ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)

στην ΑΝΑΠΤΥΞΗ & ΑΣΦΑΛΕΙΑ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

MASTER THESIS

“Security Analysis of Software-Defined Networks”

«Ανάλυση Ασφάλειας Δικτύων Οριζόμενων από Λογισμικό»

KONSTANTINOS KARACHALIS

F3312307

Supervisor: Prof. George Xylomenos

ATHENS, OCTOBER 2024

Abstract

There has been a rapid development on the communication and information technologies field (e.g. big data, artificial intelligence, cloud networking) introducing various new obstacles that the internet has to overcome such as omni-present accessibility, high availability, bandwidth, security threats. Deprecated methods in networking concerning manual configuration and management of multiple devices are incommodious and liable to errors, therefore making them unable to harness the potential of the physical network infrastructure. One of the solutions to the aforementioned issue rests within the Software-Defined Network (SDN).

Network devices have three planes, the management plane (responsible for the management, configuration, supervision of the network), the data plane (involves all the activities regarding the packets sent forth by the users from end-devices that comprise this plane) and the control plane (responsible for the performance of the data plane activities, excluding the activities of the end-users). However, as we will see in the second section of the thesis, the focus of the layers according to SDN differs from the traditional network models. The two most interesting features of the SDN are the abstraction of the control plane from the data plane and the provided service of programmable network applications.

This paper will dwell on the capabilities of SDN, latest developments, potential benefits, and disadvantages as well as its security implementations on other devices based on practical examples.

Περίληψη

Υπάρχει ραγδαία εξέλιξη όσον αφορά τους τομείς τεχνολογιών επικοινωνιών και πληροφορίας (λόγου χάριν μεγάλα δεδομένα, τεχνητή νοημοσύνη, δικτύωση νέφους) παρουσιάζοντας πληθώρα νέων δοκιμασιών, που καλείται το διαδίκτυο να ξεπεράσει όπως πανταχού προσβασιμότητα, διαθεσιμότητα, high bandwidth, απειλές στο κομμάτι της ασφάλειας δικτύων. Παρωχημένες μέθοδοι όσον αφορά την διαχείριση των δικτύων, όπως χειροκίνητη παραμετροποίηση και διαχείριση πολλών συσκευών δεν είναι πρακτικές και επιρρεπής σε σφάλματα. Συνεπώς, καθιστώντας τις μεθόδους που προηγήθηκαν ακατάλληλες να φτάσουν πλήρως τις δυνατότητες της φυσικής υποδομής δικτύων. Μία από τις λύσεις για αυτό το ζήτημα αποτελεί το Software-Defined Network (SDN).

Οι συσκευές δικτύου έχουν τρία πλάνα, το πλάνο διαχείρισης (υπεύθυνο για την διαχείριση, παραμετροποίηση, επίβλεψη κομματιών των συσκευών δικτύου), το πλάνο δεδομένων (περιλαμβάνει όλες τις δραστηριότητες όσον αφορά τα πακέτα, που αποστέλλονται από τους χρήστες από τις τελικές συσκευές που αποτελούν αυτό το επίπεδο) και το επίπεδο ελέγχου (υπεύθυνο για την εκτέλεση των δραστηριοτήτων του επιπέδου δεδομένων, εξαιρουμένων των δραστηριοτήτων των τελικών χρηστών).

Ωστόσο, όπως θα δούμε στη δεύτερη ενότητα της διπλωματικής, η εστίαση των επιπέδων σύμφωνα με το SDN διαφέρει από τα παραδοσιακά μοντέλα δικτύου. Τα δύο πιο ενδιαφέροντα χαρακτηριστικά του SDN είναι η αφαίρεση του επιπέδου ελέγχου από το επίπεδο δεδομένων και η παρεχόμενη υπηρεσία προγραμματιζόμενων εφαρμογών δικτύου. Αυτό το άρθρο θα ασχοληθεί με τις δυνατότητες του SDN, τις τελευταίες εξελίξεις, τα πιθανά οφέλη και τα μειονεκτήματα καθώς και τις εφαρμογές του στο πεδίο της κυβερνοασφάλειας σε άλλες συσκευές με βάση πρακτικά παραδείγματα.

Table of Contents

Introduction

In the present day, the emergence of new trends in the network field has been most expected with the rapid development brought by cutting-edge technology. As a result of the demographic increase, the demand for performing big data analytics on various sources of data together with high-quality of multimedia content is growing with the network speeds increasing as well. For instance, high-definition televisions and GPS applications bring forth massive client-server traffic to data centers and big data analysis methods in turn trigger the same amounts of traffic to data centers for data partitioning and a combination of the results. Nevertheless, with the development of the network potential security threats arise with the growing demand for the implementation of protection mechanisms that ensure confidentiality, availability and integrity of the data circulating around the network. The master's thesis will aim at presenting and defining the meaning of Software Defined Network and the architectural structure while offering an insight into its capabilities and the opportunities that arise with its implementation in various systems, as well as the obstacles that need to be overcome. These will be covered in the first section and as for the rest of this paper, it will be organized as follows.

The second section will focus on the layers of the SDN architecture, mainly the application layer, infrastructure layer and control layer. More specifically, in the B.1 section, there will be a discussion with an approach to build routing and switching SDN devices and the obstacles surrounding them when interacting with varying transmission media. The B.2 section introduces the potential issues in performance and operations surrounding the control layer. As for the B.3 section, there will be a discussion of problems surrounding the application layer.

Finally, the third section entails details on the security threats for the network and the implementation of protection mechanisms that address each problem. Moreover, a more practical approach will try to cover the capabilities of the most prominent open-source tools in a compromise event through a use case.

A. What is Software Defined Network (SDN)?

A.1. Definition – Software Defined Networks

GeeksforGeeks (GFG) is an Indian company, which is dedicated to computer science, its various fields, such as the new developments in the networking field and provides coding challenges for aspiring programmers and technology enthusiasts, as well as material for educational purposes. GFG has defined explicitly the Software-Defined Networks as follows:

"Software-defined networking (SDN) is an approach to network management that enables dynamic, programmatically efficient network configuration to improve network performance and monitoring. This is done by separating the control plane (which decides where traffic is sent) from the data plane (which actually moves packets to the selected destination)."

According to this definition, the two distinctive characteristics of SDN are namely the abstraction of the control plane from the data plane and the programmability of the control plane. Nonetheless, these are not recently developed in the field of networking. Several attempts have been made in order to promote programmability in the network field. For instance, ANTS (Active Network Transport System) is an active solution which allows programming at the packet level as well as the modification of network behavior during runtime. Moreover, it is possible to construct routers (software) by enabling network devices to be programmable, such example is FRRouting and their behavior can be adjusted by loading new or changing existing routing software. This allows for dynamic customization of the device's functionality to meet specific network requirements.

As for the abstraction amidst the control plane and the data plane, it has been established in the recent years. Many frameworks like ForCES (Forwarding and Control Element Separation), which was released by Internet Engineering Task Force during 2004 (RFC 3746) in an attempt to create a framework that could decouple the control and the data planes. Another concept of the abstraction between control and data plane is Routing Control Protocol, which was introduced in 2004, and it involves a project in which the OSPF and BGP protocols are replaced by centralized routing decisions so as to improve flexibility and network management.

In the case of Software Defined Networks, the factor which differentiates it from the other approaches is due to the fact that through the abstraction of the control from the data plane, programmability is achieved. SDN is capable of providing programmable network devices instead of turning existing networking devices more complex, like the traditional networking methods dictate. In addition, SDN includes the decoupling of the data plane from the control plane in its architectural structure, without altering the flows of data, therefore making the separation between the two aforementioned planes very effective.

A.2. Benefits & Obstacles of SDN

SDN as previously mentioned is a unique network model that sets it apart from the traditional models and acts as a solution to various problems. Nevertheless, like all the network models it has its benefits and its challenges. A potential characteristic of the Software Defined Network resides on the fact, that it provides an experimentation platform for conducting surveys and testing new network concepts, attributed to its network programmability and the capability of isolating virtual networks through the control plane. This subsection focuses on the benefits SDN provides and the challenges it has to overcome.

A.2.1. Benefits - SDN

Improved Scalability

A characteristic of virtualized and software-defined networks which represents a significant advantage is its scalability. Specifically, scaling virtualized processes and network components is more effective in comparison to traditional networking methods, on the grounds that it eliminates the need for additional hardware attainment. There is no necessity to enhance machines with additional RAM or processing power, or even to purchase new equipment, especially when virtual functions are hosted on cloud servers.

Moreover, automated scaling capabilities can be arranged through cloud service providers. If that demands better security, then these providers are able to perform dynamic allocation of new instances in order to address the additional resource requirements.

As a result, with expanding organizations and with the growth of their infrastructures, their demands for better security will also increase. Therefore, the deployment of security tools will be executed with remarkable ease, facilitating the evolution of various operations.

Enhanced Performance

Many organizations in the field of networks aim to increase the utilization of the network infrastructure as much as possible. On the grounds that various technologies and stakeholders exist within a single network, attempting to accomplish maximum performance for the entire network has proven to be burdensome. Existing methods frequently concentrate on enhancing the performance of specific network segments or improving the user experience for certain services. It is clear that these methods, which rely on localized data without considering cross-layer interactions, can result in suboptimal outcomes or even conflicting network operations. Due to the centralization feature of SDN, they allow for centralized control within a global network and control feedback with exchanged information amidst various layers within its architecture.

Therefore, numerous complex performance optimization challenges could be addressed more effectively through the implementation of well-designed centralized algorithms. As a result, new tactics and measures can be implemented and deployed in order to address and solve conventional problems, simplifying the evaluation process of their impact on the improvement of the network performance.

Enhanced Network Security

The rapid development of virtualization has introduced great obstacles in the management of the networking field. Due to the dynamic creation of virtual machines and their removal amidst physical systems, it becomes arduous to consistently revise firewall rules and content filtering policies. In response to these challenges, the Software-Defined Networking (SDN) Controller offers a centralized mechanism for the management and distribution of diverse security policies throughout the network. By abstracting the control plane from the data plane, the tasks of the SDN Controller are simplified and thus becoming more efficient in applying security protocols throughout the enterprise.

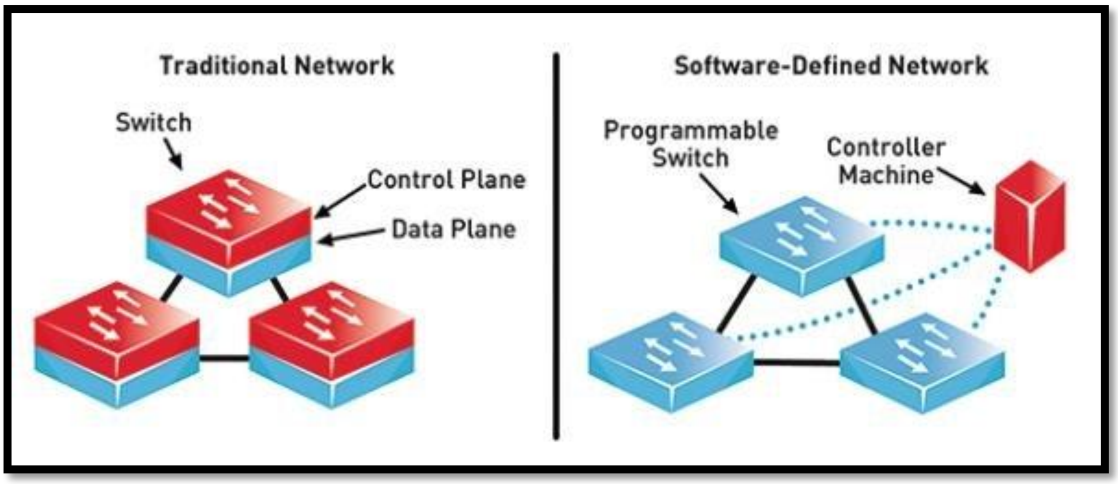
However, the centralization of security around a central point, for instance, the SDN Controller, introduces a single point of failure, which can create opportunities for attackers to manipulate the data flow and consequently result in a potential compromise event of the controller, having a tremendous impact on the network's security. Even so, if the SDN architecture is fortified with robust security measures, it could serve as an efficient tool for managing and enforcing security policies throughout complex and diverse network environments.

Facilitation of Configuration

Configuration is one of the most vital functions in the networking field. Every time new systems/devices or even functions are introduced to existing networks, then to achieve united operation of the network as a whole, it is necessary to include proper configurations. Be that as it may, due to the diversity of network device manufacturers and configuration interfaces, current network configuration generally requires a certain degree of manual intervention. The manual configuration process is both time-consuming and prone to errors. Additionally, troubleshooting a network with configuration mistakes requires considerable effort. The problem is that with contemporary networking architectures, trying to accomplish automatic and dynamic reconfiguration remains a significant obstacle that needs to be overcome. Even so, Software-Defined Networking (SDN) offers a solution. More specifically, via the implementation of SDN, the control plane is unified across various network devices. This unification enables the configuration of these devices from a single, centralized point, allowing for automated control via software. Consequently, the entire network can be programmatically configured and dynamically optimized in response to real-time network conditions.

Traditional Networking & Software Defined Networking

As the evolution in the networking industry is endless and ongoing, future network infrastructure should encourage change instead of trying to meet with detailed precision the requirements for future applications. The primary challenge stems from the prevalent use of proprietary hardware in traditional network components, which hinders modifications necessary for experimentation. Furthermore, when experimentation is possible, it is frequently carried out in isolated, simplified test environments. Such experiments do not provide adequate assurance for the industrial adoption of new concepts or network designs. The differences and changes that SDN brings forth compared to the conventional networking methods are shown in the following table.



	Software-Defined Networking	Traditional Networking
Characteristics	Abstraction of control plane from data plane, programmability of control plane	Introduction of new protocols per every potential issue, network control complexity
Performance	Dynamic Global Control – Cross Layer Information	Limited Information & Relatively Static Configuration
Configuration	Centralized control which enables facilitated configuration via automation mechanisms	Manual Configuration with great liability for errors

Figures A.2.1.1 & A.2.1.2: Differences amidst SDN & Traditional Networks

Software-Defined Networking (SDN), contrasted to the conventional networking methods encourages evolution via the offer of a programmable networking platform that eases the implementation, experimentation, and deployment of new concepts, applications, and revenue-generating services conveniently and flexibly. The high configurability of SDN allows for a clear distinction between virtual networks, enabling experimentation within a real-world environment. Furthermore, the gradual deployment of new ideas can be achieved through a seamless transition from the experimental phase to the operational phase.

A.2.2. Obstacles – SDN

Despite the possibility of accommodating configuration, enhanced performance, enhanced network security and encouraging evolution, SDN is still in its early stages of development. Various conventional challenges have yet to be fully addressed, with standardization and widespread adoption being the most pressing issues. Due to the aforementioned problems, we will present next the challenges SDN has to overcome.

Issues with Interoperability

Networks that were newly introduced, share a common trait regarding the fact that implementing SDN is relatively straightforward, as all network devices are usually compatible with SDN. Alternatively, the transition process to an existing legacy network to SDN presents more obstacles, on the grounds that the legacy infrastructure often supports critical business and networking systems. Organizations and most networking environments must undergo a phased transition to SDN, necessitating a period of coexistence between legacy and SDN technologies.

Legacy network nodes and SDN components can work together through the use of suitable protocols that facilitate SDN communication while maintaining backward compatibility with existing IP and MPLS control plane technologies. This approach minimizes the cost, risk, and service disruptions associated with the transition to SDN.

Deployment Complexity

The deployment of Software-Defined Networking is presented with significant obstacles, especially when it has to be implemented into legacy systems. Conventional network operations are used with distributed control planes, while SDN attempts to centralize the management of networks, therefore making the existence of a substantial configuration of network devices necessary so as to facilitate this shift. Diverse older systems exist that are not compatible with existing SDN technologies, like OpenFlow and as such often require expensive hardware upgrades to ensure greater compatibility. The previously mentioned shift involves restructuring the network architecture, including revising policies, and traffic management systems, and adopting new interfaces for network operations.

Consequently, the demand for integrating security policies and rules for the fortification of the centralized control plane increases, adds to the complexity of deploying SDN systems. Organized integration, planning and testing will be needed in order to avoid having consistent downtime and potential disruption during the network operations so as to retain the maximum possible performance.

Adoption Costs Regarding Software-Defined Networks

Adoption of Software-Defined Networks can be quite costly because it entails direct and indirect expenses, making necessary the need for investment in newly introduced hardware and software, which will have to be compatible with SDN. Moreover, organizations must upgrade their network infrastructure and implement robust security mechanisms so as to ensure the protection of the centralized control plane. This transition will also need a concrete training program for the staff responsible for the management and operation of the introduced SDN structure, which will further increase the expenses needed. Implementation of the SDN usually has as requirements for minimizing downtime, consistent testing and phased deployments, therefore making the adaptation process more costly. While the aforementioned measures represent a substantial upfront cost, they are indeed utilized so as to deliver long-term benefits regarding flexibility, scalability and automation in the network infrastructure of many organizations.

Control & Stability of Performance

In SDN architecture, due to its distinct characteristic involving the abstraction of the control plane from the data plane, performance issues are being introduced constantly. Compared to SDN, in conventional networks, the devices are primarily responsible for the decision-making process. However, in the case of Software-Defined Networks a centralized SDN controller is solely capable of making decisions, therefore causing processing delays. Every time new traffic patterns are being observed, the controller has to process and instruct the network devices accordingly and consequently creating latency, especially in case that the controller is dealing with a high volume of traffic. Furthermore, the stability of the performance will be affected when there are many requests towards the controller and therefore causing it to become a bottleneck resulting in slower response time.

The SDN is capable of offering a platform with the aim of developing and introducing networking methods. However, the shift from traditional networking architecture to SDN architecture is rather complex and challenging. This is why phased configurations, robust security strategies, training of the staff involved, and redesign of the network infrastructure are measures that are needed to address the main concerns of the aforesaid transition. Implementations in Software-Defined Networks are usually restricted to test sites used solely for research prototypes, which are not adequate enough to encourage its application in large-scale, real-world deployments.

A.3 Architecture Model of SDN

Figure A.3.1 shows a Software-Defined Network reference model from the academic paper [5]. According to this model, the SDN entails three layers, the infrastructure layer, the control layer and the application layer.

