however, there are many other less popular options that are out of the scope of this paper.

Software-defined storage (SDS) is a technology, where local storage is located across multiple servers and the physical management is centralized. It can be compared with a traditional NAS network. However, it is not required to purchase actual NAS. As a result, with SDS it is possible to earmark storage within all physical computers, appearing as a solid logical volume and to hold it as a shared storage. Moreover, auto tiering is introduced with SDS. Providing flexibility, it automatically shifts the storage based on workload and needs of virtual machines. (Harvey 2018.)

Software-defined data centers (SDDC) are combinations of SDS, SDN and virtualization. More specifically, this structure involves a server, a CPU and memory, network and storage virtualization. It provides environment flexibility, especially for multi-tenant data centers. It is a great solution to improve scalability and automation in data centers. (Nadeau & Gray 2013.)

Software-defined security (SDSec) is common in infrastructures that are virtualized, or cloud-based. The capability of the controller to be aware of the whole network topology is a significant benefit in dynamically address security threats. Being in possession of such information, the powerful monitoring can be configured in the network. (Yoon et al. 2017).

The first use case, where SDN comes very handy, is a lack of network omniscience. A centralized controller gives a live overview of the network conditions allowing to troubleshoot easier and faster. The second use case is when managing many devices is time consuming. A standard interface for all devices via the controller is implemented using software. The long provisioning time is the third reason why SDN could be considered. Its virtualization and programmability improve the deployment and make DevOps achievable.

To summarize, according to Dargin (2018), SDN design is beneficial and highly advantageous in the following cases:

- When there is no necessity or recourses for buying hardware.
- When the resources are not utilized constantly.
- When the quality of service is essential for the business.

A different useful aspect of SDN is the ability of a controller to speak to different devices supporting the same protocol using standard interface. An SDN controller has the capability to orchestrate all the necessary changes as required. The vendor lock-in of the devices is not an unsolvable problem anymore. Consistency across devices is achieved with centralized controller automation, simultaneously updating the required configurations. Typos and human mistakes are less of a problem. Rolling back is straightforward, because the SDN controller knows the current state of the network. (Azodolmolky 2013.)

2.4 Security of SDN

To ensure the reliability and stability of SDN infrastructure, the network administrators need to implement common security practices. Particularly, the system and the OS should be kept updated and patched. The unused ports should be disabled and locked. For easy and reliable operations and troubleshooting, configuration management and log monitoring should be enforced. (Dargin 2018.)

The Confidentiality, Integrity, Availability (CIA) triad is a model that is applied to evaluate the security of any IT infrastructure and its appropriateness for the business requirements and standards. Figure 8 shows the CIA model of each plane of SDN, as suggested by Yoon et al. (2017).

<i>Location</i>	<i>Potential risk (CIA)</i>	<i>Critical assets</i>
Control Plane	Confidentiality	Controller configuration Network service configurations Global network-view
	Integrity	Controller configuration Network service configurations Active network services Global network-view Network behavior
	Availability	Controller instance Network services Global network-view
Control Channel	Confidentiality	Control messages
	Integrity	Control messages
	Availability	Control-data plane connections
Data Plane	Confidentiality	Network information Flow table (entries)
	Integrity	Flow table (entries)
	Availability	SDN-specific services

Figure 8. CIA model of SDN (Yoon et al. 2017)

This section covers the general attack surface of SDN. Separately, the security of the control plane is discussed. These topics are important to understand in order to answer the research questions of this work.

2.4.1 Attack surface

SDN is a new and developing architecture. Thus, the attention of security hackers focuses on its potential vulnerabilities. This statement is true for OpenFlow protocol as well. Moreover, as SDN is a highly virtualized infrastructure, the attack footprint is big. (Dargin 2018.)

The programmable application plane and the NBI are agile and flexible, and this brings several security challenges. Firstly, there are no compulsory security features enforced for these applications. Secondly, the NBI does not have a default standard that forces the development to be depended on a controller and its implementation. (Banse & Rangarajan 2015.)

If a business wants to implement SDN in its network, it is important to consider OpenFlow protocol vulnerabilities during the design stage. To give an example, OpenFlow handshake does not request authentication of switches. (Thimmaraju 2018.) This can be mitigated using other authentication methods, such as TLS certificates.

Data plane attacks are regularly related to wrong or illegitimate entries of the flow rules. For instance, the flow rules can overload the switch with bogus traffic. Other things to consider regarding the data plane attack surface is that a switch is a good target for eavesdropping; an intruder can explore the set up and configuration of the SDN network to plan an actual attack. (Yoon et al. 2017.)

2.4.2 Control plane security

The synchronization between a controller and the switches in a big SDN network takes a noticeable amount of time. Delays, packet loss and equipment from different vendors may cause asynchronization. This is a vulnerable moment for system security as it is not easy to predict its behaviour. This is the reason why monitoring traffic in the system is very important. (Hu et al. 2017.)

Multi-controller architecture is a way to resolve a single point of failure problem of SDN. However with such setup it is challenging to manage the reliability of the control plane. (Hu et al. 2017.) Hu et al. (2017) also suggest using controller clustering for load balancing of the controllers.

The control plane is the brain of the network. In order to avoid the stability issues of the controller, the appropriate security solution should be in place. It requires to cover confidentiality, integrity and availability of the controller. (Yoon et al. 2017.)

3 DENIAL OF SERVICE

Denial of Service is one of the most common attacks nowadays and has a share of 15% of all network attacks, after 20% for both browser and brute force attacks (McAfee Labs 2017). In a Denial of service attack, the hacker tries to prevent normal operations of a website, a server or a user. By overloading the system with traffic, it consumes all the available resources and bandwidth.