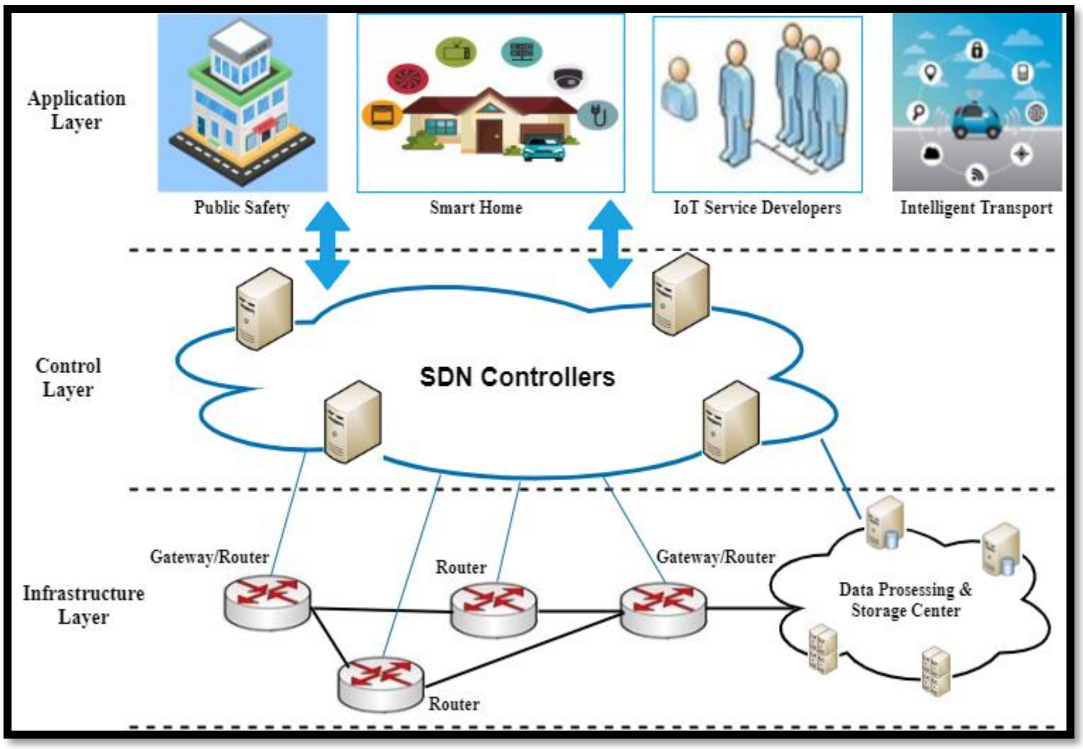


Figure A.3.1: SDN Reference Model

As for the first layer, it consists of mainly switching and routing devices in the data plane. The responsibilities of the systems are limited to status collection of the network and to storing them temporarily into local devices. Afterwards, the systems send them to the SDN controllers of the control layer. The status of the network usually holds additional information regarding the volume of the traffic and the percentages of network usage. Moreover, these systems must process data packets according to the rules provided by the SDN controller.

Secondly, the next layer, the control layer handles bridging the other two layers through its two interfaces. Mainly, for downward interaction with interface layer (from application to interface layer), the control layer instructs the accordingly the controllers in order to utilize only certain functions provided by the devices from the 1st layer. Sometimes, these functions might be involved with the report of the network status. As for the upward interaction with the 3rd layer (from interface layer to application layer), the control layer offers access points with specific services in diverse forms, such as an API.

Applications of Software-Defined Networking are able to access information regarding network status reported from the switching systems through the aforesaid API, making decisions about system tuning according to the access information and carrying out these decisions via the setting process of the forwarding rules of the data packets to the switching devices using the API. Considering that the various SDN controllers are going to exist for large-scale administration domains in the network a new communication interface will be required in order to share the network information and for the coordination of the process for making decisions.

Finally, the last layer is the SDN applications that are created with the aim of completing the requirements of the network users. SDN applications can have access to and are capable of controlling switching and routing devices that belong to the 1st layer. The aforementioned fact is only possible through the usage of the programmable platform offered by the previous layer. The following figure shows the architecture of Software-Defined Networks at the infrastructure level.

The next section will focus on the three layers mentioned in the SDN reference model and their relationships within the network as seen in the figure A.3.1.

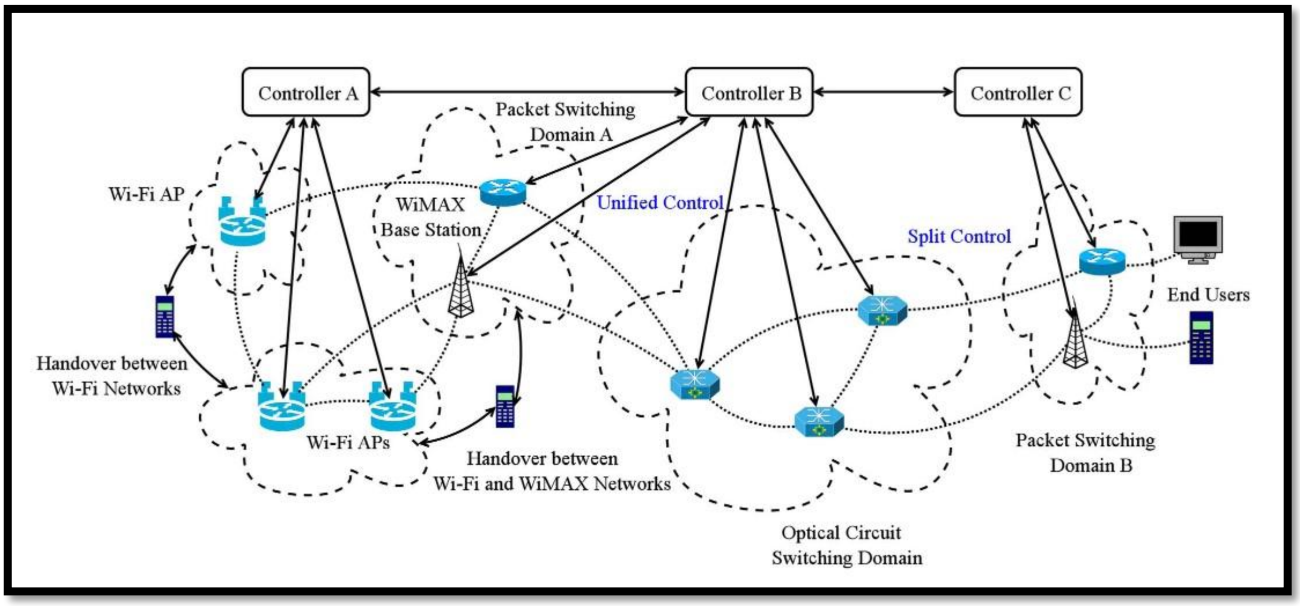

Controller A
Controller B
Controller C
Packet Switching Domain A
Wi-Fi AP
WiMAX Base Station
Unified Control
Split Control
End Users
Handover between Wi-Fi Networks
Wi-Fi APs
Handover between Wi-Fi and WiMAX Networks
Packet Switching Domain B
Optical Circuit Switching Domain

Figure A.3.2: SDN Reference Model in an Infrastructure, with a mesh topology (created by the connection of switching devices, through diverse transmission media)

B. SDN – Layers

B.1. Infrastructure Layer

This is the 1st layer of the SDN architecture, which entails transmission media, such as copper wires, optical fibers, wireless radio and switching devices, like routers, which are connected with each other in order to construct an exclusive network. It is noteworthy to mention that these connections amidst switching devices are only formed via the various transmission media. In subsections B.1.1 and B.1.2 the main focus will be the various operations that can be utilized via the switching devices as well as through the support of the transmission media.

B.1.1. Switches - SDN

The switching devices in SDN consist of the memory and the switching fabric, which are important elements for the two planes: control and data plane. For the data plane, the switch demonstrates the forwarding of data packets via the processor unit, based on the forwarding rules indicated by the 2nd layer.

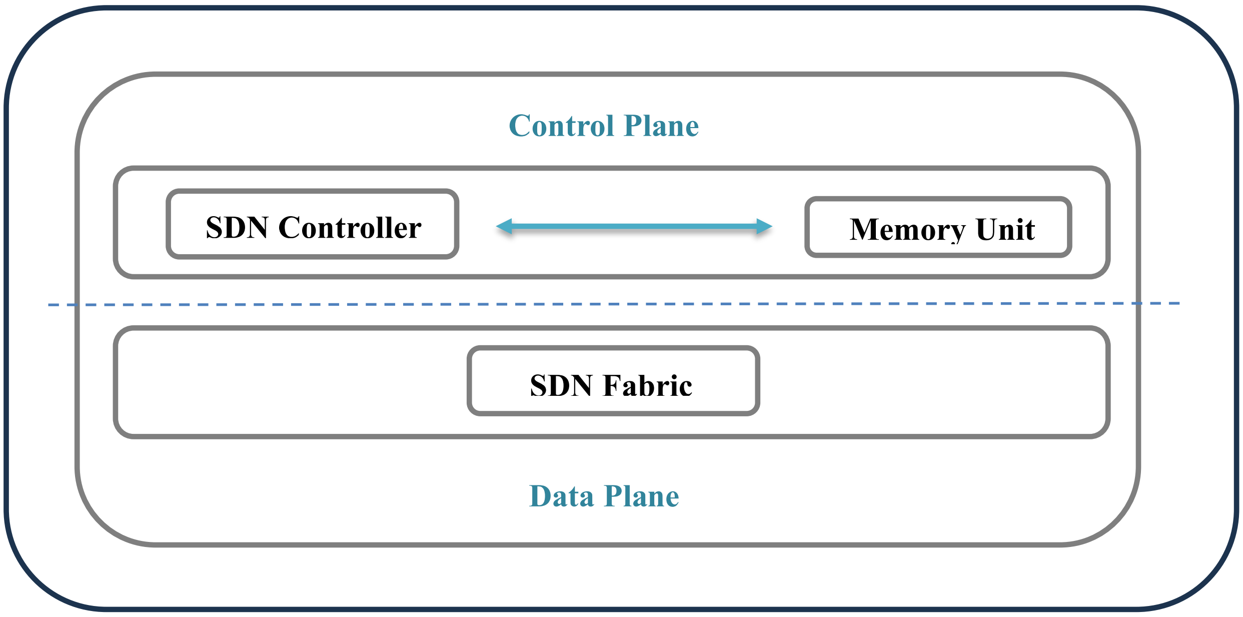


Figure B.1.1: SDN Switching Device Model with a dual-layer logical framework comprising a processor responsible for data transmission and integrated memory dedicated to managing control information.

The previously shown figure attempts to illustrate the design of a switching device in the SDN environment, referring to the aforementioned elements of the switches. As for the control plane, the switching device initiates the communication with the SDN controllers in order to receive the forwarding rules at a switching level, link the tuning rules at a data-link level and finally store them in its local memory.

The previously mentioned upcoming architectural principle offers significant benefits to Software-Defined Networking (SDN), which renders it competitive. Compared to traditional switching devices, which are not only able to handle packet forwarding but moreover being able to execute routing protocols, SDN separates the decision-making process between routing and switching devices. As a result, these devices have as their sole task to report network status and then gather it, as well as process packets according to predefined forwarding rules. Therefore, the design of SDN switching devices is eased due to the separation, making them easier to produce. The reduced complexity leads to a more cost-effective solution.

Nonetheless, this new approach necessitates the development of specialized hardware for SDN-enabled switches. In this section, we are going to review recent advances in switching hardware design, thus covering both the two planes, data and control. Furthermore, we will categorize the most widely used switching platforms and explore methods for testing and educational purposes.

Data Plane

The main task that the data plane has in the SDN switching device is the forwarding of data packets. When a packet is received, the device is capable of identifying the matching forwarding rule and sending the packet to the next destination. Unlike traditional networks, where forwarding is based on IP or MAC addresses, SDN allows packet forwarding to be decided by various parameters, for instance, TCP or UDP ports, VLAN tags, and the switch port where the packet is entered. Nevertheless, using a wide range of criteria for forwarding increases the complexity of processing, therefore creating a trade-off between cost and efficiency in SDN packet handling.

Numerous attempts have been made to formulate and commit various methods with the aim of enhancing packet processing efficiently, with the two key solutions being shown as follows.

Firstly, in PC-based switches, trying to rely greatly on software for packet processing can negatively affect performance. In order to improve the aforesaid situation, the recommended hardware-based method is to boost processing throughput. According to that design, incoming packets that were directed to an onboard Network Interface Controller (NIC), which handles the flow classification in hardware, are now allowing the CPU to bypass the lookup process.

Secondly, the diverse characteristics of great and diminutive flows can now be utilized. Compared to the great data flows, the diminutive ones, are numerous but each one entails only a few packets, for instance, those involved in web page retrieval. These small flows generate the majority of frequent network events, and recognizing this difference can help optimize packet processing strategies.

Control Plane

In the control plane of SDN switching devices, the efficient management of onboard memory is one of the biggest obstacles to overcome in this design. The memory requirements of an SDN switching device are directly influenced by the size of the network. In the case of greater-sized networks, having more memory needs to be one of the main requirements, otherwise, often hardware upgrades will be needed to prevent memory overload. If the memory space is insufficient, then data packets might either be dropped or forwarded to SDN controllers for additional processing, which can negatively impact network performance.

In order to address the aforementioned issue, alternative methods are required to be formulated and applied. To that end, traditional networking techniques used for memory management with the aim of optimization of SDN switches can be potentially adapted. Especially when there are storing rules and minimizing memory usage included. Conventional routing devices utilize methods like route aggregation, along with proper cache replacement policies. Route aggregation consolidates multiple routing records with a shared prefix into one, effectively reducing memory usage. Moreover, a cache replacement policy that is well-built improves the hit rate of packet forwarding rules, and as a result, allows limited memory to be utilized more efficiently. These techniques have many applications on the enhancement of SDN switch designs.

Another consideration that is vital in enhancing SDN switching devices is the careful combination of diverse storage technologies used to balance memory capacity, processing speed, and flexibility at a reasonable cost and complexity.

Different types of storage hardware have varying characteristics. Static Random Access Memory (SRAM) is scalable and flexible, whereas Ternary Content Addressable Memory (TCAM) provides faster search speeds for packet classification. By using both SRAM and TCAM, a balance can be achieved between packet classification performance and flexibility.

Categories of Switching Devices

Switches usually fall into the following categories in SDN as demonstrated in Figure B.1.1.2, under the requirements in hardware (software-based SDN switches (general purpose hardware), bare-metal SDN switches (open network) and vendor-specific)).

Types of SDN Switches (Based on Hardware Implementation)	**Software-Based SDN switches – Application in General- Purpose Hardware**	**Bare-Metal SDN switches – Application in Open Networks**	**Application in Vendor-Specific Switch**
SDN Switching Devices	Open vSwitch (OVS)	Open Network Install Environment (ONIE)	Indigo
Flexibility	High, offers a wide range of features and APIs, its characteristics are software-based	High	Low
Processing Speed	low	Medium, usually 1 – 10 Gbps	High, >= 1Gbps
Density of Port	Low, Limited often due to the scarce number of NICs	Medium, usually 48 ports	High, > 48 ports

Figure B.1.1.2: SDN Switching Device Categories – Hardware Specs Table

Software-Based SDN Switches (Application in General-Purpose Hardware)

The SDN switching devices that fall into this category are often applied as software applications, running on Hosting Operating Systems, such as Linux. Hardware PC x64 or x86 is not the only piece of hardware that is highly compatible with the running host operating system, said example includes the OpenFlowClick, which is based on Click Modular Router and is designed to support general PC hardware, running as an extension of the Linux Kernel. Software-based switching devices are usually characterized by low port density, on the grounds that it is constrained by the limited number of network interface cards that are currently present on the device and show slow data packet processing speeds due to their reliance on software processing. Moreover, it is noteworthy to mention that another benefit of SDN software-based switches is their capability of easing virtual switching for virtual machines among the well-known frameworks of server virtualization and cloud computing. Software-Based SDN switches, for instance, Open Virtual Switch (OVS) offer improved network visibility and control in a user-friendly way. Amidst the virtual machines residing on the same physical server being kept on a local server. Alternatively, in hairpin switching all the traffic is routed to the physical switch that is connected with the server and then subsequently bounced back. A detailed demonstration of the architecture behind the OVS is as follows.

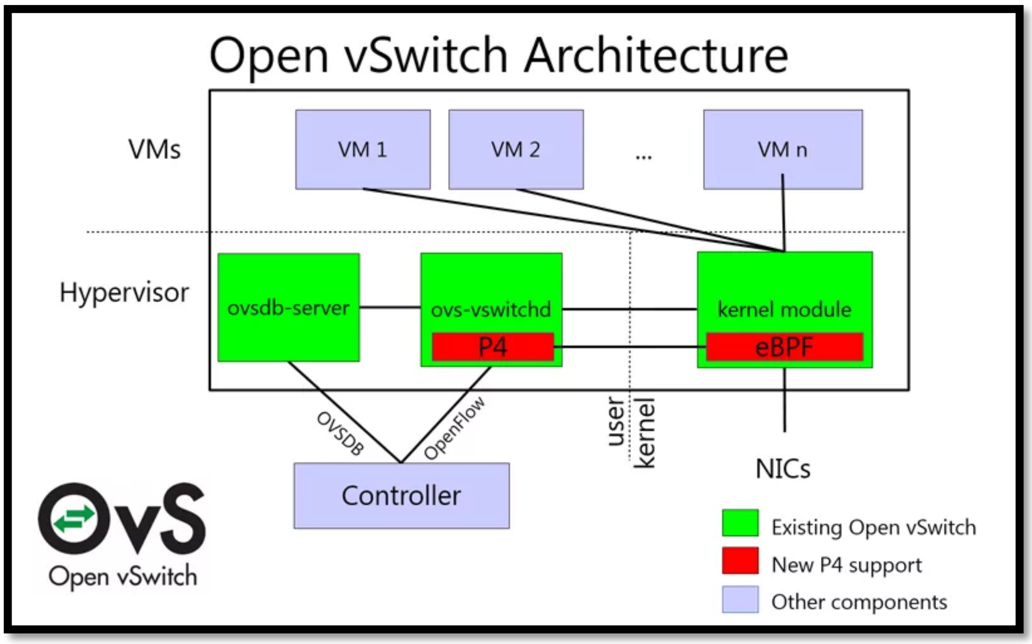


Figure B.1.1.3: Exemplary Illustration of the Open Virtual Switch Architecture

Bare-Metal SDN Switches (Application in Open Networks)

As for bare-metal SDN switches' applications in Open Network Hardware, they are provided with an autonomous vendor and a network development programmable platform used for educational and research purposes. Open Network Hardware Platforms are more industrialized compared to the previously mentioned category and as a result, they have received more support.

Well-known examples of these devices are ORAN and ONIE. Switching devices that are Open Network hardware based are the most often utilized in order to construct SDN prototypes in laboratory settings, on the grounds that they offer higher flexibility and throughput than the other two categories.

Application in Vendor's Specific Switch

Recently, there has been a rapidly increasing number of networking hardware vendors detected, which have been introducing their strategies and solutions for Software Defined Networking (SDN), accompanied by a wide range of SDN-enabled switches, such as the Juniper QFX5100, NEC PF5240, MikroTik Cloud Router Switch and Pica8 3920. In addition to that, there are initiatives, such as the Indigo project, with the aim of enabling SDN functionalities via the application of firmware upgrades on vendor-specific switches which were not primarily designed so as to support SDN features.

In order to make certain that the spirit of evolution will be present in the environment of SDN switching devices, it is of vital importance to include performance evaluation standards. Appropriate functional operations and the enhancement of performance are able to materialize, for instance via continuous testing of the SDN switches. The simplification of performance evaluation standards can be achieved through the utilization of OpenFlow OPS (Operations Per Second), which is a framework that is capable of data packet capture, and timestamping and has great compatibility with diverse packet generation. Moreover, the measurement of performance mechanisms is entailed in the aforementioned framework with control plane operational activities, such as traffic statistics of delayed queries. Since the previously mentioned feature can be applied to each of the two, namely hardware and software implementations in SDN switches, the OpenFlow OPS vital can also be utilized as a tool capable of performing more accurate performance measurements of the SDN switches.

B.1.2. Media of Transmission – Software Defined Networks

Media of transmission such as optical media, wired or wireless ought to be encompassed by the environment of the Software-Defined Network, so as to achieve an omni-present scope of inclusion, as it is demonstrated in the Figure A.3.2. However, it is of great importance to bear in mind that diversity in the transmission media equals a great variety in unique configurations and specific administration technologies. Therefore, Software-Defined Networks ought to be incorporated with the aforementioned technologies in the environments of wireless and optical networking, such as Software-Defined Radios, which facilitates the economic advancement of radio devices. Through this implementation SDN is given the capability to achieve extensive behavioral control over networks, which consists of wireless channels, optical frequencies and data packet forwarding. As a result, Software-Defined Networks are able to achieve better control over the network infrastructure and utilize the resources needed within that infrastructure with greater efficiency. Various technologies on the wireless transmission have been constructed in order to ensure the most enhanced utilization in the said networks. In this subsection there will be a demonstration focused on the most prominent types of transmission media, namely optical fibers and wireless radios.

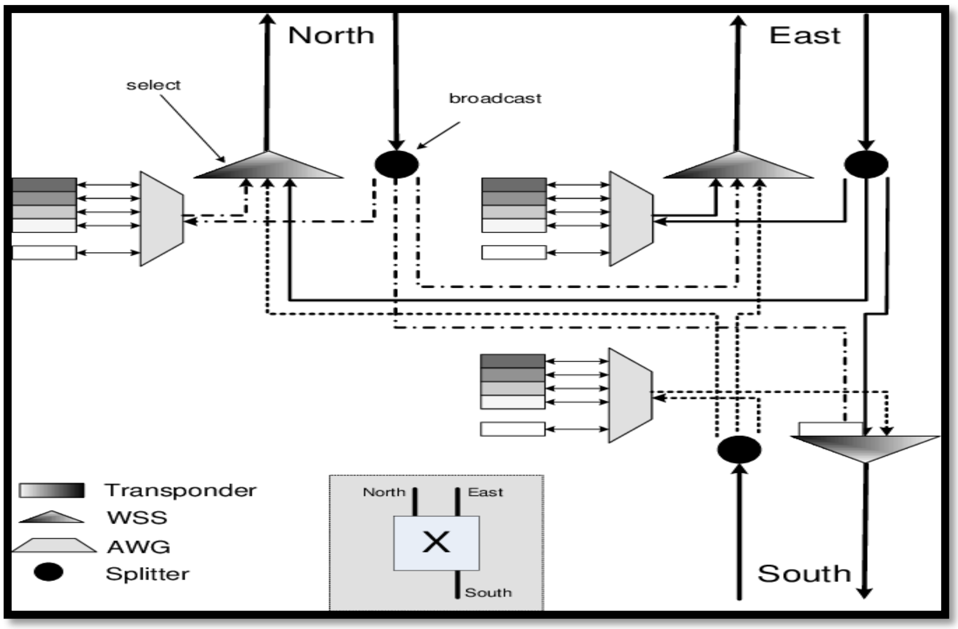


Figure B.1.2.1: Exemplary Illustration of Reconfigurable Optical Add and Drop Multiplexer

Optical Networks (Fibers) - SDN

This type of transmission media is often used as the core network for consolidated traffic, as they provide ways to lower the rate of power consumption and greater capacity as well. The rearrangement of software as a concept, commonly applied in wireless networks, can similarly be employed in optical networks through the use of Reconfigurable Optical Add and Drop Multiplexers. Incorporating these technologies into the SDN control plane enables more accurate and efficient management of the data plane. Unified strategies utilizing a single SDN control plane across both packet-switching and circuit-switching domains are initially considered. As illustrated in the second figure of section A.3, Controller B (figure A.3.2) oversees an optical circuit-switching domain as well as Packet Switching Domain A (figure A.3.2). In light of that, a proposal expanding the parameters used for forwarding rule matching has been made, extending beyond layer 2, 3, and 4 headers of packets to include layer 1 switching technologies, such as timeslot, wavelength, and fiber switching. As a result, the unification of the control plane for both packet and optical networks is achieved. It is noteworthy to bear in mind that the even though the proposed model simplifies control, an enhancement of the optical switches for the circuit is needed in order to support these additional functions.

The use of a virtual switch on each optical switching node to achieve a unified control plane could be applied. In this method, each physical interface of an optical switching node is mapped to a corresponding virtual interface. The messages between the controller and the virtual switch are translated into commands that can be understood by optical switching devices. A similar approach is proposed for integrating legacy equipment with SDN switching devices. During deployment, an added layer is introduced to bridge controllers and legacy switches. Although these methods allow for the reuse of existing network equipment, they introduce more communication latency due to message proxying. On the grounds that, long-distance transmission is inherent in optical networks, it is necessary for an end-to-end data path from source to destination to be managed by various entities, with each one being responsible for diverse segments of the path. In this case, the incorporation of a single control plane throughout the entire data path may not be possible. Split-control approaches, as illustrated in the SDN infrastructure architecture illustration, where Controller B (figure A.3.2) governs an optical circuit switching domain and Controller C (figure A.3.2) manages Packet Switching Domain B (figure A.3.2), might be a more practical solution. These approaches can leverage advanced techniques in optical circuit switching, such as using a Generalized Multiprotocol Label Switching control plane, which is a networking technology that enables fast and reliable network switching of data flows on any type of network infrastructure, for instance, the management of the optical network.

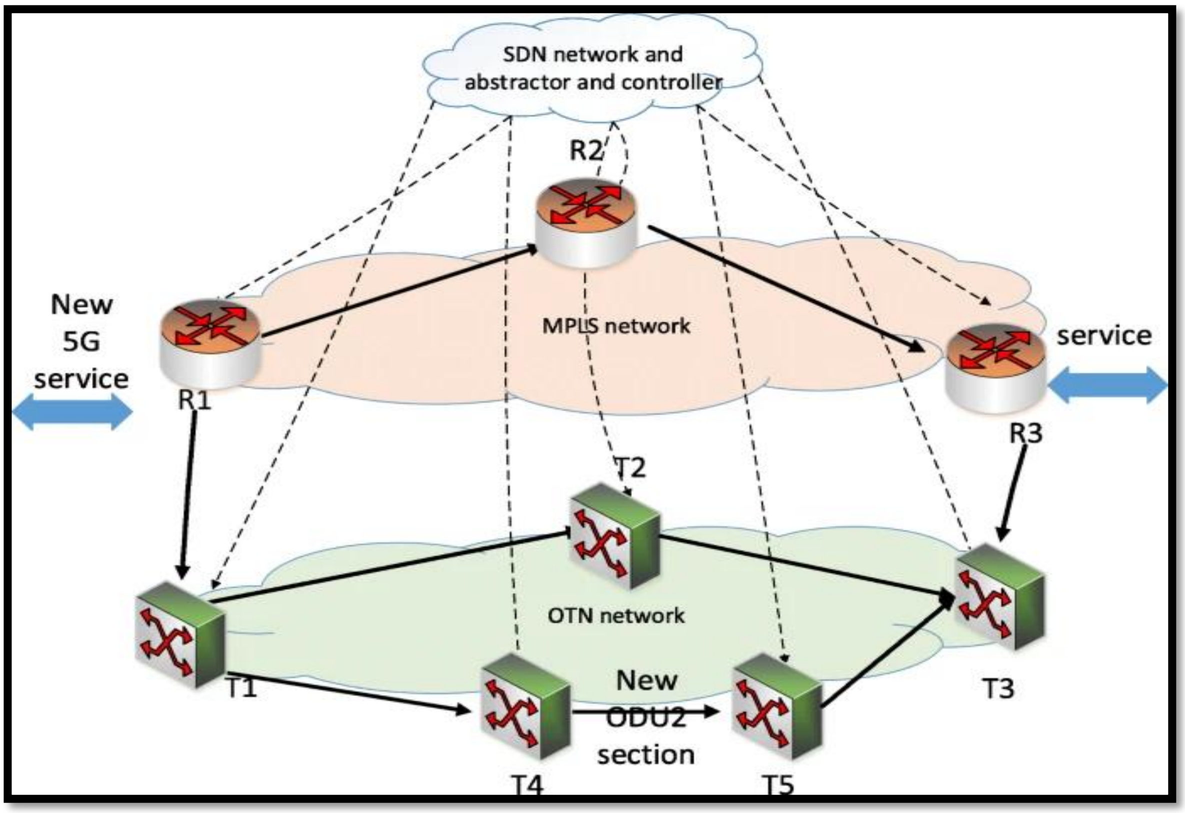


Figure B.1.2.2: Architectural Illustration of Software Defined Optical Networking

SDN Wireless Radios

Enhanced networking technologies have been developed and later incorporated into wireless networking environments so as to improve the utilization of the electromagnetic frequencies concerning network communications. Software-Defined Radio is among the most prominent technologies on the grounds that it enables wireless transmission control through the use of software. Since Software-Defined Radio has various common traits with the SDN, its implementation to the Software-Defined Networks should not pose a challenge. The fact that numerous processing blocks, which have a great computational complexity and are dominant at the physical layer might only have a few differences purely restricted to their configurations.

For instance, the majority of wireless devices utilize the algorithm known as the Fast Fourier Transform varying in lengths. According to the previously mentioned trait, an approach that could be taken is the implementation of an Open Radio Access Network to perform the abstraction of the wireless from the hardware trait and therefore construct an assertive wireless programmable interface for various protocols. As far as the access points and the clients are concerned, they need to keep on passing information to the central SDR controller on the measurement of the performance, the total size of data packets and their count. Afterward, tasks such as the administration of the choice of channels, rate of transmission and the traffic of both the access points and clients that pass through the application programmable interface (with filters: previous and current recordings concerning measurement information) are the responsibility of the central controller. Mainly, Open Radio is able to control physical layer operations via specialized software, therefore rendering it very similar to the Software-Defined Radio frameworks.

As for the issue of the configuration management of software it can be addressed through the incorporation of a Software-Defined Radio system that would allow its controllers to have an interface aiming to improve the central control and the ubiquitous interpretation of Software-Defined Networks. As a result, the central controller of an SDN could enhance its control over the existing SDR devices which could extend to all the network systems existing within its scope.

B.2. Control Layer

As was seen previously in the first illustrations (A.2.1.1, A.3.1) presenting the architecture of the Software-Defined Networks, the 2nd layer, namely the control layer, acts as the link between the other two layers (infrastructure and application). The demonstration of the schematics of the SDN controller, which entails a control operation specialized for the data plane, virtualization factor, agent, administrator and the three interfaces of the controller layer (northbound, southbound, east/westbound used for interaction with infrastructure and application layers and availability purposes) will be the primary focus of this section. Moreover, there will be a brief explanation concerning the problems of the control layer, namely policy and rule validation, and measurement of performance. The characteristics of the control layer will be addressed afterward in the security section.

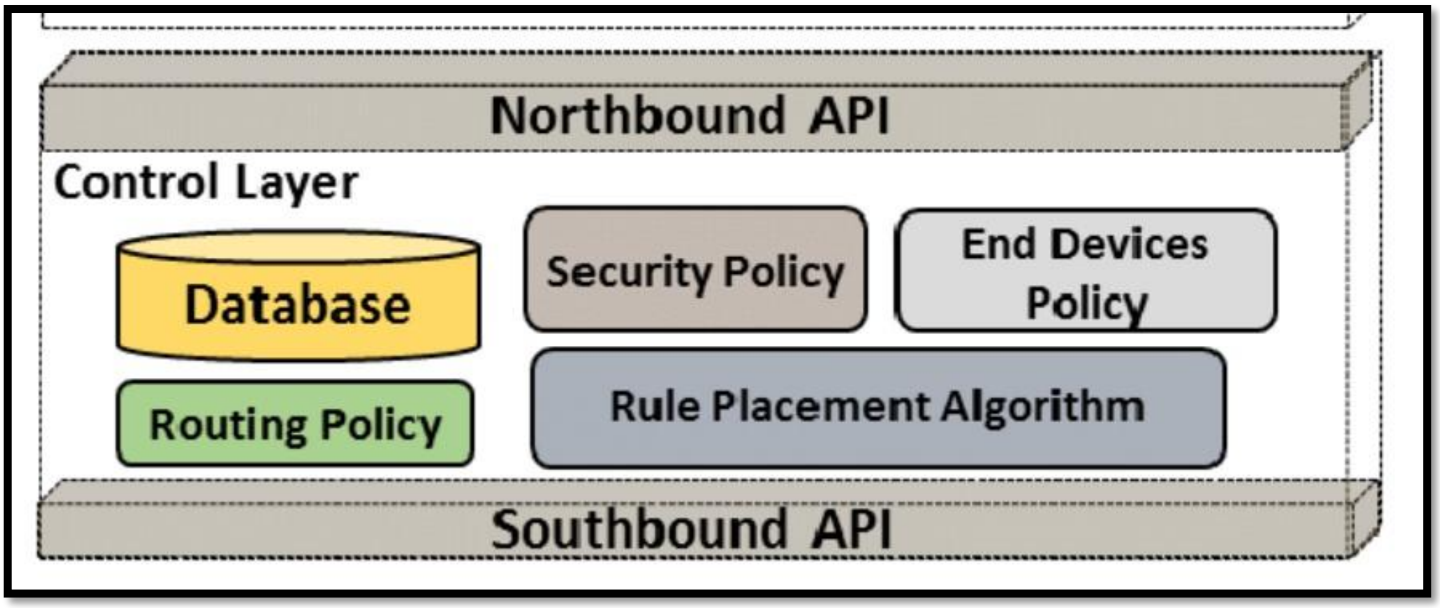


Figure B.2.1.1: Architectural Illustration of the Control Layer

B.2.1. SDN Controller Schematics

SDN Controllers are essentially one of the most vital elements of this architecture, which affects even the level of complexity depending on its structure. Furthermore, it is noteworthy to bear in mind that the SDN controllers' boundaries are not straightforward. The administrator element is solely responsible for the management of the client/server domains. Management of the client and server is important due to the fact that amidst all factors of data, control and application models, require coordination for their functions. Afterward, the next more prominent component of SDN controllers is the control function for the data plane, as it can efficiently use the available resources and redistribute them according to the directives received from either the administrator or the virtualizer that has control over them. The resources usually are processed as information instances with their only access point being the agent. Because the range of the SDN controller spans across various virtual networking environments, the control function for the data plane is needed to consist of operations that work as an aggregation.

In the architectural model of Software-Defined Networking, the virtualization factor has been developed to allocate resources to diverse applications. The Software-Defined Network Controller provides operating services to various applications via the usage of the available resources, policies and supporting functions. More specifically, the functional avatar that provides support to the informational structure instance of an application controller plane interface is the virtualizer. It is manifested by the administrator for every application or client in the SDN scope. Afterwards, the allocated resources are utilized by the virtualizer for the application-controller plane interface which exposes its view to the application clients and sets up the installed policies therefore causing the development of the agent to take place for each of the clients that participated in the aforesaid process. In addition to that, the virtualizer is also responsible for the management of the client requests throughout the application controller plane interface, while ensuring the validity of the requests according to the enforced policies and finally translating them into the form of underlying resources and sending the results to the data plane controller functions and data controller plane interfaces. As a result, in order to achieve enhanced resource management, provided services and integrity of the managed data coordination among the virtualizer, data plane controller functions and SDN controller is required. Every protocol is needed to conclude at a functional entity. The element that is most proper for the relationship amidst the controlled entity and its counterpart is the controller agent model, on the grounds that it can be applied recursively to the SDN architecture. The agent is the aforementioned controlled entity, which represents the client's resources and capabilities within the server's environment.

As mentioned before, the controller of the SDN environment is made up of three application programmable interfaces, which are vital for communication and interaction with the other layers. Following there will be an explanation of each one of these interfaces.

Northbound Interface

The Northbound programmable interface handles the connection link amidst the application and control layers. It is capable of implementing the programmability factor of the SDN controllers into the controllers that are utilized by applications for the 3rd layer through the exposure of the circulating data and the operations that exist inside the utilized SDN controllers. Unfortunately, since the northbound programmable interface is for the most part software, the development of a common version of this interface is not yet feasible.

Southbound Interface

The Southbound programmable interface is primarily responsible for the interaction and communication amidst the Software-Defined Networking Controller and the forwarded in the 1st layer as well as for the communication with SDN Controller and the network systems that are manifested via interfaces. It is noteworthy to mention that usually, the connection that is instantiated amidst the network systems follows the TLS protocol. It is implemented in the aforementioned link with the aim of providing security and authentication factors in the connection. The administration of the virtual and/or physical systems in the SDN by the Controller is only possible provided that it entails the appropriate drivers. Therefore, making the Southbound interface one of the most prominent elements for clarifying the differentiation of the roles between the control and the data planes.

<u>Westbound/East Interface</u>

As for the Westbound/East programmable interface, its primary role is to provide a specific communication interface that enables the synchronization of the state for allocated Software-Defined Networking Controllers within the Control Layer in order to achieve higher availability. In addition to that, it is capable of performing data importation and exportation amidst the SDN Controllers as well as performing state monitoring aiming to ensure whether the SDN Controller is in UP state or sends a notification for a takeover for the appropriate set of forwarded components.

B.2.2. Validation of rules & Policy tables

Rendering policies and rules as appropriate or dysfunctional is a very important process that could affect the stability of the decision-making process for selecting the best routing paths in the Software-Defined Networks and by extension the performance and the functionality of the control layer. SDN networks are made up of various applications and devices that could potentially connect to the same SDN Controller, therefore affecting the overall performance of the Controller. Consequently, this could give rise to the creation of conflicts amidst the existing configurations, which might have negative effects on the coordination process throughout multiple participation units.

Nevertheless, there are various strategies that have been created such as CORA, and role-based source authentication with priority which are able to prevent potential conflicts. Subsection B.2.2 will focus on ways to analyze stability in policies that dwell in intra-switches, the security field and in network domains of SDN.