According to Bulletproof Annual Cyber Security Report 2019 (2019), the cost of denial of service attacks can go up to 120,000 USD for a small business and

more than 2 million USD for an enterprise. Thus, these attacks are very expensive for companies, especially for startups and other small business.

In this chapter, the history of denial-of-service attacks is described. Next, the available methods to execute such attacks are explained. Lastly, the impact of DoS attack and the solutions to mitigate them in SDN are research before the practical part.

3.1 History

In 1974, the first DoS attack was initialized by accident on a PLATO terminal, a shared electronic learning system. David Dennis, a 13-year-old student from the United States, accidentally found the “external” command that was initially used for communication with external hosts. A Denial of Service was caused by the student running the command on a computer with no external devices. This action made the terminal freeze with a shutdown being the only way to fix it. (Radware security 2017.)

The first distributed DoS attack succeeded in 1999 using the Trinoo tool. The UDP traffic from the infected hosts flooded servers of University of Minnesota in 1999. The attacker used the vulnerability of buffer overrun in the RPC protocol services. (Kessler 2000.) The Trinoo was used to make zombies out of Unix hosts. Later, The Wintrinno tool made it possible to infect Windows systems in 2000.

One more way to create a DoS attack is to flood a target host with ICMP packages. This attack is called ping of death. This is quite an old-fashioned attack and usually does not have big impact nowadays. Operating systems usually have ICMP requests blocked on the firewall level. (Ballal 2018.)

Since 1999 there has been an enormous amount of DDoS attacks differentiated by approach and impact. For instance, in 2012, several big US banks became the targets of a DDoS attack. The attack was executed by different methods in order

to monitor which one worked better. The peak load from this attack was over 60 Gbps. (Constantin 2012.)

Once technology has been evolving and more resources and companies have moved their businesses to the internet world, the potential impact of a DoS-like attack increased significantly. Moreover, nowadays such attacks are not performed for fun or curiosity, but for other motivation, such as gaining profits or political manipulation. Not to mention that the tools to perform such attacks exist and are available for purchase on the Darknet. Before, advanced knowledge and skills were the essential components to implement cyber security attacks. Nowadays, denial-of-service is available as-a-service. (Radware security 2017.)

3.2 Techniques

To efficiently mitigate an attack, it is important to correctly categorize it. Denial-of-service attacks can be grouped by three types: volume, protocol and application layer. Whereas they have a common goal, each of the three types targets different levels of the network. (Ballal 2018.)

Volume based attacks are processed by overloading the bandwidth of the target. Bits per second is the measure of its size. According to Ballal (2018), a few of the most well-known attacks of this type are the following:

- UDP flood is triggered when the attacker sends UDP packets to random ports of the attacked host. After a non-listening application is found on the port, the host replies with ICMP package “destination unreachable” that causes the resource starvation of the victim. This attack can be performed using ready-to-use software, for instance, UDP Unicorn.
- ICMP flood works the same way as UDP flood, but instead of UDP, the attacker overwhelms the target with ICMP Echo request without waiting for the reply. Such an attack not only consumes incoming bandwidth but outgoing as well, as the victim host tries to respond with ICMP Echo reply messages.

Protocol attacks are focused on the resources of servers or on intermediate devices, for instance, load balancers and firewalls. The strength of this type is measured in packets per second. According to Cloudflare (2019), the protocol attacks are as follows:

- TCP SYN flood attacks exploit an implementation of TCP protocol, particularly the design of the three-way handshake. An attacker initiates a TCP connection with a victim with SYN package. Then the victim responds with a SYN-ACK packet to the attacker or a spoofed IP address and waits for the response, the final ACK packet. If there is a sufficient amount of half-open connections that occupy entire resources, the victim will not be able to accept any further TCP connection requests.
- A Smurf DDoS attack is another attack that makes use of ICMP protocol. An attacker, using the spoofed source IP address of the victim, broadcasts ICMP Echo request within a network. Then the victim will receive all the ICMP Echo reply traffic, which may depreciate the whole victim's network.

Application layer attacks are usually based on webserver requests. They are measured in requests per second. Ballal (2018) describes the following application layer attacks:

- HTTP flood is an attack in which an attacker targets a victim web server with HTTP request.
- Slowloris attack is processed by holding many open connections to a victim web server for long time. It is implemented by transmitting partial requests, that overload the victim and results in denying other connections.

The separate type of DoS attacks is a zero-day attack. It exploits the vulnerabilities in a system that are not yet known by the software vendor but found by an attacker. To minimize the chance of this attack, the best security practices should be implemented in the environment. (Bennet 2016.)

3.3 DoS of SDN

Although the controller is usually more attractive as a DoS attack target, SDN switches are vulnerable too. For example, an intruder can overload the Ternary Content Addressable Memory (TCAM) table of an OpenFlow switch. For this overflow attack, many new flows are created and forwarded to the switch. Therefore, it is occupied with removing and adding the entries to the table and flooding the messages to the controller. (Balogh 2017.)

There are a few attack vectors in SDN that an attacker can use to successfully implement a DoS attack on an SDN controller. There are several potential

vulnerabilities described by Yoon et al. (2017) that can be used to complete the attack against the controller as follows:

- Architectural bottleneck: Hosts in the same network can generate an extensive number of packets to flood the control plane.
- Weak authentication: Switches in the network can be spoofed and be exploited to drop the connections or to generate not legitimate traffic.
- Improper exception handling: An intruder can inject malformed messages to the control plane to perform the attack.
- Dependence on external variable: Manipulation with system time can lead to unstable behaviour of the controller.

Using the above vulnerabilities, for instance, an attacker can target a data plane device and inject many false flow request that are not yet existing in flow table. The controller receives all these requests and needs to process them. If there are sufficient requests sent, the controller's CPU usage might go up to 100%. (Dridi & Zhani 2016.) A different approach is to send fake flow requests to the controller that will push wrong network flows back to the switches.