In order to identify and ensure that the rules and policies for network systems in finite-state examination models are usually the most prominent method for achieving the aforementioned statement. Since then, various strategies and tools have been developed with the aim of addressing this issue for intra-switches. For instance, there are ways (FlowChecker, PreChecker) to perform network configuration encoding for examining overall the behavior of a networking environment in a system which is in a single-state. Moreover, FlowChecker is capable of ensuring validity and security via writing security parameters in the form of a computational tree, while utilizing a

Binary Decision Diagram-based model examination. It is noteworthy to mention that a weakness of the Binary Decision Diagrams is that they can be used to test for intra-switch misconfigurations within a single flow table. Therefore, the FlowChecker is able to take advantage of the FlowVisor, which is able to perform network resource partitioning and effectively isolate this part from the rest of the network. In addition to that, verification of flow policies can be ensured through the implementation of modulo and assertion sets, while VeriFlow studies the verification of invariants in real-time. An added layer, which sits between the SDN controller and the network devices, intercepts flow rules before they reach the network. Although VeriFlow boasts low latency in the checking process, it cannot handle multiple SDN controllers. Through the use of a security-based language to enable flow-based policy enforcement along with network isolation, the aforementioned problem could be resolved. It is incorporated as a NOX application and allows the integration of external authentication sources so as to offer access control. Veriflow is also capable of succeeding in examining in real-time with low latency through the introduction of a proxy amidst a controller and switches aiming to check network-wide invariant violations dynamically as each forwarding rule is updated. Firstly, it performs rule division into equivalence classes based on prefix overlapping and afterward uses the data structure of the prefix tree in order to quickly find overlapping rules. Then, the proxy generates individual forwarding graphs for all the equivalent classes. Last but not least, the OpenFlow Testing Environment is capable of performing black-box testing in physical switches with state synchronization with the aim of validating and cross-checking the integrity of the rules and policies in these switches and the devices in a Software-Defined Networking environment. The next subsection will show a detailed explanation of the ways in which performance is measured in Software-Defined Networks.

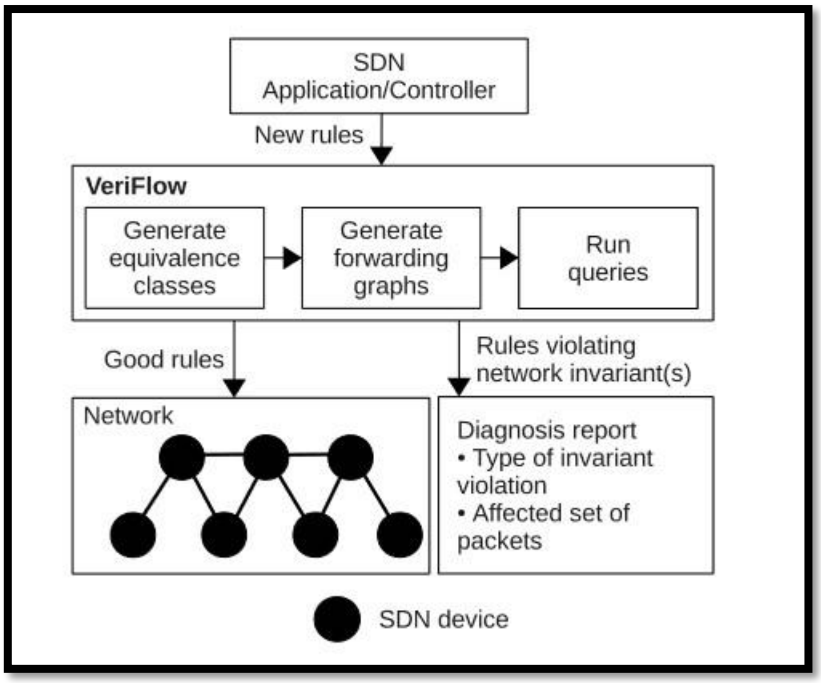

SDN Application/Controller
New rules
VeriFlow
Generate equivalence classes
Generate forwarding graphs
Run queries
Good rules
Rules violating network invariant(s)
Network
Diagnosis report
• Type of invariant violation
• Affected set of packets
SDN device

Figure B.2.2.1: Rule & Policies Interception and Examination for SDN Applications (VeriFlow)

B.2.3. Measurement of performance

Depending on how the control layer functions, the performance of Software-Defined Networks could be greatly affected and, in turn, the scalability factor of the SDN Controllers constrains it. All of the transactions that take place in the control plane are associated with the SDN controllers. For the first packet of each flow that arrives, switches are needed to request the controller for packet forwarding reactive rules. As for the update of rules and the collection of the overall network status communication amidst controllers and switching devices becomes more often. Consequently, the rate of consumption for the bandwidth and the latency of frequent communication could affect the scalability of the control layer greatly. The aforementioned problem must be addressed as it has a negative impact on the SDN performance. Following there will be a demonstration of the ways that various techniques could be used so as to prevent the issue with scalability and by extension that of the overall SDN performance.

SDN Controller Performance - Benchmarks

A common issue in SDN controllers when it comes to the stability of the performance is processing power and the bottlenecks that might be formed causing the situation to become more complex. Therefore, the implementation of benchmarking could be of use to address the issue of scalability on the grounds that it is capable of performance bottleneck identification and at the same time it is needed for the enhancement of the processing speed. There are frameworks such as OpenFlow Controller Benchmark that could perform performance benchmarking for SDN controllers, as they provide statistics of various response metrics (time, percentage, count of packets dropped) for every switch individually. Additionally, there are other frameworks that are able to offer these features and more like Cbench, Iperf with the only exception being that these are better suited for multithreading experimentation. The first framework is capable of conducting tests for the performance of the SDN controllers through request generation for packet forwarding rules, while watching for SDN controller response and offers aggregated statistics of controller throughput and response time for all the switching devices.

SDN Controllers - Dropping Frequency Rate of Ongoing Requests

Since the performance of the SDN environment is affected by the scalability of the control layer, it is needed to stabilize and manage the offset of the request load and its burden on the SDN controllers. The technique that is going to be analyzed for the final subsection of this layer is the restructuring of the way the switches are organized in the SDN. Well-defined divination of the workload and precise role assignments are able to enhance the performance of the control layer. Tools like ONIX and Kandoo are capable of achieving the previously mentioned feats. More specifically, ONIX is a distributed control platform, which can run various instances of the SDN controller throughout multiple locations, working together to manage a network. Allowing ONIX to perform load balancing of the requests effectively preventing the creation of potential bottlenecks via an individual SDN controller. Furthermore, it is capable of sharing instantly the information of the current state of the overall network enabling a constant view of the SDN environment. As for Kandoo due to its two layers (top and bottom layer), it can offload the heavy burden of the requests on the SDN controller through replication of the SDN Controllers at the 2nd layer while the 1st layer is responsible for management of the network view and decision-making process like forwarding.

B.3. Application Layer

The final layer of the SDN architecture resides above the control layer as showed in the illustration A.3.1. Through the previous layer, applications in the environment of Software-Defined Networks are able to effortlessly gain access to a global network view with instantaneous status via the utilization of the northbound programmable interface of SDN controllers. Furthermore, the application layer entails various network programs and applications that are capable of communicating with their desired network behavior and informing the SDN control layer of their requirements. During the past, when the traditional networking methods were prominent dedicated firewalls would be used for load balancing. Nonetheless in the domain of SDN, the management of the data plane is handled by the application layer. Section B.3. will focus on what approaches there are regarding the applications of SDN as a service platform.

B.3.1. Cloud Computing – SDN

In recent years, the cloud field has been advancing rapidly influencing many network environments, businesses and therefore SDN as well. It is capable of providing on-demand services like storage resources, programming infrastructure and even instant updates in software applications and charges always with server usage and virtualization of networks. On the other hand, Software-Defined Networks are able to provide the services required to move beyond the resources needed for computational operations and storage to include more extended services to achieve efficient cloud computing by broadening the horizons of the model of the Infrastructure as a Service. Another vital element of Cloud networking is the data center which in order to function properly is needed to entail high scalability for large-scale projects, dynamic resource provisioning, quality of service for different platforms and wide network availability and visibility. These conditions can be met through the implementation of the SDN. More specifically, SDN can provide the foundation for the enhancement of the IaaS model and by extension for cloud computing by addressing an issue of cloud computing, namely virtualization of switches. Virtualization of switches is used mainly for achieving a communication channel amidst virtual machines that share the same host. Following the traditional networking methods virtualization of switching devices was offered with hypervisors, programs that allow multiple operating systems to share resources of the same physical hosting machine. Nonetheless, it does not provide a satisfactory level of clarity and management. Aiming to address this problem, it is possible to have an edge in virtual switching in virtual machines via the implementation of a previously mentioned SDN framework, Open Virtual Switch (Open vSwitch/OVS). OVS as stated before can monitor and report the overall network status as well as manage data packet rules from SDN controllers. Nevertheless, due to the fact that it does not provide an abundant amount of storage unlike the physical switching devices, it is necessary to incorporate another component with the aim of resolving this issue. What is needed is a system that would enable the proper management of rules that are both virtual and applicable to the cloud environment. Such an example is vCRIB according to the paper presented in USENIX "**Scalable Rule Management for Data Centers**", which can find and enforce the most appropriate rule amidst virtual and physical switching devices, while offloading the increasing traffic and adapting to dynamic changes in the cloud field, like those in the traffic.

B.3.2. Security in Application Layer

In this subsection, the primary focus will be the security implementation in the applications in SDN. Security in networking is a founding part of the cyber security field. Deprecated methods and practices in networking security dictate the implementation of firewalls and proxies in order to ensure the safety of physical infrastructure. On the grounds that there are many differences and unique traits in various applications in the network, the complexity level of enforcing large-scale policies and configuring the aforementioned devices becomes higher. In order to mitigate the repercussions of this issue, Software-Defined Networks can be applied since they provide centralization and merging platforms that examine thoroughly policies, rules and configurations to ensure that the implementation meets the security conditions needed to prevent potential events of security breaches. The SDN is able to collect the status of the overall network rendering it capable of monitoring and analyzing various patterns in traffic to notify instantly and examine security threats. For instance, distributed denial of service (DDoS), replay attacks can be identified in an instant through the previously mentioned feature of the SDN. Also, due to its programmability factor, it offers more centralized control of flows of packet traffic and as a result, the reports of SDN can be applied/transferred directly to systems such as intrusion detection systems (IDS) and intrusion prevention systems (IPS). In case there is an attack detection, then the SDN could install packet forwarding rules to switches with the aim of blocking the attack traffic from entering and propagating in a network. The feature of centralized control that the SDN has, enables the dynamic quarantine of compromised hosts and authentication of legitimate hosts according to the information obtained via the request of end hosts, requesting a Remote Authentication Dial-In User Service (RADIUS) server for users' authentication information tainting traffic or system scanning during registration. Therefore, through the previously mentioned strategies, it is possible to ensure a high level of network security for applications in the SDN.

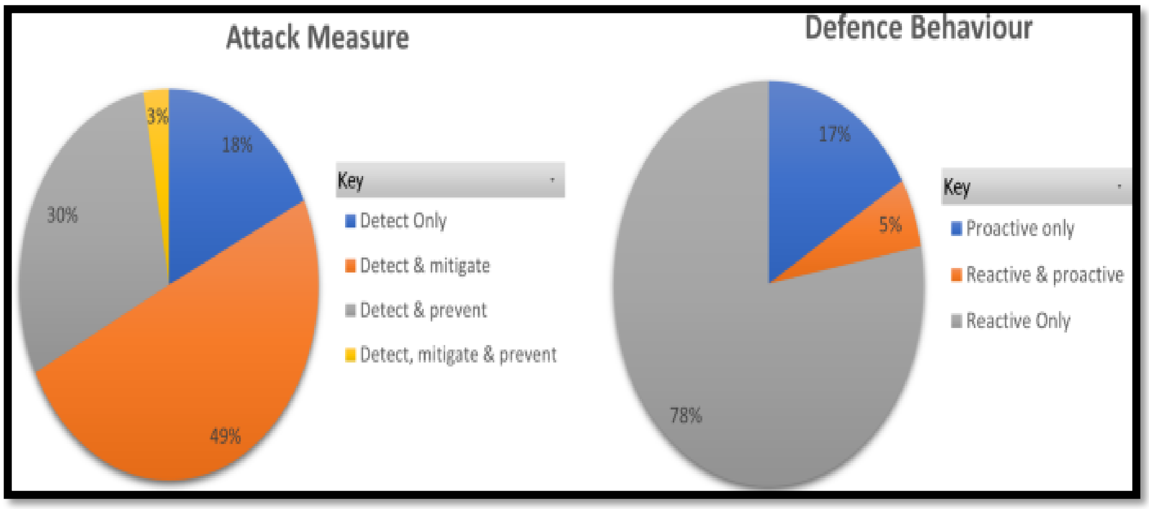


Figure B.2.3.1: Statistics on the behavior of the implemented strategies of the SDN in network security

B.3.3. Management of Network Applications

The most common issue that network applications face is the configuration errors that afterward cause various types of failures. Most of the time, the downtime in network infrastructures is the result of human mistakes during the configuration process. Unfortunately, well-known networking tools such as ping, traceroute, tcpdump cannot offer automated solutions for maintaining the network. In comparison to these conventional methods, SDN is capable of providing centralized and automated revisioning and enforcement of rules, policies and thus effectively preventing configuration errors, while achieving the delivery of automated maintenance and reports regarding the state of the overall network. Fortunately, the development of tools such as network debugger has already taken place, aiming to identify and detect the reason behind networking errors. With Network Debugger each time that a switching device comes into contact with a data packet the SDN controller is notified via a specific way, "postcards" and afterward the SDN controller constructs a backtrace aiming to perform network debugging. To conclude, these tools together with the capabilities of the SDN are able to act as a solid way of performing automated maintenance of the network, alleviating the burden of the tedious manual work that is required to achieve the aforementioned.

C. Network Security - SDN

From the earlier sections, the principles behind the Software-Defined Networking concept have been elucidated and it is understood that it can serve as a breakthrough point with the aim of further extending the capabilities of the existing network facilities. Nevertheless, in many software applications, and networks no matter the traits they possess and the functions they have, there is not a single one that is not susceptible to cyber security attacks as the following sections will demonstrate. More specifically, there will be a detailed explanation regarding the cyber security vulnerabilities found in the architecture of the Software-Defined Networks, the security threat that various attacks pose to the aforementioned network concept, strategies and methods that are able to act as countermeasures and some use cases that demonstrate in a more practical way how attacks can be performed in a virtual SDN environment.

C.1. Vulnerabilities of SDN Architecture

Even though the conventional networking architectural structure was not able to offer the flexibility that the SDN architecture provides, and its components were more dispersed, the latter has its core elements more concentrated and therefore its threats in cyber security are focused on certain aspects of the SDN architecture. Following the susceptible traits of the SDN architectural nature will be shown.

C.1.1. - Susceptibility – Software-Defined Network Controllers

Various operations and decision-making processes like routing, packet forwarding rules, monitoring, gathering of information, and configuration of the networking environment revolve around the SDN controllers. The fact that the Software-Defined Networks have been designed in a way that a single entity (controller) handles most of its functions and daily operations further limits the target scope of potential malicious actors and simplifies the attack performance. In addition to that, the fact that the SDN is compatible with cloud computing due to its programmable nature, so, provides the attacker with many ways to maneuver around the network and implement attacks with greater efficiency. As a result, in the event that the SDN controller is compromised and there is only one controller, then the malicious user will be able to negatively affect the entirety of the SDN domain. Moreover, in case there is only a single SDN controller in the aforementioned environment, since there can be more than one controller, if the targeted controller is flooded with malformed data packets with attacks such as UDP flooding and DNS poisoning then the controller becomes swiftly a single-point-of-failure (SPOF), thus causing the performance of the Software-Defined Network to drop rapidly and all of its functions will cease to work.

Additionally, it is noteworthy to mention that in the event that the SDN environment has multiple controllers, without proper implementation of security protocols, like TLS/SSL, due to the requirement of communication amidst the controllers for the maintenance of the stability of the entire network, the data packets that are circulating the SDN are susceptible to interception techniques such as packet sniffing rendering them a potential source of information for the malicious actors that aim to target the network.

C.1.2. - Susceptibility – Open Nature of Application Programmable Interface (API)

The architecture of SDN is defined by open application programmable interfaces and as such the aforementioned trait causes the Software-Defined Networks to be more vulnerable to cyber security attacks. Open programmable interfaces due to their nature tend to unintentionally expose vulnerabilities that software in SDN tend to have. For instance, conflicting interactions and rules can be caused by the presence of various software applications that have different goals, therefore causing rules with disturbed flow to be integrated into switching devices creating an inconsistent and chaotic behavior of the devices belonging to the SDN domain. Another example of vulnerability of the software application lies in the fact that the SDN applications might be due to the workload and the cost-efficiency goals handled by cloud-service providers and since there are many software applications that must be developed and controlled by various entities (cloud), sometimes there is an issue of unauthorized activities performed by the said SDN applications without being able to view what is causing this. On the grounds that, applications handled by the cloud are the responsibility of third parties, the programmers are unable to discern the main reason that causes the aforementioned issue and by extension do not have any authority on the matter of security. The susceptible elements that are known due to the open nature of the APIs give the opportunity for malicious actors to analyze different patterns and orchestrate an attack plan. Additionally, due to the trait of SDN controllers to offer many open programmable interfaces to the attackers, potential backdoors upon which they are able to perform malicious software embedding, for example, trojan viruses. As a result, it is noteworthy to implement robust security policies and incorporate mechanisms that thoroughly evaluate the open programmable interface provided by the SDN controller. Bear in mind that the software applications that have been previously mentioned, are developed on a controller in the SDN architecture and it is stationed at the same hosting physical hardware device with the controller itself. In case the said software calls functions of the controllers via the link of the control and application layers, northbound application programmable interface then, if an attacker succeeded in compromising the controller, then the malicious script that is embedded on the controller will be executed, causing all sorts of undesirable activities. Consequently, making the application layer as another susceptible element of the SDN architecture that the malicious actors can exploit.

C.1.3. - Susceptibility – Software-Defined Network Switching Devices

Another basic part of the SDN architecture is the switching device. The basic trait of the SDN switching devices is the fact that compared to the conventional switches when data packets are received, they are not required to enter a flow for matching and exiting, but they will attempt to communicate with the SDN controller which will perform decision-making process, directing the switching devices exactly what their tasks will be regarding the data packets. For example, a well-known directive the switches receive from the SDN controllers is the forwarding rules for the data packets. Nonetheless, as previously mentioned in case the attacker performs packet eavesdropping in the event the communication is not secure, the malicious actor will be able to take advantage of the susceptibility of the link amidst controllers and switches, which consequently will result in rule tampering and malformed packet insertions. Therefore, the malicious actor has the opportunity to provide fraudulent rules to the said switching devices, effectively manipulating a part of the SDN environment. The case of a vulnerable connection is not limited solely to the link between SDN switching devices and SDN controllers, since it can be further extended to the communication amidst the switches themselves. Inside the Software-Defined Network, many of the data packets circulated and transmitted amidst the switching devices are in unencrypted form, containing vital information regarding the users of the network, resulting in susceptibility to intercepting mechanisms which are easily performed in the communication of switches, since their links are essentially wireless media.

As said previously, the SDN architecture is comprised of three layers, mainly the infrastructure, control and application layers. Even though each one of them is located in a different place in the Software-Defined Network, their frequent communication is required in order to maintain the stability and the high performance of the SDN environment. Thus, as seen from subsection C.1.1. the SDN provides more patterns for malicious entities to perform various attacks, in comparison to the conventional networks. Following there will be an illustration, which will demonstrate the basic points an attacker could take advantage of to seize control of the Software-Defined Network architecture.