3.3.1 Detection and mitigation solutions

The fundamental attack detection solution in SDN is to detect dubious traffic using complete overview of the network. The main mitigation technique is to configure flow tables correctly. For instance, they should include event filtering, rule timeout adjustments and packet dropping. Access Control List (ACL) should be configured as flow rules as well. (Ballal 2018.)

The SDN infrastructure should adopt specialized monitoring systems for deeper packet inspection (DPI). DPI is an advanced traffic filtering technology that is commonly implemented in firewalls to mitigate attacks, particularly DoS. (Cisco 2019). According to Finnie (2012), DPI is the most efficient when collocated with the devices on the data plane layer or the controller. A challenging aspect of this method is to configure DPI in real time. As the network resources should be properly allocated, Quality of service (QoS) should be implemented as well.

Machine learning (ML) provides intelligence into traffic analysis within SDN. It has capabilities of dynamically updating flow tables based on machine learning

algorithms. Machine learning models reside in the application plane of SDN and communicate with the controller via the NBI. A deviation from what is known by the model triggers an alert. Before the model is properly trained, it gives many false positive alerts. However, with the right dataset this approach can achieve significant results in DoS attack detection. (Nguyen 2018).

4 IMPLEMENTATION

In this chapter, an SDN environment is built in the GNS3 simulation environment. Then, a DoS attack is created against the controller and its behaviour is observed under the attack. Before the actual documentation of the built system, the technologies that are used to conduct the practical part of this thesis are described.

4.1 Simulation environment setup

This thesis project is implemented on a single physical machine. Thus, the simulation and hypervisor software is required in order to build a working test setup. GNS3 and VMware were chosen as such tools.

Graphical Network Simulator 3 (GNS3) is an open source and free-to-use software under GNU General Public License for virtual and real networks simulation. GNS3 has an expanding community of 800,000 users, who support and develop the application. (Bombal 2018). To start working in the GNS3 environment, two components are required to be installed: GNS3 virtual machine and GNS3-all-in-one GUI application. GNS3 VM acts as a server to run the nodes and appliances created in the GUI. The installation link of GNS3 is available after registration on the official website (<https://www.gns3.com/>). For this project GNS3 2.2.0 version is installed. During the setup, it is possible to install different components, such as console clients, network packet sniffing and analyzer tools.

VMware Workstation is hypervisor software that can be installed on Windows and Linux operating systems. It is free to use for non-commercial personal use.

(VMWare 2019). VMware Workstation is chosen as a tool for creating virtual machine's setup for this project. GNS3_VM.ova file can be downloaded from the GNS3 website. This Open Virtualization Format (OVA) file is used to describe the GNS3 VM image and is loaded into VMware Workstation. The imported GNS3 VM runs on the Ubuntu 64-bit operating system and its configurations are shown in Figure 9.









Device	Summary
 Memory	4 GB
 Processors	2
 Hard Disk (SCSI)	19.5 GB
 Hard Disk 2 (SCSI)	488.3 GB
 CD/DVD (IDE)	Auto detect
 Network Adapter	Host-only
 Network Adapter 2	NAT
 Display	Auto detect

Figure 9. GNS3 VM 2.2.0 settings

The combination of these tools provides flexibility. This setup allows to experiment with the network configurations. Moreover, both technologies are free for personal use.

4.2 Simulation of software-defined network

The chosen approach is to have the topology consisting of two Open vSwitch devices and one controller. The controller is the OpenDaylight controller running in a Docker container. Four Ubuntu hosts are added to the network to complete a DoS attack.

First, the Ubuntu appliance is downloaded from the GNS3 website. The file ubuntu.gns3a is imported as a Docker container to the GNS3 project. Later, this container is set up to host the OpenDaylight controller. Additionally, a layer 3 Ethernet switch and a NAT cloud are added to provide internet access for the topology. The next step is to create two Open vSwitch switches, the management interface of these switches is eth0. To each of Open vSwitch switches, two Ubuntu hosts are connected. Figure 10 presents the SDN topology created.

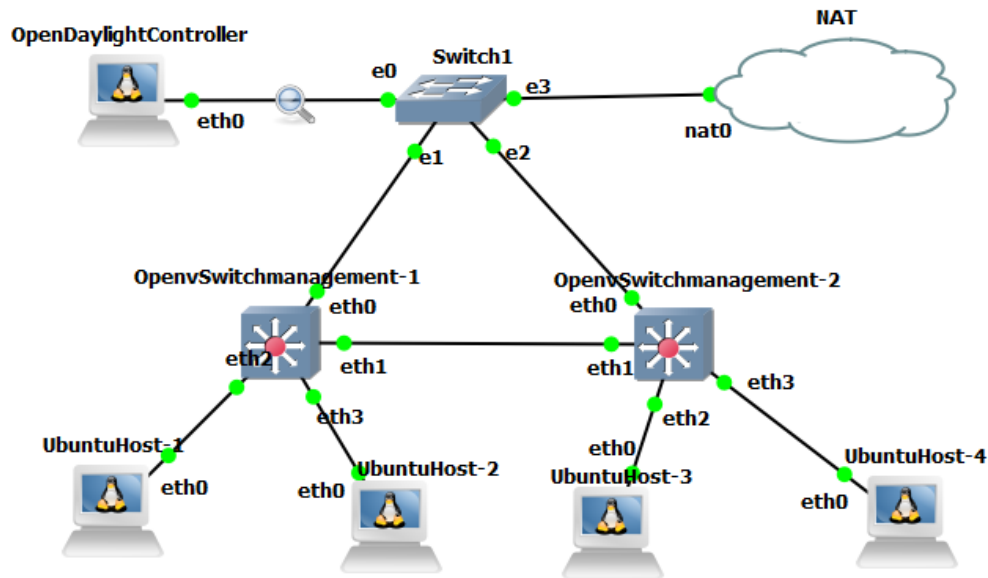


Figure 10. SDN topology in GNS3

At this stage, the OpenDaylight controller and OF switches are in the same network and can ping each other. The next steps are to install OpenDaylight controller and to configure devices to communicate with each other using OpenFlow protocol.