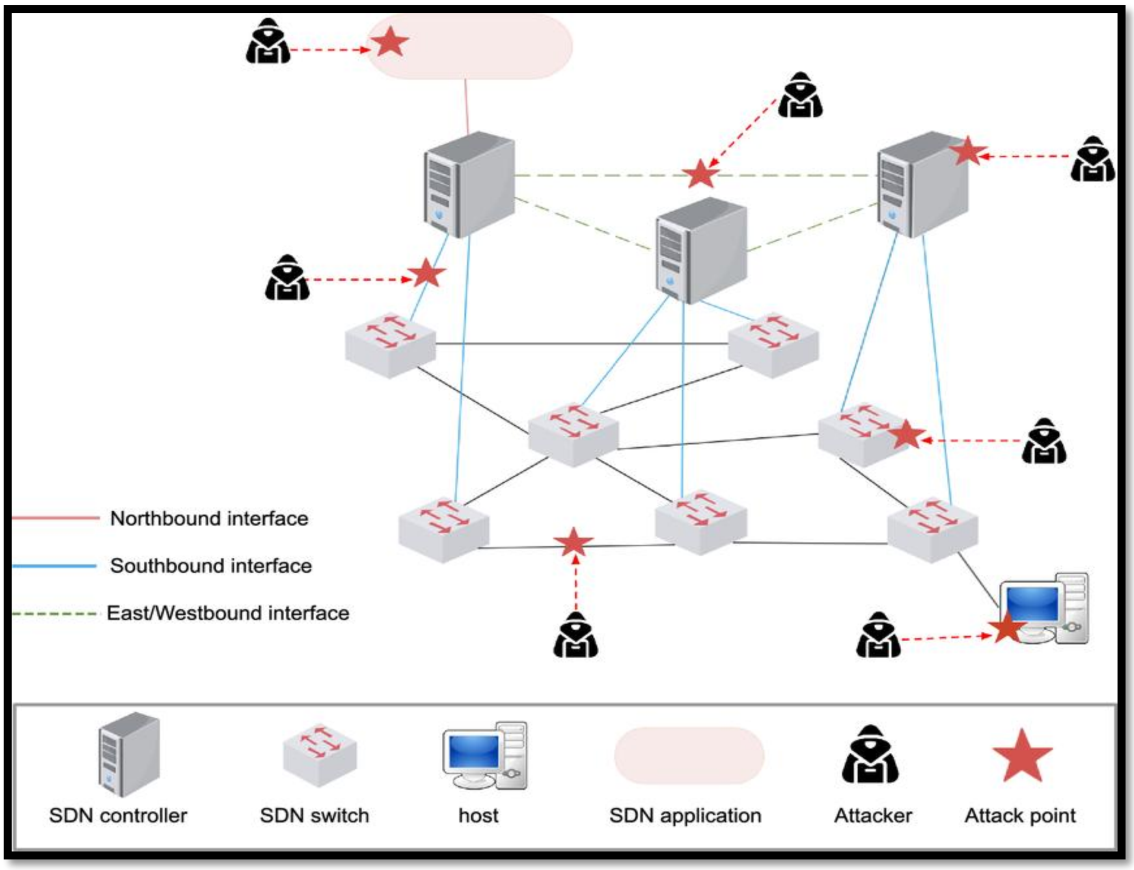


Figure C.1.1.1: Susceptible spots of the SDN architecture that the malicious actors are able to take advantage of

C.2. Security Threats - Software-Defined Networks

The evolution of the Software-Defined Networks is rapid due to the extended research being conducted on this networking concept in recent years. However, the more SDN advances the more eminent the need for the implementation of robust security measures becomes. The main focus of this section is C.2. will be a thorough and detailed explanation of the various security threats to SDN. According to the illustrations C.1.1.1 & A.3.1, the threats can fall under the following categories, about their target, either one of the three layers (infrastructure, control, application).

C.2.1. - Threats – Software-Defined Networks – Infrastructure Layer

As has been stated previously, this is the 1st layer of the SDN architecture, which consists of various interconnected switching devices, whose main responsibility is to perform data packet forwarding. In case a compromise event takes place, the data packets will not follow the proper flow as the forwarding rules would have been changed. Furthermore, it is noteworthy to bear in mind that switching devices represent the central entrance of network access for the end users, so the attackers could embed a malicious link to a port service of the said switch. To conclude, it is vital that security threats are identified swiftly via the incorporation of mechanisms that detect and correlate adverse events. For this section, the structure of the SDN switches will adhere strictly to the concept of the OpenFlow protocol. It entails the following basic elements, mainly the flow buffer, table and the client of the OpenFlow. In case a data packet is received from the proper input port, then the aforesaid data packet will be placed in the buffer flow and afterward a search will be conducted in order to locate a matching rule to the message fields of the corresponding packet, like transmission control protocol port. When the ideal rule is found, then the previously mentioned data packet is going to be removed from the buffer flow, and it will be routed to the corresponding output port. However, in the event that the appropriate rule is not found, the SDN switching device is required to send a packet in a message to the SDN controller with the aim of receiving directives. When the decision-making process is completed the SDN controller will perform routing and the insertion of an ideal rule into the table flow. The threats that are prominent in this case are Denial-of-Service attack in order to paralyze the table and buffer flow and a Man-In-The-Middle attack in which a malicious actor could intercept the data packets between the switch and the controller and therefore perform rule modification without either one (switching device, controller) being none the wiser.

C.2.1.1 - Denial-of-Service (DoS) Attack – Flow Table & Buffer Saturation

The OpenFlow protocols bear a trait in their structure which involves changing the existing rules in a data packet in case it does not have a known address. This design creates a potential way for any malicious actor to easily perform Denial-of-Service attacks or even distributed DoS attacks, since all it would take is for the attacker to keep on sending a tremendous amount of data packets with destinations that do not have known addresses during a short time frame via the utilization of a script aiming to generate a high volume of traffic that would cause the table flow to malfunction and stop performing the appropriate forwarding of the legal data packets, due to the fact that there will be no more available resources used for the creation and attachment of the new ideal rules, therefore preventing the overwriting of the rules.

On the other hand, it is important not to forget that another potential target of the previously mentioned attack is also the buffer flow. Previously, it was mentioned that it is needed for the data packet to be buffered in the buffer flow in order to wait for the result of the ideal rule search or the insertion of a new ideal rule, and afterward perform the data packet forwarding. On the grounds that the buffer flow has a limitation regarding its storage, it always marks the data packets that are to be deleted according to the principle of First In First Out (FIFO) with the aim of releasing unnecessarily used storage space. The concept behind the attack strategy is similar to the one mentioned before. The malicious actor can perform DoS/DDoS attacks such as tcp flooding by sending tcp packets that are a part of another data flow, different from the one that is currently on the switching device causing the device to perform a buffer of the data packets that are currently flowing on to the switch, which consequently is forcing the buffer flow to mark the new data packets as deleted in order to free storage space and store the current non-fraudulent data packets to be stored. This causes the buffer flow to malfunction as well, with its performance dropping rapidly.

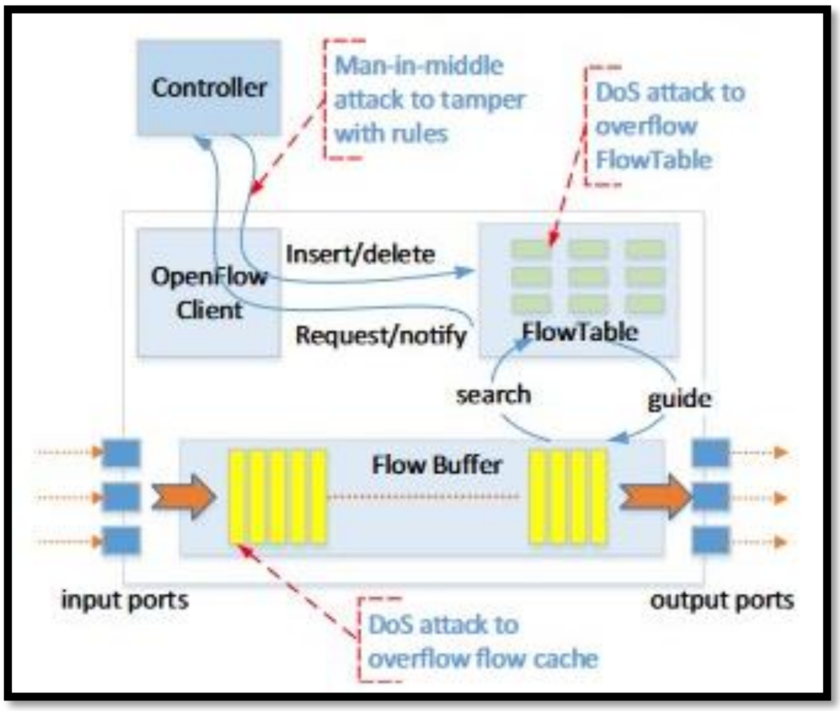

Controller
Man-in-middle attack to tamper with rules
DoS attack to overflow FlowTable
OpenFlow Client
Insert/delete
Request/notify
FlowTable
search
guide
Flow Buffer
input ports
output ports
DoS attack to overflow flow cache

Figure C.2.1.1.1: Attack Vectors in Software-Defined Networks towards OpenFlow Switch affecting the table and buffer flow

C.2.1.2 – Man-In-The-Middle (MITM) Attack – Controller & Switching Device

The concept behind the Man-In-The-Middle attack is to take control over a node in the communication amidst a source and a destination node that acts as an intermediary for the other two, with the aim of intercepting the data packets sent back and forth by each one of them without being detected by either one of them, mainly source or the destination node. It is considered a network intrusion attack, and it could be applied in the Software-Defined Networking environment as well. Since the controllers and the switching devices are frequently communicating with each other, it is the ideal situation for performing this attack. Malicious actors could intercept the data packet, and change the existing forwarding rules which are attached to the switching device in order to take advantage of the data packet forwarding process. This gives the opportunity to the attackers to perform ARP poisoning attacks as will be demonstrated in the following use cases. Moreover, it is vital to remember that it is not necessary for the controller and the switching device to have a physical connection, they might be connected in a virtual environment or even have data packets travel via multiple different switches in order to reach their destination (amidst switch and controller). Resulting, in the susceptibility of all the switching devices and controllers to the MITM attack in the SDN domain. Following we will have an illustration demonstrating in a simple way how the MITM attack works.

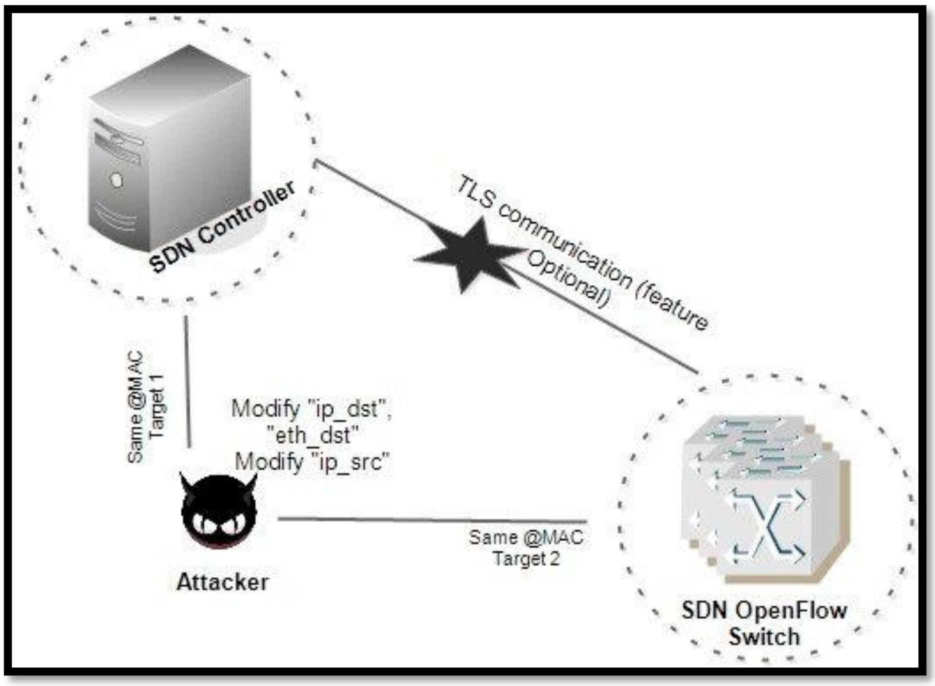

SDN Controller
TLS communication (feature Optional)
Same @MAC Target 1
Modify "ip_dst", "eth_dst" Modify "ip_src"
Attacker
Same @MAC Target 2
SDN OpenFlow Switch

Figure C.2.1.2.1: Man-In-The-Middle Attack Representation in SDN (OpenFlow)

C.2.2. - Threats – Software-Defined Network - Control Layer

According to the SDN architecture the 2nd layer, Control Layer is connected to both the 1st and 3rd layer and in case of a security compromise event both layers can be affected (infrastructure, application) since this affects adversely the data from the infrastructure layer as well as the data packets that are circulated from the link amidst the controllers and the software application in SDN (application layer). The controller in Software-Defined Networks is a core element for ensuring the stability and high performance of the SDN, since it is primarily responsible for decision-making processes that affect many of its components from switching devices to software applications. Therefore, if a malicious actor were to take control of a controller, it would have everlasting consequences on the entire SDN. The SDN switches receive ideal forwarding rules from the controller so as to forward data packets correctly, otherwise the process cannot be carried out properly. On the grounds that the controller is a figure of great importance, it is one of the primary targets for a potential attacker. Following there will be a detailed explanation of the main threats of the control layer, which involve DDoS attacks on the SDN controller and the concept around the multiple SDN controllers' structure.

C.2.2.1 – Distributed Denial-of-Service (DDoS) & Denial-of-Service (DoS) Attack – SDN Controller

The main target of the Distribute Denial of Service attacks is to force the SDN controller to enter a paralysis state, making its functionality completely non-existent and therefore making the controller's services unavailable to all the users of the network. The approach that is taken in the DoS attack is very similar to the approach discussed in subsection C.2.1.1. The aforementioned state is achieved through the exhaustion of the available resources. The attacker is able to create high-volume traffic through the generation of a huge amount of UDP, TCP packets through the utilization of distributed compromised computers also known as zombie computers or bots or via utilizing their own hosting device. Since the packets produced are malformed it means that they cannot be easily discerned from the non-fraudulent packets existing in the current traffic. As explained before, according to the principles of the OpenFlow protocol the switching devices do not handle well new data packets and as a result, it will first store the current packet in its buffer flow and send afterward a packet in the message so as to ask for directives from the SDN controller. As a result, in case of the Distributed-Denial-of-Service attack (as seen in the figure C.2.2.1.1), the controller is forced to manage the high-volume traffic generated by the creation of multiple packets in a short time frame, which ensures available resource depletion. The controller will not be able to process the current traffic, the bandwidth will be occupied by the fraudulent traffic, and this will drop the levels of performance and functionality of the SDN network rapidly.

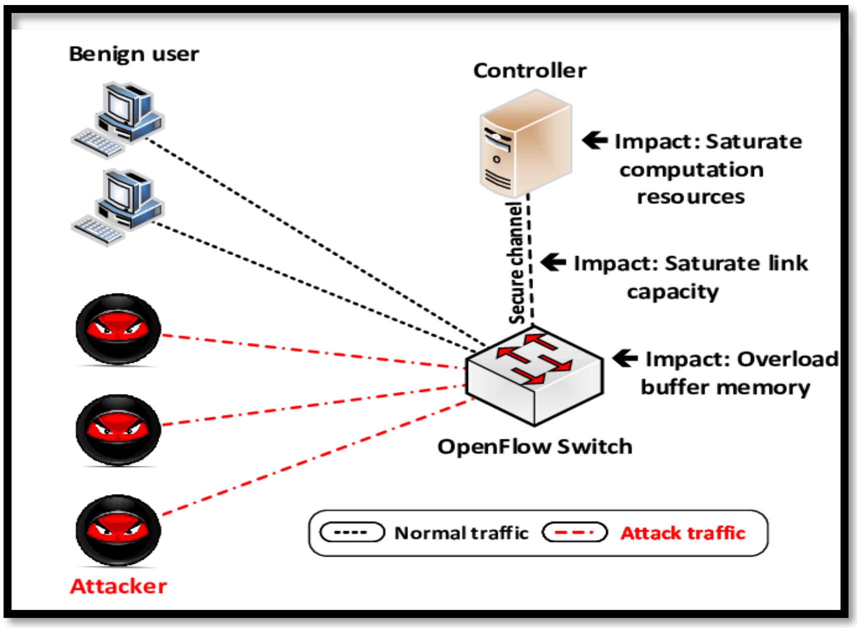

Benign user
Controller
Impact: Saturate computation resources
Secure channel
Impact: Saturate link capacity
Impact: Overload buffer memory
OpenFlow Switch
Normal traffic
Attack traffic
Attacker

Figure C.2.2.1.1: Distributed Denial-of-Service Attack in Software-Defined Networks towards OpenFlow Switch (& SDN Controller)

C.2.2.2 – Model with Various SDN Controllers

Software-Defined Networks had been designed at first with the concept of having one SDN controller, however with the aim of avoiding the loss of scalability and the creation of a Single-Point-Of-Failure (SPOF), various solutions were proposed by many researchers in the field among which was the integration of multiple distributed SDN controllers. Every one of them will be individually capable of specific switching devices and then afterwards these controllers would cooperate with each other, achieving total management of the whole network. Nevertheless, even though there are many controllers responsible for the maintenance of the SDN environment it is still transparent to the infrastructure layer, which means that on the surface it is needed for the controllers to appear as a single one. In the aforementioned scenario, the software application which is spread across various network control environments is required to handle several cyber security issues, like authentication, privacy during the circulation of information, and authorization. Moreover, with the cooperation of the distributed SDN controllers, the existence of many controllers at once and the switching amidst the controllers for the role of the master controller might cause the potential creation of conflicting configurations. Thus, in the previously mentioned architecture, it is of the essence to identify constantly changeable configurations, on the grounds that they could pose a potential security threat.

C.2.3. - Threats – Software-Defined Networks - Application Layer

This is the final layer of the Software-Defined Networks where software applications are set up for executing various operations within the scope of this environment. In order for these applications to complete their tasks, it is of the essence to call functions. Malevolent actors could take advantage of the aforementioned need for the software applications and attempt to inject malicious code through the insertion of malware such as spyware (which is capable of gathering information regarding the activities of the targeted user, in a stealthy way) into the said application. Additionally, since it is not uncommon for the management of the SDN software applications to be handled by third-party service providers, like the cloud, it would prove to be less difficult for an attacker to pretend to be the third-party application and gain unauthorized access. Therefore, making the applications and the controllers themselves prime subjects to attacks, while effectively interfering with their everyday operational activities and negatively affecting the availability and reliability of the SDN domain.

Another issue that arises within the application layer is whether the management and incorporation of security rules, policies and configurations are handled appropriately. Even though the OpenFlow protocol exists and is capable of executing specific functions that provide security identification and detection services for a multitude of software applications, the management of multiple applications is challenging, due to their differences in their programming language, structure and configuration settings. Consequently, this results in misconfigurations and conflicts amidst the various security policies and rules that need to be implemented to ensure protection from potential threats. Following, there is going to be a detailed explanation that aims to prove how the conflicting problems that arise represent a threat and why unauthorized access and spoofing can adversely affect the 3rd layer of the SDN architecture.