To update, install or delete the packages in Linux from the command line, `apt-get` command is used. (Siever et al. 2009.) The commands in Code 1 update the installed packages, install Java, export `JAVA_HOME` and install `wget` to download files from the internet.

```

root@OpenDaylightController:~# apt-get update
root@OpenDaylightController:~# apt-get install default-jre-headless
root@OpenDaylightController:~# export
JAVA_HOME=/usr/lib/jvm/default-java
root@OpenDaylightController:~# apt-get install wget
root@OpenDaylightController:~# apt-get install unzip

```

Code 1. Updating and installing packages to the ODL controller

The next step is to download the Karaf distribution of OpenDaylight from Nexus Repository manager where all versions of ODL are stored and to install the ODL.

The compressed file with Karaf 0.6.4 Carbon is downloaded using `wget` and unzipped with `unzip` (Code 2).

```
root@OpenDaylightController:~# wget
https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/integration/distribution-karaf/0.6.4-Carbon/distribution-karaf-0.6.4-Carbon.zip
root@OpenDaylightController:~# unzip distribution-karaf-0.6.4-Carbon.zip
```

Code 2. Downloading and unzipping the file

Figure 11 illustrates what the installation folder of the ODL distribution artifact consists of.

```
root@OpenDaylightController:~/distribution-karaf-0.6.4-Carbon# ll
total 64
drwxr-xr-x  9 root root  4096 Apr 21  2018 ./
drwx----- 4 root root  4096 Dec  8 12:10 ../
-rw-r--r--  1 root root  1126 Apr 21  2018 CONTRIBUTING.markdown
-rw-r--r--  1 root root 11266 Apr 21  2018 LICENSE
-rw-r--r--  1 root root   172 Apr 21  2018 README.markdown
drwxr-xr-x  2 root root  4096 Apr 21  2018 bin/
-rw-r--r--  1 root root    76 Apr 21  2018 build.url
drwxr-xr-x  2 root root  4096 Apr 21  2018 configuration/
drwxr-xr-x  3 root root  4096 Apr 21  2018 data/
drwxr-xr-x  2 root root  4096 Apr 21  2018 deploy/
drwxr-xr-x  2 root root  4096 Apr 21  2018 etc/
drwxr-xr-x  5 root root  4096 Apr 21  2018 lib/
drwxr-xr-x 26 root root  4096 Apr 21  2018 system/
-rw-r--r--  1 root root  2717 Apr 21  2018 taglist.log
```

Figure 11. Content of distribution-karaf-0.6.4-Carbon folder

The command `./bin/karaf` starts Apache Karaf, that is OSGi runtime environment, that hosts OpenDaylight controller. Figure 12 shows the ODL console launched.

The command configures the ODL controller on the switches (Code 5). The management interface of the switches is by default `eth0`, so all other connections are made to `br0` interface. The port for OpenFlow TCP traffic is 6633.

```
ovs-vsctl set-controller br0 tcp:192.168.122.203:6633
```

Code 5. Setting the controller

Figure 14 shows an output of the `ovs-vsctl` command that confirms that the controller is successfully connected to the OF switch.

```
Bridge "br0"
  Controller "tcp:192.168.122.203:6633"
    is_connected: true
  Port "eth5"
    Interface "eth5"
  Port "br0"
```

Figure 134. br0 interface with controller connected

Now, in Figure 15, `dump-flows` command shows the flow entries with the controller specified.

```
/ # ovs-ofctl -O OpenFlow13 dump-flows br0
OFPST_FLOW reply (OF1.3) (xid=0x2):
  cookie=0x2b00000000000001, duration=299.353s, table=0, n_packets=60, n_bytes=6660, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
  cookie=0x2b00000000000001e, duration=295.296s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=10 actions=output:11,output:12,output:13,output:14,output:15,output:3,output:2,output:1,output:7,output:6,output:5,output:4,output:9,output:8,CONTROLLER:65535
  cookie=0x2b00000000000001f, duration=295.296s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=11 actions=output:10,output:12,output:13,output:14,output:15,output:3,output:2,output:1,output:7,output:6,output:5,output:4,output:9,output:8,CONTROLLER:65535
  cookie=0x2b000000000000020, duration=295.296s, table=0, n_packets=0, n_bytes=0, priority=2,in_port=12 actions=output:10,output:11,output:13,output:14,output:15,output:3,output:2,output:1,output:7,output:6,output:5,output:4,output:9,output:8,CONTROLLER:65535
```

Figure 145. dump-flows command

Finally, when the SDN topology is setup and OpenFlow is enabled, packet capturing is enabled on `eth0` interface of the OpenDaylight controller.

Preinstalled with GNS3, Wireshark is a free and open-source software for packet capturing and network analysis. (Petters 2019.)

4.3 Simulation of DoS traffic

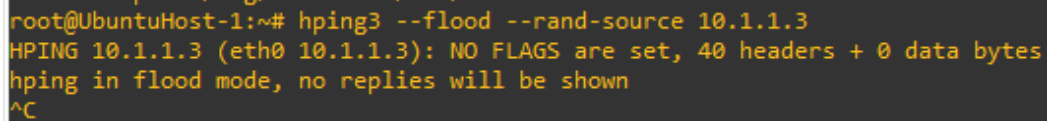
First, IPv4 addresses are assigned to the Ubuntu hosts connected to the OF switches, as illustrated in Figure 10. The hosts are in the same 10.1.1.0

255.255.255.0 network. With the command `ifconfig eth0 10.1.1.4 netmask 255.255.255.0 up`, UbuntuHost-1 obtains 10.1.1.1.

Correspondingly with the similar command, UbuntuHost-2 gets 10.1.1.2, UbuntuHost-3 – 10.1.1.3 and UbuntuHost-4 – 10.1.1.4.

The approach of the attack is to create ping packets from UbuntuHost-1 to UbuntuHost-3 and from UbuntuHost-4 to UbuntuHost-2. The source IP addresses of these ICMP packets are generated randomly. The `hping3` tool is used to create a DoS attack on the controller. This tool can send custom ICMP and TCP/IP packets. (Kacherginsky 2008.)

The `hping3` is executed on UbuntuHost-1 and UbuntuHost-4. This command is represented in Figure 16. `--flood` option rapidly sends the packets and does not show incoming replies, `--rand-source` is a random source option, changes the source IP of each request and, in the end, 10.1.1.3 is the IP address of the destination, of UbuntuHost-3. The same command is run on UbuntuHost-4, with the destination address of UbuntuHost-2, 10.1.1.2.



```

root@UbuntuHost-1:~# hping3 --flood --rand-source 10.1.1.3
HPING 10.1.1.3 (eth0 10.1.1.3): NO FLAGS are set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C

```

Figure 156. `hping3` command

4.4 Findings and results

There are several observations about the controller's behaviour during the ping flood attack. These can be used as an indication of malicious traffic in order to detect a DoS attack in time and to mitigate it. For instance, such network actions are good entries to be logged into monitoring systems and to trigger alerts.

Firstly, the ODL controller is not able to reply to each received `OFPT_PACKET_IN` message with an `OFPT_PACKET_OUT` message, as illustrated in Figure 17.

During the normal load, if a switch receives a packet for which it does not have a flow rule yet, it buffers the data and sends only the header in `OFPT_PACKET_IN`

message to the controller. In case the buffer of the switch is already full, `OFPT_PACKET_IN` message will consist of both the header and the data. Accordingly, the controller will also send the full packet in `OFPT_PACKET_OUT` message back as a reply. Transmitting complete packets in these messages massively consumes the SBI bandwidth. (Kandoi 2015.)

No.	Time	Source	Destination	Protocol	Length	Identification	Info
99975	328.141765	192.168.122.250	192.168.122.203	OpenFlow	168	0x38bf (14527),0x70f8 (28920)	Type: OFPT_PACKET_IN
99976	328.144561	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c0 (14528),0xaab1 (43697)	Type: OFPT_PACKET_IN
99977	328.144665	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c1 (14529),0x0231 (561)	Type: OFPT_PACKET_IN
99978	328.144695	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c2 (14530),0xc9a6 (51622)	Type: OFPT_PACKET_IN
99980	328.156336	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c3 (14531),0xc644e (25678)	Type: OFPT_PACKET_IN
99981	328.162484	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c4 (14532),0x9b1c (39708)	Type: OFPT_PACKET_IN
99982	328.168753	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c5 (14533),0x7aed (31469)	Type: OFPT_PACKET_IN
99983	328.168835	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c6 (14534),0xf6b5 (64357)	Type: OFPT_PACKET_IN
99984	328.168866	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c7 (14535),0xf72e (63278)	Type: OFPT_PACKET_IN
99985	328.168892	192.168.122.203	192.168.122.215	OpenFlow	1514	0x0d04 (3332)	Type: OFPT_PACKET_OUT
99986	328.168921	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c8 (14536),0x58fc (22780)	Type: OFPT_PACKET_IN
99987	328.168946	192.168.122.203	192.168.122.215	OpenFlow	585	0x0d05 (3333)	Type: OFPT_PACKET_OUT
99988	328.168967	192.168.122.250	192.168.122.203	OpenFlow	168	0x38c9 (14537),0xfaba (64186)	Type: OFPT_PACKET_IN
99989	328.169035	192.168.122.203	192.168.122.215	OpenFlow	532	0x0d06 (3334)	Type: OFPT_PACKET_OUT
99990	328.169119	192.168.122.250	192.168.122.203	OpenFlow	168	0x38ca (14538),0x7271 (29297)	Type: OFPT_PACKET_IN

Figure 17. `OFPT_PACKET_IN` and `OFPT_PACKET_OUT` messages during the DoS attack

Secondly, the controller was periodically sending `OFPT_BARRIER_REQUEST` messages back to the switches. This example is shown in Figure 18. This message indicates that the controller requests an OF switch to process all the messages received before the `OFPT_BARRIER_REQUEST` message. When the switch processes the requested operations, it sends an `OFPT_BARRIER_REPLY` message back. At the same time, this guarantees message ordering.

No.	Time	Source	Destination	Protocol	Length	Identification	Info
104242	339.149997	192.168.122.250	192.168.122.203	OpenFlow	168	0x4512 (17682),0x001f (31)	Type: OFPT_PACKET_IN
104243	339.150916	192.168.122.203	192.168.122.215	OpenFlow	74	0x0d3b (3387)	Type: OFPT_BARRIER_REQUEST
104244	339.151560	192.168.122.215	192.168.122.203	OpenFlow	74	0xa915 (43285)	Type: OFPT_BARRIER_REPLY
104245	339.154376	192.168.122.250	192.168.122.203	OpenFlow	168	0x4513 (17683),0x6c4f (27727)	Type: OFPT_PACKET_IN
104246	339.158268	192.168.122.250	192.168.122.203	OpenFlow	168	0x4514 (17684),0xd193 (53651)	Type: OFPT_PACKET_IN
104248	339.161988	192.168.122.250	192.168.122.203	OpenFlow	168	0x4515 (17685),0x0db1 (3505)	Type: OFPT_PACKET_IN

Figure 168. `OFPT_BARRIER_REQUEST` and `OFPT_BARRIER_REPLY` messages

Also, TCP Out-Of-Order and TCP Retransmission issues were observed by Wireshark, as shown in Figure 19. The first error indicates that a transmitted segment is received in a wrong order. The TCP Retransmission problem can be a result of a buffer overflow.