C.2.3.1 – Conflicting Issues – Rules – Security Policies – Misconfiguration

As stated before, the application layer is made up of security software applications that are necessary for gaining access to the programmable interfaces of the SDN controllers with the aim of offering a variety of high-quality network services. However, when the number of applications which are set up on the network scope of SDN is high, the complexity of the management process increases on the grounds that more rules need to be incorporated and applied, leading to mass disarray and confusion of the tasks that every service has to complete, arising conflicts between countless configurations. Furthermore, amidst the tediousness of the handling process, security policies cannot effectively cover the security needs of the applications connected to the controllers and their functions, which cannot suffice as a prevention method for attacks.

C.2.3.2 – Unauthorized Access

Most of the Software-Defined Network applications which are being executed on the controllers in accordance with the OpenFlow protocol, are given the appropriate privileges that allow them to gain access to various resources regarding the entirety of the network and even participate in the process of decision-making for the molding of the network's behavior as well as for the directives that will be given. Notwithstanding, it is not the case that all of the network applications that are running on the controllers are developed by the controller vendors. Actually, the task of their development falls under third-party entities, causing a major issue. Anything that is being handled by a third-party actor does not provide visibility for security matters, leading to the inability to create robust, stable and reliable security methods that aim to thwart potential threats. This enables malicious actors to easily gain unauthorized access to the inner workings of the applications (via the injection of malware that could later be executed remotely such as trojan virus) that run on the SDN controllers and effectively have control of a significant portion of the network. Additionally, most of the security applications are optional, this condition does not ensure a trustworthy connection link amidst the software application and the controller. There are of course methods that have been developed for security purposes, such as certification, monitoring, and logging. Nonetheless, a method that is globally accepted for the task of testing the validity of network applications does not exist.

C.2.3.3 – Spoofing

The SDN controllers and the software applications that are running on them require authentication protocols, such as TLS (v1.3), and Kerberos in order to succeed in creating a secure communication. Otherwise, the malicious actor will be able to perform altering information attacks like spoofing. Spoofing is not only limited to changing the source IP address of the data packet, since by masquerading as a non-fraudulent Software-Defined Network controller altering further information, regarding the statistics (for instance the number of packets that are being received) so as to confuse the application that there is a high volume of traffic and causing the application to perform inappropriate decision-making process, becomes possible. Additionally, the attacker could take a different approach and use the spoofing attack to obtain vital information regarding the statistics of various functions gathered from other switching devices to which the controller is connected, therefore achieving illegal access to service level agreement and utilizing it in their potential future attack. All in all, without strong authentication the attacker will be able to gain control of an SDN controller (since there can be more than one) and by extension obtain intel regarding the connected switching devices and applications' daily operational activities and take advantage of them to gain a foothold and launch further attacks on the application layer.

C.3. Countermeasures – Prevention Methods for attacks towards Software-Defined Networks

From the earlier subsections, it could be deduced that many threats are lingering around the SDN architecture for each of the three layers with the majority of them targeting not only the SDN controllers, even though they bear the responsibility of decision-making processes that affect all the layers. Also, prime targets are the errors and issues that exist due to human mistakes or the complexity of handling multiple entities in the same network such as potential misconfigurations. All in all, the need for introducing and implementing countermeasures that contribute to the creation of a robust security network system is becoming urgent. The following sub-sections are going to provide insight about the ways that the aforementioned threats could be prevented.

C.3.1. - Countermeasure – Software-Defined Networks – Infrastructure Layer

C.3.1.1 - Denial-of-Service (DoS) Attack – Flow Table & Buffer Saturation – Countermeasure

OpenFlow protocol is able to interact with other frameworks/technologies that have been developed so as to mitigate events in which the quality of availability is being endangered. The virtual source address validation edge is a platform that entails protection schematics against Denial-of-Service attacks with the architectural design of NOX, which is the original controller of the OpenFlow protocol, and it is capable of serving as a platform responsible for maintaining the control of the network and also offers a wide range of programmable interfaces that are utilized for developing network applications. For instance, in the event of a DoS attack launched towards the SDN controller the data packets that will be sent without having any legitimate rules or rules that match those of the Flow table will be sent to another controller whose role is to perform validation of the source IP address, and in case it is fraudulent, (like spoofing) then a new rule will be created and sent back to the Flow table in order to halt the flow that these packets follow by adding the rule that drops packets from the specified source IP address. Additionally, the incorporation of intrusion detection and intrusion prevention systems could provide great support in finding abnormal behavioral patterns in the traffic and finding the sources of the machines that Denial-of-Service (even distributed) are launched. Implementation of the said systems in switching devices that are connected to the SDN controllers could render the dynamic behavioral access control possible, since with their support security rules and policies could be applied according to the real-time data packet analysis and flow-level information. Therefore, the access control policies are strengthened, and the impacts of the potential DoS attacks could be greatly mitigated. Another measure that can be taken into consideration about the prevention of DoS attacks against SDN controllers and OpenFlow switches is the implementation of a framework known as SFlow.

It is capable of performing real-time based traffic and according to the hardware equipment and network bandwidth its detection speed can reach even higher levels.

It is made up of three main elements as stated by the research paper [40]:

- **SFlow-RT Agent:** It is usually found and implemented in a switching device and its main role is to gather information about data packet samples that circulate on the network and send them in datagram format to the SDN collector.
- **SFlow-RT Analyzer:** As its name suggests, the responsibility of this element is to perform extensive analysis on the received datagrams and offer real-time based information about the data parameters in order to discern abnormal behavioral patterns and identify swiftly attacks like DoS/DDoS.
- **SFlow-RT Collector:** This is the server in which the aforementioned datagrams are being gathered and afterward stored.

This will be further tested on a use-case with Mininet, which is an SDN virtual framework, together with the open-source SDN controller Ryu and the Open Virtual Switches (OpenFlow protocol).

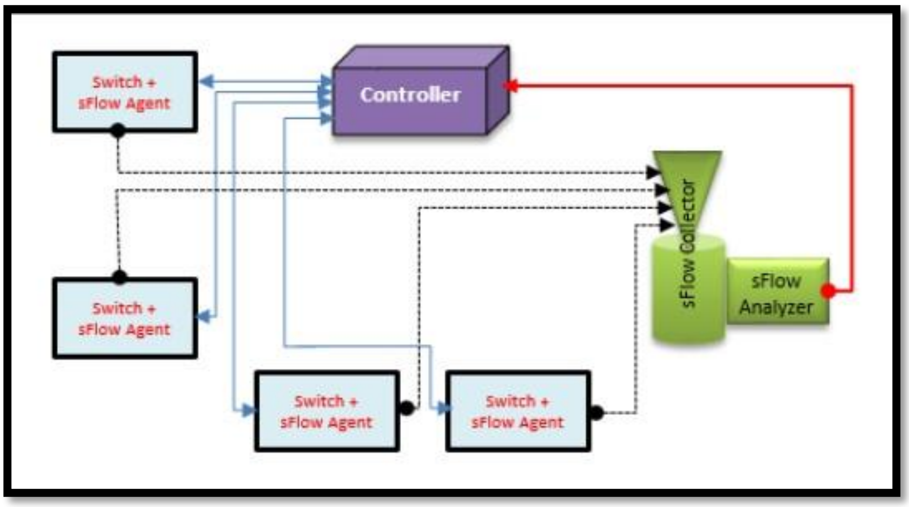


Figure C.3.1.1.1: SFlow Architecture

C.3.1.2 – Man-In-The-Middle (MITM) Attack – Controller & Switching Device – Countermeasure

This kind of attack has targeted many organizations over the years, and it has become well-known in the cyber security industry, and because of that extensive studying has been done behind its principles and its inner workings. The platform NOX and more specifically the NOX OpenFlow controller, which was mentioned previously, entails an extension, which is capable of performing authentication and source authorization in accordance with a role-based model. Through the functional operations it is made up of, rule collision detection is viable. The extension is named FortNOX and due to its ability to perform validation of the alteration mechanisms through the use of digital signatures, prior to the execution of the software applications that utilize them on forwarding rules, it provides a robust security measure against potential eavesdroppers or fraudulent intermediaries. Together with FortNOX another technology similar to sflow that could be incorporated as an intermediary amidst the controller and the switching device is VeriFlow, which is able to perform instant validation on the packets sent between two targets on the network. Apart from the support of existing technologies that have been developed for compromise events, it is noteworthy to bear in mind that another key to achieving the limitation of the adverse effects of a potential MITM attack is to the application of recovery strategies. The OpenFlow protocol entails algorithms that can examine the effectiveness of the stability of the controller's performance by having SDN switching devices keep on pinging or sending messages to the controller and check out its response. In the case that there is an error message or failure response then it uses the backup controllers via its switches (they connect to a backup controller). This also happens in case the switch does not receive a response for a specific amount of time.

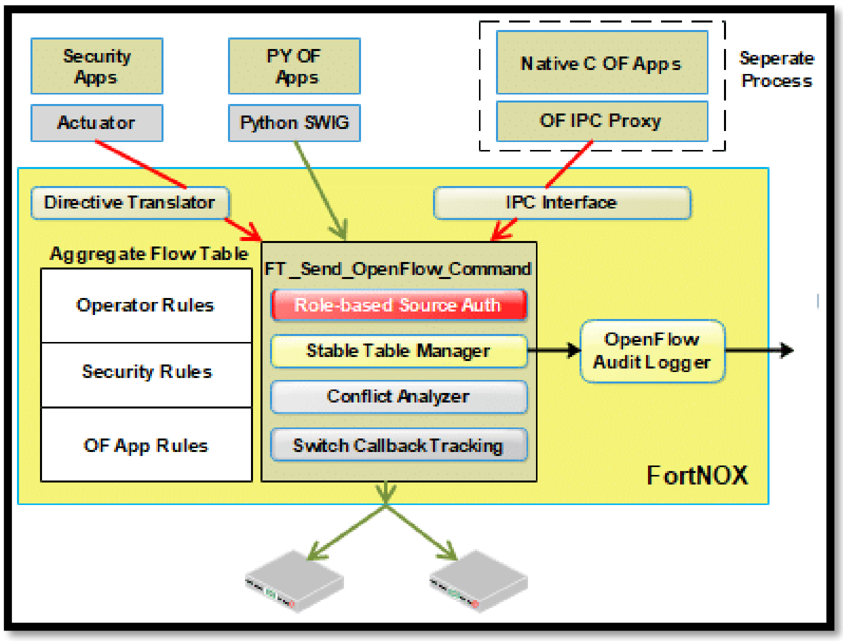


Figure C.3.1.2.1: FortNOX Architecture

C.3.2. - Countermeasure – Software-Defined Networks – Control Layer

C.3.2.1 – Distributed Denial-of-Service (DDoS) Attack & Denial-of-Service (DoS) Attack – SDN Controller – Countermeasure

Similar to the approach that was taken in subsection C.3.1.1, where we took advantage of the analysis of the flow of traffic midst the controllers and the OpenFlow switching devices, a specific framework could be utilized to identify irregular patterns swiftly and detect the DoS-related events. According to the research conducted on ways to mitigate Denial of Service attacks in the paper [41], there is a framework that been developed specifically for DoS related situations. The name of the aforementioned technology is FloodGuard, and it is a security framework that is SDN-aligned and entails two basic elements Analyzer of Active Flow and Packet Migration. The respective roles of the two software modules are as follows, for the former the aim is to perform dynamically a thorough analysis of the data packets in the traffic flow, that is real-time based on the view of the SDN controller, so as to identify which of the flows in the traffic are the result of a DoS attacks. As for the latter, the main task is to perform buffering on the received data packets and afterward sending them to the corresponding controller in order to process them via the utilization of an algorithm that works on a rolling schedule and thus effectively limits the rate in which resources are being consumed by the controller. If the identification of the said attack has been detected, the migration of packets software component will keep on monitoring the abnormal flow traffic aiming to discern the appropriate variables, which are going to support the controller with the task of forwarding flow rules generation and their afterward insertion to the switching devices.

Nonetheless, the aforesaid solution could not have the same effect on the case of the Distributed version of the DoS attack, since it is more potent on the grounds that it is not launched from one machine but multiple compromised devices at the same time that are being controlled by the bot herder machine, which causes high-volume attack traffic towards targets, like the controller. A promising solution to the prevention of DDoS attacks is discussed in the paper [42], which is the Content-Oriented Networking Architecture (CONA). It is a node (proxy) that is situated between the client and the server. CONA is able to communicate with the SDN controller. Furthermore, since all the request messages, data packets that originate from the clients are being examined, intercepted and analyzed by the previously mentioned technology in case the rate of the received data packets, messages (received from the server) surpass the intended value, that flow traffic will be marked as a potential DDoS related event.

Afterwards, with the aim of thwarting this attack scenario the controller will send a proper message to all related CONA agents and redirect the non-fraudulent data packets and messages to another server, effectively mitigating the adverse effects of the DDoS attack. All in all, from the aforementioned it is eminent that these technologies with the support of the SDN characteristics are utilized in order to prevent DDoS and DoS attacks and they might be used as the foundation for further research so that more advanced strategies and methods are formulated for the enhancement of the Software-Defined Network structure.

C.3.2.2 – Model with Various SDN Controllers – Countermeasure

In order to defend against attacks that take advantage of the flaws of the multi-controller architectural design of the Software-Defined Networks it is not enough to implement technologies and frameworks that are limited to a single entity such as the controller, but those that can be applied for the entire control layer. One existing method, which could be incorporated into the SDN 2nd layer, is the load balancing technique, due to its ability to improve scalability and stabilize the functionality of the controllers. Load balancing in SDN can be utilized to gather information from both the application and the infrastructure layers together with the state of the network and apply it to support the controllers in the decision-making process, and therefore effectively limiting the workload of the controllers and elucidating the configuration process of the applications and the controllers alike, preventing the creation of further conflicting issues that could be used by malicious actors for launching various attacks. Moreover, it supports the correct placement (logical) of the SDN controllers improving the security of the controllers and enhancing their scalability.

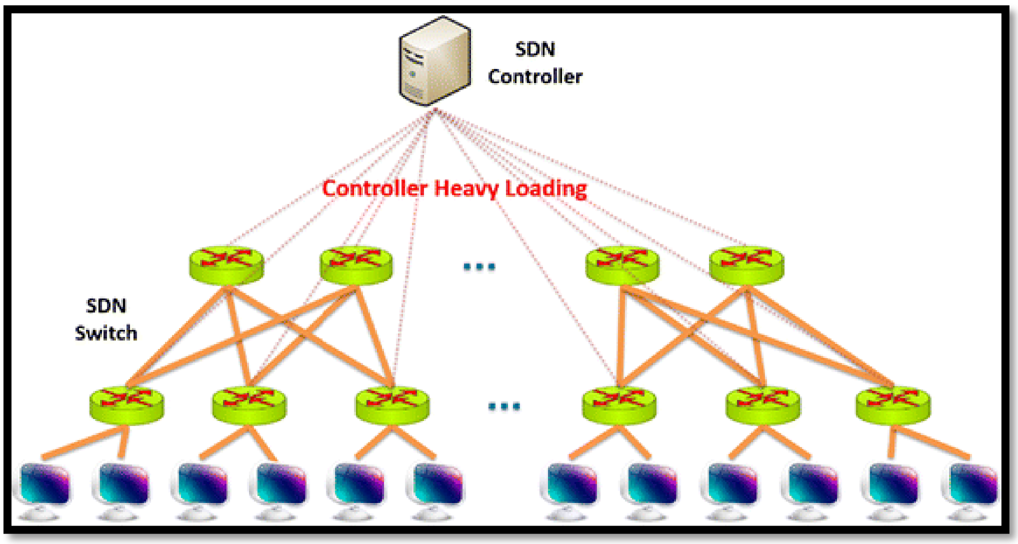


Figure C.3.2.2.1: Load-Balancing in multi-controller SDN architecture

Even though the load balancing technique is efficient, it is not potent enough on its own so as to address the issue of the raising complexity that comes with the management of multiple controllers in the environment of the Software-Defined Network. Hyperflow [43], which is a distributed event-driven controller framework developed for the OpenFlow protocol. In the environment of the HyperFlow, various controllers are running their functional operations at the same time, with every one of the controllers performing decision-making processes at a local level. As a result, the new forwarding and flow rules are being created and issued at a faster rate, enhancing the overall performance of the controllers in the 2nd layer.

Another approach that could be taken for this scenario is the allocation of the SDN controllers with the implementation of resilient controller placement. In the research paper [44] it is discussed that the development of a framework that forwards the suggestion that a controller is ought to fulfill certain conditions in its everyday operations, such as the impediment of the communication amidst the controllers and the switching devices in SDN, aiming to prevent relative attacks with more efficiency.

C.3.3. - Countermeasure – Software-Defined Networks – Application Layer

C.3.3.1 – Conflicting Issues – Rules – Security Policies – Misconfiguration – Countermeasure

According to the research work that has been conducted on [45][46] the frameworks Flover and Anteater are discussed which contribute to the examination of the behavior of the software applications of the SDN network and prevent the unintended creation of conflicts amidst security policies, rules and configurations. The former, is actually a system tasked with the examination and verification of policies, which are issued for the various flows of traffic circulating in the SDN domain, through the utilization of assertion sets. Its incorporation follows the principle of NOX, with its feature the Yices Solver (SMT) and offers verification services for the functional operations of the behavior of a OpenFlow network from the aspect of security. Afterwards, for any abnormality detected in the policies implemented, Flover considers them as conflicts. The aforementioned service is conducted through the continuous gathering of response messages from the controller via batches.

On the other hand, the latter system is capable of performing static examination of potential network misconfigurations or conflicts, debugging and is able to provide validation operations for the infrastructure layer of the SDN. Even though its execution and effect do not last for a long time period, it is still capable of identifying the source of the issues that have already occurred since it runs static tests instead of getting dynamically information based on real-time events.

C.3.3.2 – Unauthorized Access – Countermeasure

In papers [48] and [49], the authors of the corresponding papers discuss technologies that are capable of enforcing verification techniques on the validation of activities conducted inside the Software-Defined network. The first one is verificare, which is tasked with creating models for distributed devices using validation methods. The practical example given for this tool was to formulate a model based on the OpenFlow Network continuously so as to validate its most vital properties and appropriateness. The second platform, vericon it is responsible for the validation of how accurate the controllers' functions are. One of Dijkstra's theories is incorporated via the first-order principle and the ideal network-wise constants, which is the FloydHoare-Dijkstra. The aforementioned experiments showed great promise according to their final results, which were great since irregularities and bugs for wide SDN applications were identified swiftly and as for the process of the validation it was a success.

Finally, according to [50] it represents a new enforcement tool, NICE, in which the examination process of the OpenFlow network applications becomes automated and therefore the validation of the correctness of them is performed at a faster pace. Additionally, it is capable of testing event handlers via model checking that can swiftly examine the current condition of the NOX controllers which were not changed.