No.	Time	Source	Destination	Protocol	Length	Identification	Info
148146	409.028567	192.168.122.203	192.168.122.215	TCP	1067	0x0dec2 (3778)	[TCP Retransmission] 6633 → 5
148147	409.028609	192.168.122.250	192.168.122.203	OpenFlow	168	0xc055 (49237),0x2f85 (12165)	Type: OFPT_PACKET_IN
148148	409.028652	192.168.122.203	192.168.122.250	TCP	74	0xdad1 (56017)	[TCP Out-Of-Order] 6633 → 344
148149	409.028685	192.168.122.215	192.168.122.203	TCP	78	0xaa50 (43600)	52754 → 6633 [ACK] Seq=119731
148150	409.028764	192.168.122.250	192.168.122.203	OpenFlow	168	0xc056 (49238),0xe54b (58699)	Type: OFPT_PACKET_IN
148151	409.028808	192.168.122.203	192.168.122.215	OpenFlow	1514	0x0ec3 (3779)	[TCP Spurious Retransmission]
148152	409.028842	192.168.122.250	192.168.122.203	OpenFlow	168	0xc057 (49239),0xd99 (28057)	Type: OFPT_PACKET_IN
148153	409.050309	192.168.122.203	192.168.122.250	TCP	74	0xdad2 (56018)	[TCP Retransmission] 6633 → 3
148154	409.050369	192.168.122.215	192.168.122.203	TCP	78	0xaa51 (43601)	[TCP Dup ACK 148149#1] 52754
148155	409.050415	192.168.122.250	192.168.122.203	OpenFlow	168	0xc058 (49240),0x0b98 (2968)	Type: OFPT_PACKET_IN
148156	409.050470	192.168.122.203	192.168.122.215	OpenFlow	1514	0x0ec4 (3780)	[TCP Spurious Retransmission]
148157	409.050522	192.168.122.250	192.168.122.203	OpenFlow	168	0xc059 (49241),0x556b (21867)	Type: OFPT_PACKET_IN
148158	409.050581	192.168.122.203	192.168.122.215	TCP	66	0x0ec5 (3781)	6633 → 52754 [ACK] Seq=194122
148159	409.050601	192.168.122.250	192.168.122.203	OpenFlow	168	0xc05a (49242),0x94f8 (38136)	Type: OFPT_PACKET_IN
148160	409.050655	192.168.122.203	192.168.122.215	TCP	78	0x0ec6 (3782)	[TCP Dup ACK 148158#1] 6633 →
148161	409.050675	192.168.122.250	192.168.122.203	OpenFlow	168	0xc05b (49243),0xe5b5 (58805)	Type: OFPT_PACKET_IN
148162	409.050740	192.168.122.203	192.168.122.215	TCP	78	0x0ec7 (3783)	[TCP Dup ACK 148158#2] 6633 →

Figure 19. TCP error packets

One of the ways to prevent bandwidth starvation is to set a rate limit on the controller. In OpenFlow 1.3 version, it is possible for the controller to install a rate limiting rule for one of the OF switches, which sends more packets than the controller can handle. The disadvantage of such a solution is that it slows down legitimate traffic together with malicious packets.

5 THE FUTURE OF SDN

SDN made its way from the virtualization technology of data centers to the architecture that is seen as a flexible and programable alternative to traditional networks. According to a Market Research Future (2019) prediction, the market revenue of SDN will increase over three times in a 5-year period, from 2018 to and 2023, rising to 28.9 billion USD. One of the most important aspects to consider, when forecasting the future of a technology, is the implementation and usage costs for the consumer. SDN is highly beneficial economically as it combines virtualization and agility, which brings hardware components and IT management expenses down. (Nadeau & Gray 2013.)

Today, the biggest challenge is to find a universal way to recognise a DoS attack. In academic research, machine learning techniques are already accessible. However, to adopt these practices for commercial products, ML models should generate proper logs and follow a security development process. Artificial intelligence and machine learning have great potential to improve packet analysis and cyber-attack prevention within SDN. (Nguyen 2018.)

To sum up, SDN will grow along with cloud computing and virtualization. The security aspect of it is advancing as well. There are not many experiences with SDN-targeted attacks yet. Since traditional networks are still vulnerable for new and unpredictable attacks, the potential target point of SDN should be only anticipated. Even if currently an SDN has security flaws for some use cases, more security solutions and research work will be done in this field by industry and in academia in the future. (Banse & Rangarajan 2015.)

6 CONCLUSIONS

The purpose of this work was to research the area of software-defined networks. It was important to evaluate if this new approach to network design brings additional security concerns and threats. Particularly, the goal was to understand if the centralized management could become a critical vulnerability to denial-of-service attacks.

The background research covered the initial separation and evolution of SDN from traditional networks. Furthermore, the main architecture components and use cases of SDN were described. Then, the overall attack surface and the control plane security were evaluated. Before moving to practical part, it was important to explain the history and technologies behind denial-of-service attacks.

In the practical part, an SDN network topology simulation was created using GNS3 software. In this simulation, the OpenDaylight controller communicated to switches using OpenFlow protocol. The ping flood DoS attack against the controller was simulated. During the attack, Wireshark software was capturing the traffic and based on this the controller's behaviour was evaluated.

This thesis described a general high-level overview of the comprehensive field of software-defined networks. The security aspects were covered more in depth and were elaborated on. Nevertheless, there are topics that can be researched and developed more in scope of future projects, for instance, ML models built specifically for SDN or application layer security software.

Nowadays, more and more companies move their businesses to cloud and adopt DevOps methodologies to keep up with the pace of information technologies. SDN is a rapidly evolving technology and with the added flexibility and virtualization, it gains more popularity in enterprises. Another thing to bear in mind is that security risks and threats should always be carefully evaluated and considered on the design stage of any network to avoid complications in the future.

A centralized management of SDN can be properly secured against DoS attacks using various methods, such as a multi-controller setup or appropriate monitoring techniques. Moreover, security should be considered and be properly planned from the initial stages of network design. Today, successfully maintaining a stable SDN environment is challenging but it is possible with the right knowledge and technologies.

REFERENCES

Amazon Web Services. 2019. About AWS. WWW document. Available at: <https://aws.amazon.com/about-aws/> [Accessed 8 April 2019].

Azodolmolky, S. 2013. Software Defined Networking with OpenFlow. Birmingham: Packt Publishing Ltd.

Azodolmolky, S., Nejabati, R., Escalona, E., Jayakumar, R., Efstathiou, N. & Simeonidou, D. 2011. Integrated OpenFlow-GMPLS control plane: an overlay model for software defined packet over optical networks. Optical Society of America.

Ballal, S. K. 2018. Bumper to Bumper: Detecting and Mitigating DoS and DDoS Attacks on the Cloud, Part 2. WWW document. Updated 16 May 2018. Available at: <https://securityintelligence.com/bumper-to-bumper-detecting-and-mitigating-dos-and-ddos-attacks-on-the-cloud-part-2/> [Accessed 25 November 2019].

Balogh, A. 2017. Addressing TCAM limitations in an SDN-based Pub/Sub System.

Banase, C. & Rangarajan, S. 2015. A Secure Northbound Interface for SDN Applications.

Bartels, A. 2011. Data Center Evolution: 1960 to 2000. WWW document. Updated 31 august 2011. Available at: <https://blog.rackspace.com/datacenter-evolution-1960-to-2000> [Accessed 1 April 2019].

Bennet, M. 2016. Zero-Day DDoS Attacks? What? WWW document. Updated 1 November 2016. Available at: <https://medium.com/@bennettaur/zero-day-ddos-attacks-what-93f57e0e2d6> [Accessed 9 December 2019].

Bohm, M., Leimeister, S., Riedl, C. & Krcmar, H. 2011. Cloud Computing and Computing Evolution.

Bombal, D. 2018. Why Use the GNS3 Virtual Network Simulator? WWW document. Updated 4 October 2018. Available at: <https://business.udemy.com/blog/why-use-the-gns3-virtual-network-simulator/> [Accessed 25 November 2019].

Bort, J. 2014. The Inside Story Of A \$1 Billion Acquisition That Caused Cisco To Divorce Its Closest Partner, EMC. WWW document. Updated 26 October 2019. Available at: <https://www.businessinsider.com/inside-the-1-billion-vmware-nicira-buy-2014-10?international=true&r=US&IR=T> [Accessed 9 April 2019].

Brocade Communications Systems, Inc. 2015. SDN Applications. WWW document. Updated 29 September 2015. Available at: <https://github.com/BRCDComm/BVC/wiki/SDN-applications> [Accessed 25 November 2019].

Burt, J. 2012. How Cisco, HP, Juniper, Others Are Tackling SDN, OpenFlow. WWW document. Updated 27 April 2012. Available at: <https://www.eweek.com/networking/cisco-systems> [Accessed 9 April 2019].

Chao, L. 2016. Cloud computing networking: theory, practice, and development. Boca Raton: CRC Press.

Cisco. 2018. A Cisco Guide to Defending Against Distributed Denial of Service Attacks. WWW document. Available at: https://tools.cisco.com/security/center/resources/guide_ddos_defense [Accessed 28 November 2019].

Cisco. 2018. Cisco IOS XE 16: Secure, Open, and Flexible. PDF document. Available at: <https://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/ios-xe/nb-09-ios-xe-secure-open-flex-aag-cte-en.pdf> [Accessed 3 December 2019].

Cloudflare. 2019. What is a Denial-of-Service (DoS) Attack? Available at: <https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/> [Accessed 3 December 2019].

Constantin, L. 2012. DDoS attacks against U.S. banks peaked at 60 Gbps. WWW document. Updated 13 December 2012. Available at: <https://www.computerworld.com/article/2493861/ddos-attacks-against-u-s-banks-peaked-at-60-gbps.html> [Accessed 25 November 2019].

Dargin, M. 2018. Secure your SDN controller. WWW document. Updated 2 January 2018. Available at: <https://www.networkworld.com/article/3245173/secure-your-sdn-controller.html> [Accessed 1 December 2019].

Dargue, M. 2015. Software-Defined Networking: Disruption and Opportunity. WWW document. Updated 5 November 2015. Available at: <https://blog.cartesian.com/software-defined-networking-disruption-and-opportunity> [Accessed 8 December 2019].

Dridi, L. & Zhani, M. F. 2016. SDN-guard: DoS attacks mitigation in SDN networks. *5th IEEE International Conference on Cloud Networking*, 213-217.

Finnie, G. 2012. The Role of DPI in an SDN World. Heavy Reading.

Guo, C., Lv, G., Yang, S. & Wang J. H. 2014. Virtual Data Center Allocation with Bandwidth Guarantees. U.S. Pat. 8,667,171.

Harvey, C. 2018. What is Software Defined Storage? WWW document. Updated 22 February 2018. Available at: <https://www.enterprisestorageforum.com/storage-software/what-is-software-defined-storage.html> [Accessed 26 November 2019].

HP Inc. 2014. HP Launches Industry's First SDN App Store, Unleashing New Wave of Networking Innovations. WWW document. Updated 25 September 2014. Available at: <https://www8.hp.com/us/en/hp-news/press-release.html?id=1798074> [Accessed 26 November 2019].