C.3.3.3 – Spoofing – Countermeasure

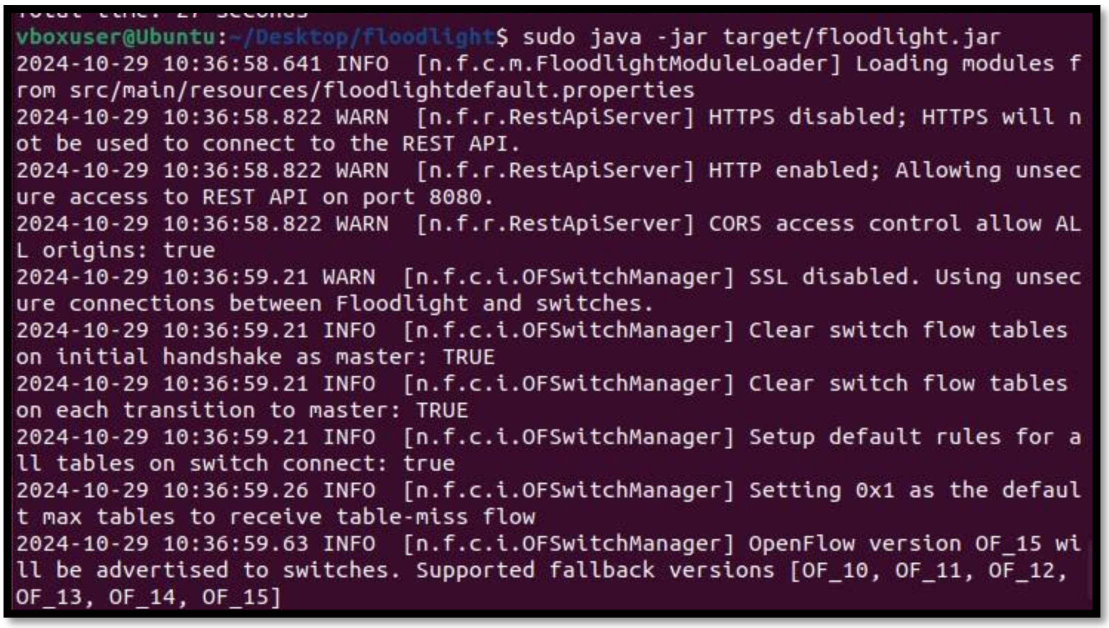
The research paper [38] has been discussing the implementation of the POX controller into the SDN environment, which is written in the programming language Python and usually it is bundled with the Mininet SDN network. According to the research conducted by the authors of the paper, the application which will be developed on the SDN POX controller will be able to defend against ARP spoofing attacks by detecting the malformed requests and replies that force the packets to be redirected to malicious domains. Furthermore, it is capable of monitoring and detecting excessive amounts of data arp packets and therefore prevents them through the installation of rules adhering strictly to the principles of the OpenFlow protocol. From the above, it can be concluded that POX controller can efficiently defend against arp spoofing attacks. Additionally, another way of preventing spoofing attacks is the incorporation of strong authentication protocols of mutual identifiers, such as the certificates that are signed by a trusted certificate authority. In the following section, there will be a demonstration of attacks conducted in the SDN environment, for educational purposes only via use-cases.

C.4. Use-Case: Distributed Denial-Of-Service Attack

For this use case, the main focus will be the examination of the effects of DDoS attacks regarding the SDN controller FloodLight and targeted host inside Mininet, as well as testing prevention and mitigation methods for the aforesaid scenario. In this use-case the hosts h5, h4, h3 will be compromised in order to launch the attack with the topology being simple with 6 hosting devices.

The Software-Defined Network oriented open-source tools will include FloodLight, Mininet, Open Virtual Switch and SFlow-RT with the installation guide being included at the beginning of this thesis. The attack scenario illustrated in this use-case is purely for educational purposes.

The goal of this use-case is to successfully cause the SDN Controller and hosting device to crash or drop their performance rate dramatically by flooding them with an excessive amount of traffic. Following, the SDN Controller FloodLight is launched, which is open-source Java-based controller that supports the OpenFlow protocol and offers a restful application programmable interface in order to issue commands, set rules and interact with the controller. The controller is launched with the command **sudo java -jar target/floodlight.jar** as seen in the following illustration.

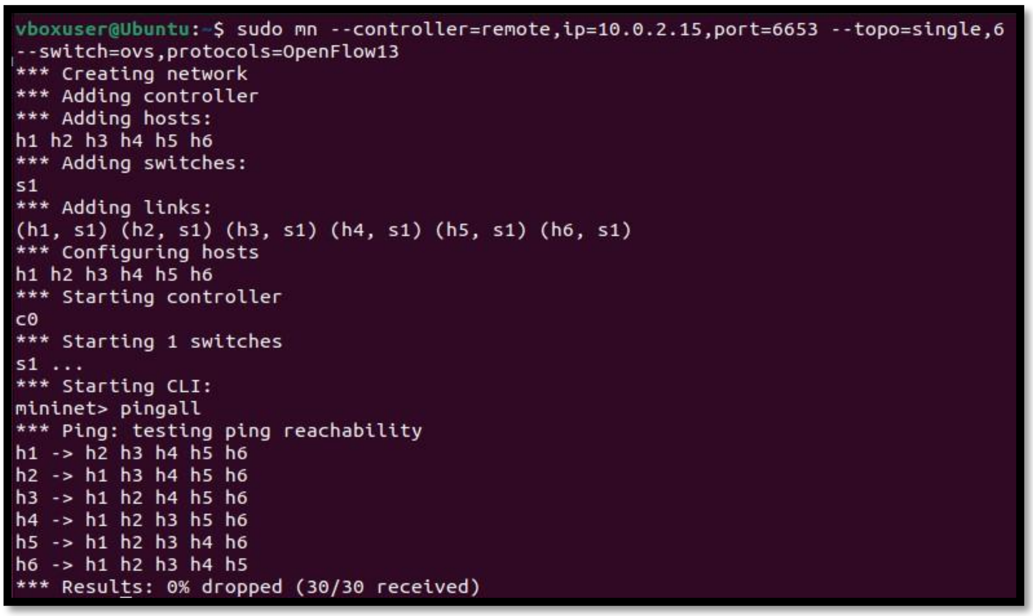


```
vboxuser@Ubuntu:~/Desktop/floodlight$ sudo java -jar target/floodlight.jar
2024-10-29 10:36:58.641 INFO  [n.f.c.m.FloodlightModuleLoader] Loading modules f
rom src/main/resources/floodlightdefault.properties
2024-10-29 10:36:58.822 WARN  [n.f.r.RestApiServer] HTTPS disabled; HTTPS will n
ot be used to connect to the REST API.
2024-10-29 10:36:58.822 WARN  [n.f.r.RestApiServer] HTTP enabled; Allowing unsec
ure access to REST API on port 8080.
2024-10-29 10:36:58.822 WARN  [n.f.r.RestApiServer] CORS access control allow AL
L origins: true
2024-10-29 10:36:59.21 WARN  [n.f.c.i.OFSwitchManager] SSL disabled. Using unsec
ure connections between Floodlight and switches.
2024-10-29 10:36:59.21 INFO  [n.f.c.i.OFSwitchManager] Clear switch flow tables
on initial handshake as master: TRUE
2024-10-29 10:36:59.21 INFO  [n.f.c.i.OFSwitchManager] Clear switch flow tables
on each transition to master: TRUE
2024-10-29 10:36:59.21 INFO  [n.f.c.i.OFSwitchManager] Setup default rules for a
ll tables on switch connect: true
2024-10-29 10:36:59.26 INFO  [n.f.c.i.OFSwitchManager] Setting 0x1 as the defaul
t max tables to receive table-miss flow
2024-10-29 10:36:59.63 INFO  [n.f.c.i.OFSwitchManager] OpenFlow version OF_15 wi
ll be advertised to switches. Supported fallback versions [OF_10, OF_11, OF_12,
OF_13, OF_14, OF_15]
```

Figure C.4.1.1: FloodLight SDN Controller Launched – Ubuntu 22.04 LTS | Logs

Afterward, Mininet SDN Framework must be launched in order to set up the SDN environment and the network topology. It is well-suited for this case, on the grounds that it is a framework OpenFlow-based capable of deploying networks of massive scale on the limited resources of a single computing device, thus being proper for personal experimentation and practice.

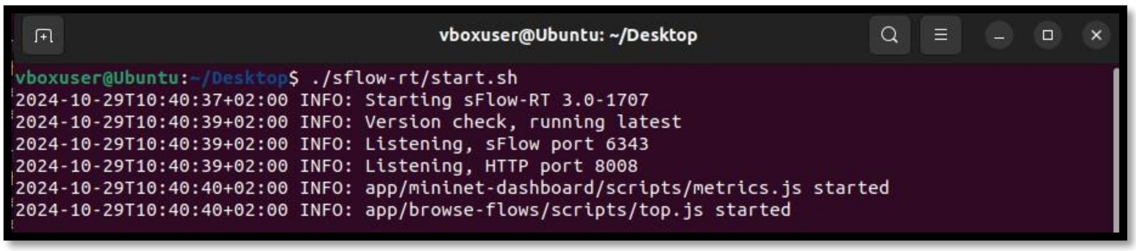
Since FloodLight is launched on the VM's IP address, for this use-case it will be the 10.0.2.15 IP address and therefore, the topology settings will be as follows. PingAll command is needed in this case to verify that all the hosting devices in the virtual SDN environment communicate with each other.



```
vboxuser@Ubuntu:~$ sudo mn --controller=remote,ip=10.0.2.15,port=6653 --topo=single,6
--switch=ovs,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
```

Figure C.4.1.2: Mininet SDN Emulator – Ubuntu 22.04 LTS | Single-switching device Topology

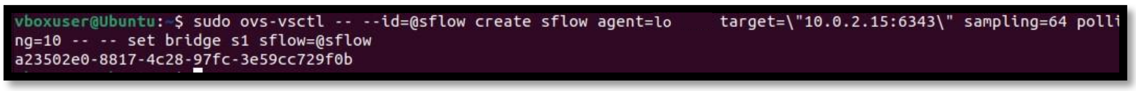
Then, it is of the essence to launch the SFlow-RT open-source network analysis and metrics tool so as to have a display of the current state of the SDN environment in real-time as seen below.



```
vboxuser@Ubuntu: ~/Desktop
vboxuser@Ubuntu:~/Desktop$ ./sflow-rt/start.sh
2024-10-29T10:40:37+02:00 INFO: Starting sFlow-RT 3.0-1707
2024-10-29T10:40:39+02:00 INFO: Version check, running latest
2024-10-29T10:40:39+02:00 INFO: Listening, sFlow port 6343
2024-10-29T10:40:39+02:00 INFO: Listening, HTTP port 8008
2024-10-29T10:40:40+02:00 INFO: app/mininet-dashboard/scripts/metrics.js started
2024-10-29T10:40:40+02:00 INFO: app/browse-flows/scripts/top.js started
```

Figure C.4.1.3: SFlow-RT SDN Analytics Tool

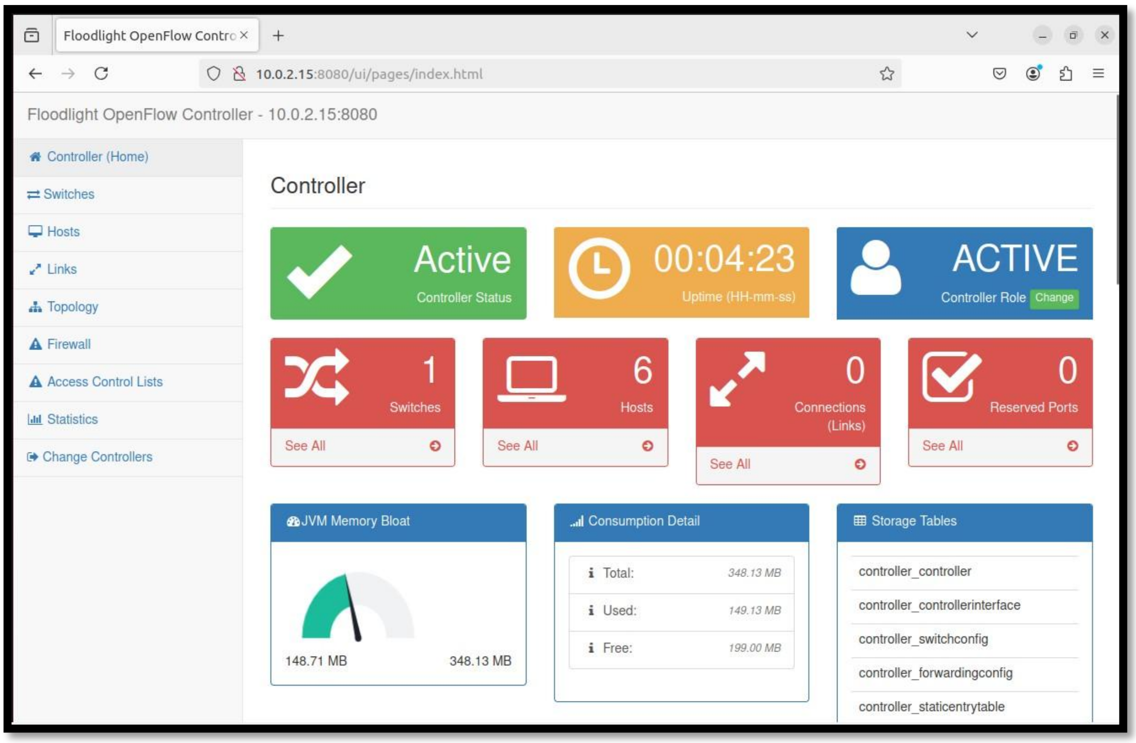
Nonetheless, it is not enough to simply launch the tool, due to the fact that it will not have visibility on the data flow, thus a bridge link must be formed in order to solve this issue.



```
vboxuser@Ubuntu:~$ sudo ovs-vsctl -- --id=@sflow create sflow agent=lo     target=\"10.0.2.15:6343\" sampling=64 polli
ng=10 -- -- set bridge s1 sflow=@sflow
a23502e0-8817-4c28-97fc-3e59cc729f0b
```

Figure C.4.1.4: Open Virtual Switch SDN | Setting bridge link s1 amidst the flow in lo interface & SFlow-RT open-source tool

If we open the FloodLight SDN controller, via heading to the link **http://10.0.2.15:8080/ui/pages/index.html** and checking the components of our network in the RESTful API, if everything went well, we should see 6 hosts and 1 switching device (OVS). The controller is active, and the devices are six as they should be.



Floodlight OpenFlow Controller - 10.0.2.15:8080

Controller

Active
Controller Status

00:04:23
Uptime (HH-mm-ss)

ACTIVE
Controller Role

1 Switches

6 Hosts

0 Connections (Links)

0 Reserved Ports

JVM Memory Bloat
148.71 MB 348.13 MB

Consumption Detail

Total:	348.13 MB
Used:	149.13 MB
Free:	199.00 MB

Storage Tables

controller_controller
controller_controllerinterface
controller_switchconfig
controller_forwardingconfig
controller_staticentrytable

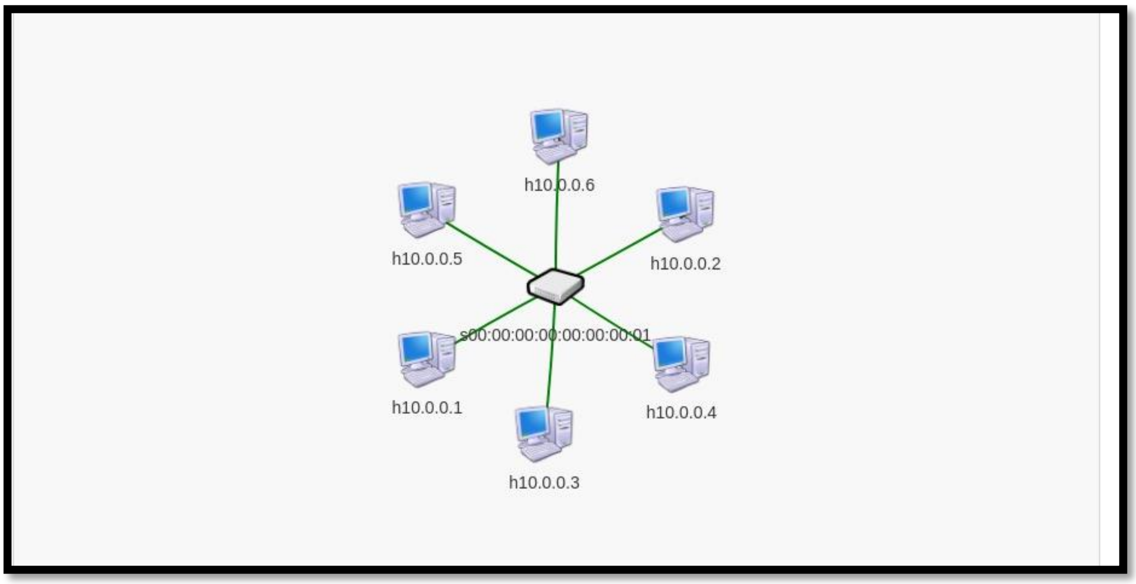
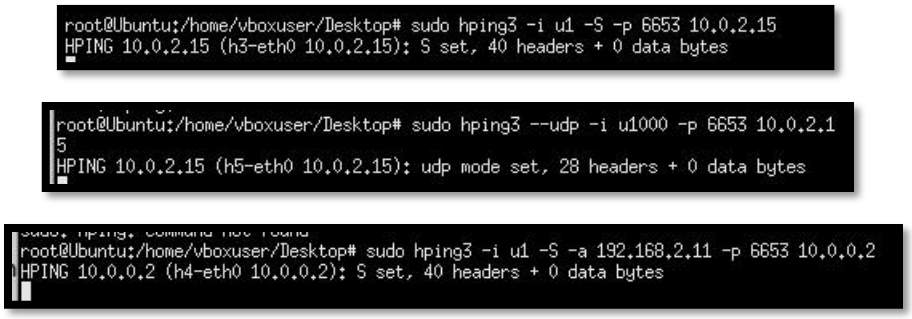


Figure C.4.1.5 & C.4.1.6: SDN Controller FloodLight Interface | Single Topology made up of six hosting devices with a sole Switching Device (OVS)

Under the assumption that the attacker has taken control of three of the six hosting devices in the SDN domain (Mininet) (h3, h4, h5) with IP addresses 10.0.0.3, 10.0.0.4, 10.0.0.5 having access in the tools xterm and hping3, the malicious actor could launch a new shell instance for each of the three hosting devices and execute Denial-Of-Service attack with the hping3. An open-source tool utilized mainly for packet fragmentation, sending custom data packets and even though it is usually used for testing purposes such as firewall rules, ports, and performance, it can also be utilized for malicious goals. Following is three shell instances launched for the three hosting devices respectively. The first two are launching a DoS attack simultaneously by trying to flood the data traffic in the FloodLight SDN Controller, while the third one will launch an IP flood DoS attack with spoofed data packets towards the second hosting device.