Hu, T., Guo, Z., Baker, T. & Lan, J. 2017. Multi-controller Based Software-Defined Networking: A Survey.

Kacherginsky, P. 2008. Hping Tips and Tricks. WWW document. Updated 13 August 2008. Available at: <https://medium.com/@iphelix/hping-tips-and-tricks-85698751179f> [Accessed 2 December 2019].

Kandoi, R. 2015. Denial-of-Service Attacks in OpenFlow SDN Networks.

Kaspersky Lab US. 2012. Survey: Among U.S. Companies Adopting IT Virtualization, 53 Percent Use Sub-Par Protection to Secure Virtual Infrastructure. WWW document. Updated 27 April 2012. Available at: https://usa.kaspersky.com/about/press-releases/2012_survey-among-u-s-companies-adopting-it-virtualization-53-percent-use-sub-par-protection--to-secure-virtual-infrastructure [Accessed 8 April 2019].

Kessler, G. C. 2000. Defences Against Distributed Denial of Service Attacks. WWW document. Available at: <https://www.garykessler.net/library/ddos.html> [Accesses 28 November 2019].

Kirkpatrick, M. 2010. Google CEO Schmidt: "People Aren't Ready for the Technology Revolution". WWW document. Updated 4 August 2010. Available at: https://readwrite.com/2010/08/04/google_ceo_schmidt_people_arent_ready_for_the_tech/ [Accessed 28 March 2019].

Knorr, E. 2013. OpenDaylight: A big step toward the software-defined data center. WWW document. Updated 8 April 2013. Available at: <https://www.infoworld.com/article/2614152/opendaylight--a-big-step-toward-the-software-defined-data-center.html> [Accessed 9 April 2019].

Market Research Future. 2019. Software Defined Networking (SDN) Market Research Report- Global Forecast 2023. WWW document. Updated February 2019. Available at: <https://www.marketresearchfuture.com/reports/software-defined-networking-market-1607> [Accessed 20 November 2019].

Lee, Y. & Chiueh, T. 2014. OpenFlow switch and methods for packet exchanging thereof, SDN controller and data flow control method thereof. U.S. Pat. US20160142285A1.

Madhava, K. 2018. What is Software-Defined Networking (SDN) Application? WWW document. Updated 27 April 2018. Available at: <https://lavellelennetworks.com/sdn-applications/> [Accessed 25 November 2019].

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. & Turner, J. 2008. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 69-74.

Miller, R. 2016. How AWS came to be. WWW document. Updated 2 July 2016. Available at: <https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/> [Accessed 8 April 2019].

Nadeau, T. D. & Gray, K. 2013. SDN: Software Defined Networks. Sebastopol: O'Reilly Media, Inc.

Nguyen, T. April 2018. The Challenges in SDN/ML Based Network Security: A Survey. PDF document. Available at: <https://arxiv.org/pdf/1804.03539.pdf> [Accessed 28 November 2019]

Open Networking Foundation. 2012. Software-Defined Networking: The New Norm for Networks. PDF document. Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf> [Accessed 28 November 2019].

Open Networking Foundation. 2016. Open Networking Foundation and ON.Lab to Merge to Accelerate Adoption of SDN. WWW document. Updated 19 October 2016. Available at: <https://www.opennetworking.org/news-and-events/press-releases/open-networking-foundation-and-on-lab-to-merge-to-accelerate-adoption-of-sdn/> [Accessed 9 April 2019].

Park, B. 2013. History and Significance of Nicira. WWW document. Updated 26 September 2013. Available at: <https://www.bhagwad.com/blog/2013/technology/history-and-significance-of-nicira.html/> [Accessed 9 April 2019].

Petters, J. 2019. How to Use Wireshark: Comprehensive Tutorial + Tips. WWW document, Updated 29 August 2019. Available at: <https://www.varonis.com/blog/how-to-use-wireshark/> [Accessed 2 December 2019].

Radware Security. 2017. History of DDoS Attacks. WWW document. Updated 13 March 2017. Available at: <https://security.radware.com/ddos-knowledge-center/ddos-chronicles/ddos-attacks-history/> [Accessed 25 November 2019].

Riveners, L. 2016. Why DevOps Should Care About SDN. WWW document. Updated 2 September 2016. Available at: <https://devops.com/devops-care-sdn/> [Accessed 8 December 2019].

Siever, E., Figgins, S., Love, R. & Robbins, A. 2009. Linux in a Nutshell, Sixth Edition. Sebastopol: O'Reilly Media, Inc.

Thimmaraju, K. 2018. CVE-2018-1000155: Denial of Service, Improper Authentication and Authorization, and Covert Channel in the OpenFlow 1.0+ handshake. Email 9 May 2018. Technische Universität Berlin.

Veritis Group Inc. 2019. DEVOPS - A Successful Path To Continuous Integration And Continuous Delivery. PDF document. Available at: <https://www.veritis.com/wp-content/uploads/2016/09/devops-a-success-ful-path-to-continuous-integration-and-continuous-delivery-white-paper.pdf> [Accessed 25 November 2019].

VMWare. 2018. 20 Year Anniversary Timeline. WWW document. Available at: <https://www.vmware.com/timeline.html> [Accessed 7 April 2019].

VMWare, Inc. 2019. Workstation Player FAQs. WWW document. Available at: <https://www.vmware.com/products/player/faqs.html> [Accessed 26 November 2019].

Yoon, C., Lee, S., Kang, H., Park, T., Shin, S., Yegneswaran, V., Porras, P. & Gu, G. 2017. Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks.

Zinner, T., Jarschel, M., Hossfeld, T., Tran-Gia, P. & Kellerer, W. 2013. A Compass Through SDN Networks. Wurzburg: University of Wurzburg Institute of Computer Science.