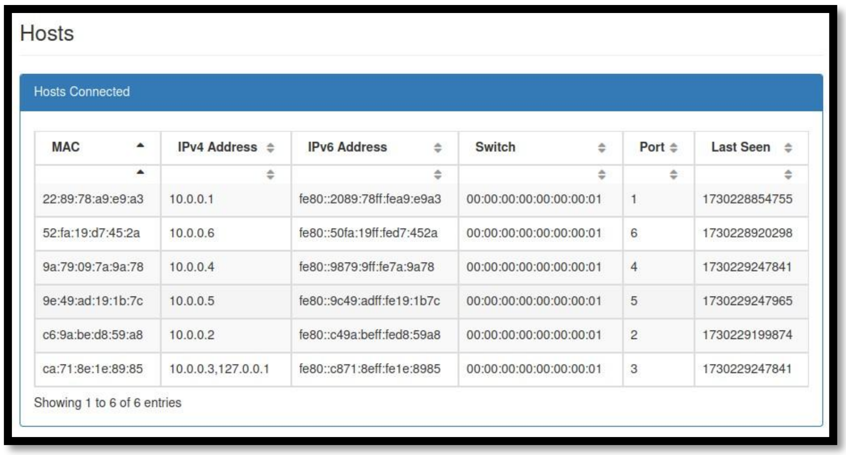
```
root@Ubuntu:/home/vboxuser/Desktop# sudo hping3 -i u1 -S -p 6653 10.0.2.15
HPING 10.0.2.15 (h3-eth0 10.0.2.15): S set, 40 headers + 0 data bytes
```

```
root@Ubuntu:/home/vboxuser/Desktop# sudo hping3 --udp -i u1000 -p 6653 10.0.2.1
5
HPING 10.0.2.15 (h5-eth0 10.0.2.15): udp mode set, 28 headers + 0 data bytes
```

```
root@Ubuntu:/home/vboxuser/Desktop# sudo hping3 -i u1 -S -a 192.168.2.11 -p 6653 10.0.0.2
HPING 10.0.0.2 (h4-eth0 10.0.0.2): S set, 40 headers + 0 data bytes
```

Figure C.4.1.7: Xterm launched instances for the compromised hosting devices h3, h4, h5 | UDP, TCP and IP Flood DoS Attack towards the SDN Controller & h2 (targeted hosting device)

The first command attempts to send multiple (-i u1) TCP SYN packets (-S) towards port 6653, because this is the default port of the OpenFlow Protocol, on which the FloodLight Controller runs with the targeted IP address localhost, while the second generates thousands (-i u1000) of UDP packets in an attempt to cause UDP flood. The final command creates the IP packets with spoofed IP source address (-a) with destination address the address of the h2 device with port 6653 since it is connected with the controller. The following are details on the six hosts of our topology.



Hosts

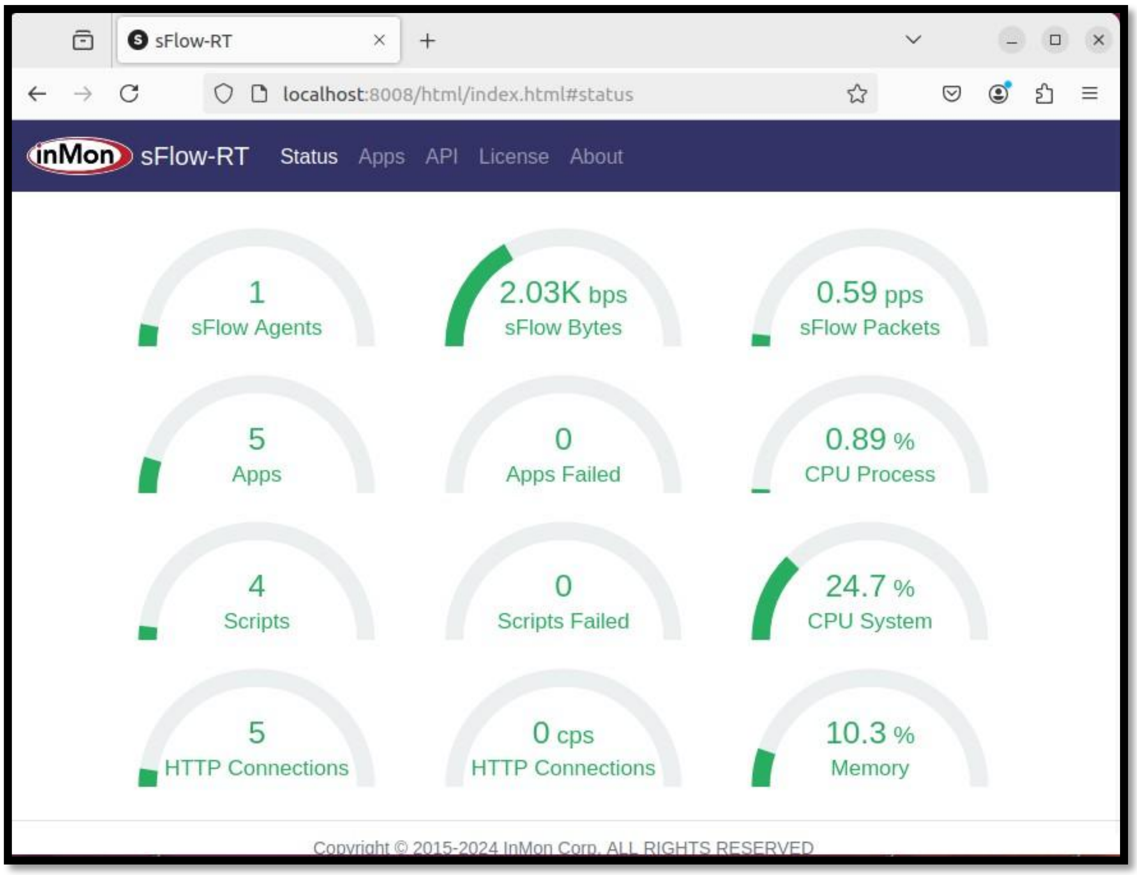
Hosts Connected

MAC	IPv4 Address	IPv6 Address	Switch	Port	Last Seen
22:89:78:a9:e9:a3	10.0.0.1	fe80::2089:78ff:fea9:e9a3	00:00:00:00:00:00:00:01	1	1730228854755
52:fa:19:d7:45:2a	10.0.0.6	fe80::50fa:19ff:fed7:452a	00:00:00:00:00:00:00:01	6	1730228920298
9a:79:09:7a:9a:78	10.0.0.4	fe80::9879:9ff:fe7a:9a78	00:00:00:00:00:00:00:01	4	1730229247841
9e:49:ad:19:1b:7c	10.0.0.5	fe80::9c49:adff:fe19:1b7c	00:00:00:00:00:00:00:01	5	1730229247965
c6:9a:be:d8:59:a8	10.0.0.2	fe80::c49a:beff:fed8:59a8	00:00:00:00:00:00:00:01	2	1730229199874
ca:71:8e:1e:89:85	10.0.0.3,127.0.0.1	fe80::c871:8eff:fe1e:8985	00:00:00:00:00:00:00:01	3	1730229247841

Showing 1 to 6 of 6 entries

Figure C.4.1.8: Table of hosting devices in the FloodLight SDN Controller

At the same time, the tools SFlow-RT and Wireshark packet sniffing tool are launched in order to detect the difference in the traffic flow and to check whether the tools are able to detect malfunctions and irregularities among the data flow. Following is the data flow before the DDoS attack is launched.



1 sFlow Agents

2.03K bps sFlow Bytes

0.59 pps sFlow Packets

5 Apps

0 Apps Failed

0.89 % CPU Process

4 Scripts

0 Scripts Failed

24.7 % CPU System

5 HTTP Connections

0 cps HTTP Connections

10.3 % Memory

Copyright © 2015-2024 InMon Corp. ALL RIGHTS RESERVED

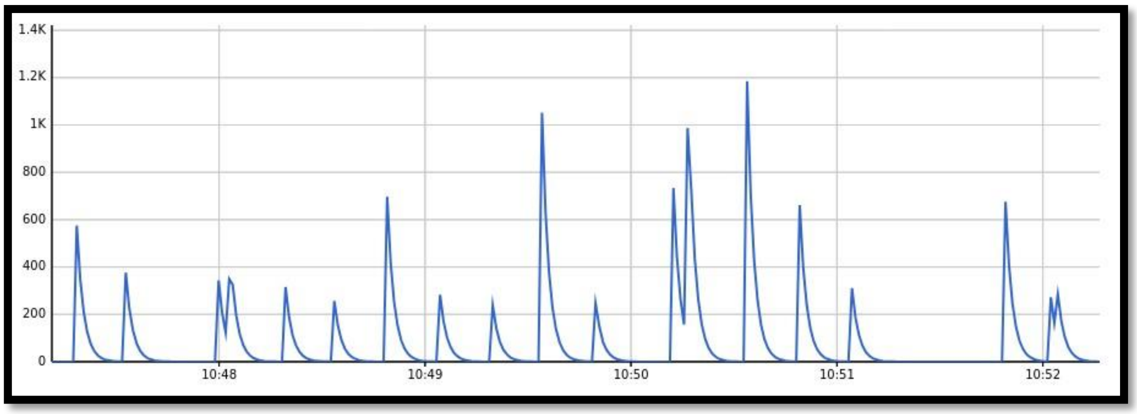


Figure C.4.1.9: SFlow-RT & Flow Table of Mininet – Before DDoS | FloodLight Environment

Nevertheless, it appears that after the DDoS attack takes place, the difference is quite evident in the SFlow-RT mn flow.

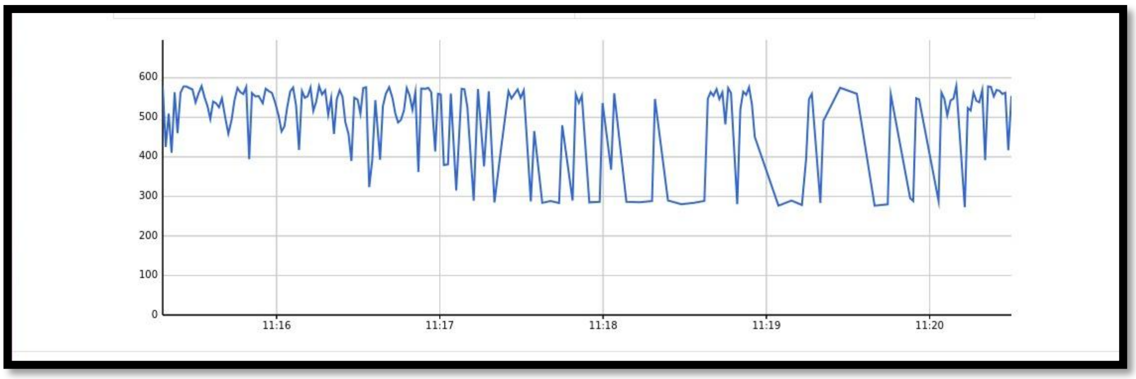
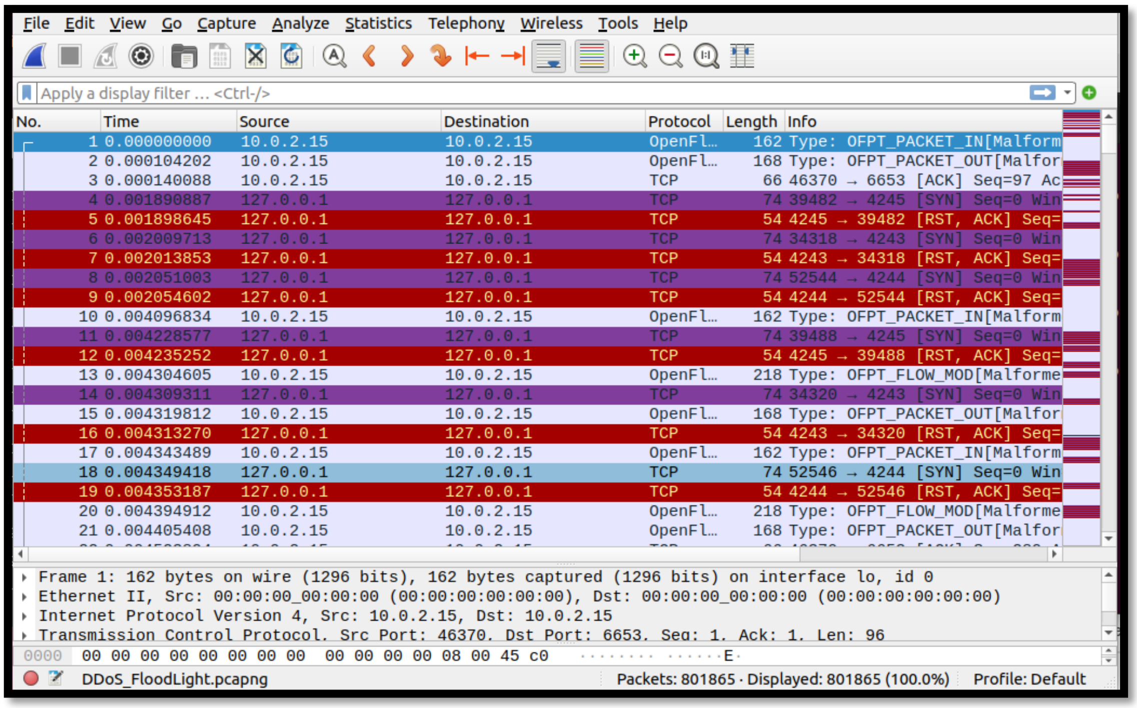


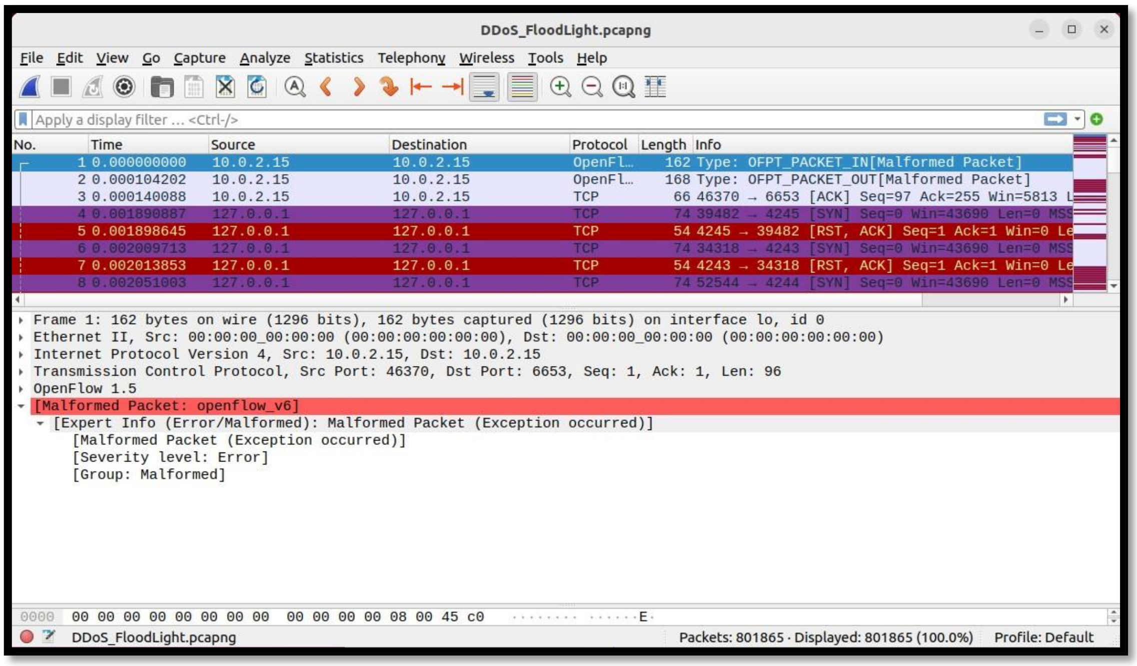
Figure C.4.1.10: Flow Table of Mininet – After DDoS | FloodLight Environment

However, even though the difference is clear, it is vital to include the Wireshark sniffing packet tool with the aim of collecting more details on the DDoS scenario and the devices, controller involved in the SDN environment.

The Wireshark tool is capable of performing filtering on packets and detailed analysis on the captured packet, passive and active monitoring.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time         Source      Destination  Protocol  Length Info
   1 0.000000000  10.0.2.15   10.0.2.15    OpenFl…   162 Type: OFPT_PACKET_IN[Malform
   2 0.000104202  10.0.2.15   10.0.2.15    OpenFl…   168 Type: OFPT_PACKET_OUT[Malfor
   3 0.000140088  10.0.2.15   10.0.2.15    TCP        66 46370 → 6653 [ACK] Seq=97 Ac
   4 0.001890887  127.0.0.1   127.0.0.1    TCP        74 39482 → 4245 [SYN] Seq=0 Win
   5 0.001898645  127.0.0.1   127.0.0.1    TCP        54 4245 → 39482 [RST, ACK] Seq=
   6 0.002009713  127.0.0.1   127.0.0.1    TCP        74 34318 → 4243 [SYN] Seq=0 Win
   7 0.002013853  127.0.0.1   127.0.0.1    TCP        54 4243 → 34318 [RST, ACK] Seq=
   8 0.002051003  127.0.0.1   127.0.0.1    TCP        74 52544 → 4244 [SYN] Seq=0 Win
   9 0.002054602  127.0.0.1   127.0.0.1    TCP        54 4244 → 52544 [RST, ACK] Seq=
  10 0.004096834  10.0.2.15   10.0.2.15    OpenFl…   162 Type: OFPT_PACKET_IN[Malform
  11 0.004228577  127.0.0.1   127.0.0.1    TCP        74 39488 → 4245 [SYN] Seq=0 Win
  12 0.004235252  127.0.0.1   127.0.0.1    TCP        54 4245 → 39488 [RST, ACK] Seq=
  13 0.004304605  10.0.2.15   10.0.2.15    OpenFl…   218 Type: OFPT_FLOW_MOD[Malforme
  14 0.004309311  127.0.0.1   127.0.0.1    TCP        74 34320 → 4243 [SYN] Seq=0 Win
  15 0.004319812  10.0.2.15   10.0.2.15    OpenFl…   168 Type: OFPT_PACKET_OUT[Malfor
  16 0.004313270  127.0.0.1   127.0.0.1    TCP        54 4243 → 34320 [RST, ACK] Seq=
  17 0.004343489  10.0.2.15   10.0.2.15    OpenFl…   162 Type: OFPT_PACKET_IN[Malform
  18 0.004349418  127.0.0.1   127.0.0.1    TCP        74 52546 → 4244 [SYN] Seq=0 Win
  19 0.004353187  127.0.0.1   127.0.0.1    TCP        54 4244 → 52546 [RST, ACK] Seq=
  20 0.004394912  10.0.2.15   10.0.2.15    OpenFl…   218 Type: OFPT_FLOW_MOD[Malforme
  21 0.004405408  10.0.2.15   10.0.2.15    OpenFl…   168 Type: OFPT_PACKET_OUT[Malfor

› Frame 1: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface lo, id 0
› Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
› Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15
› Transmission Control Protocol, Src Port: 46370, Dst Port: 6653, Seq: 1, Ack: 1, Len: 96
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 c0   ········ ······E·

DDoS_FloodLight.pcapng        Packets: 801865 · Displayed: 801865 (100.0%)    Profile: Default
```

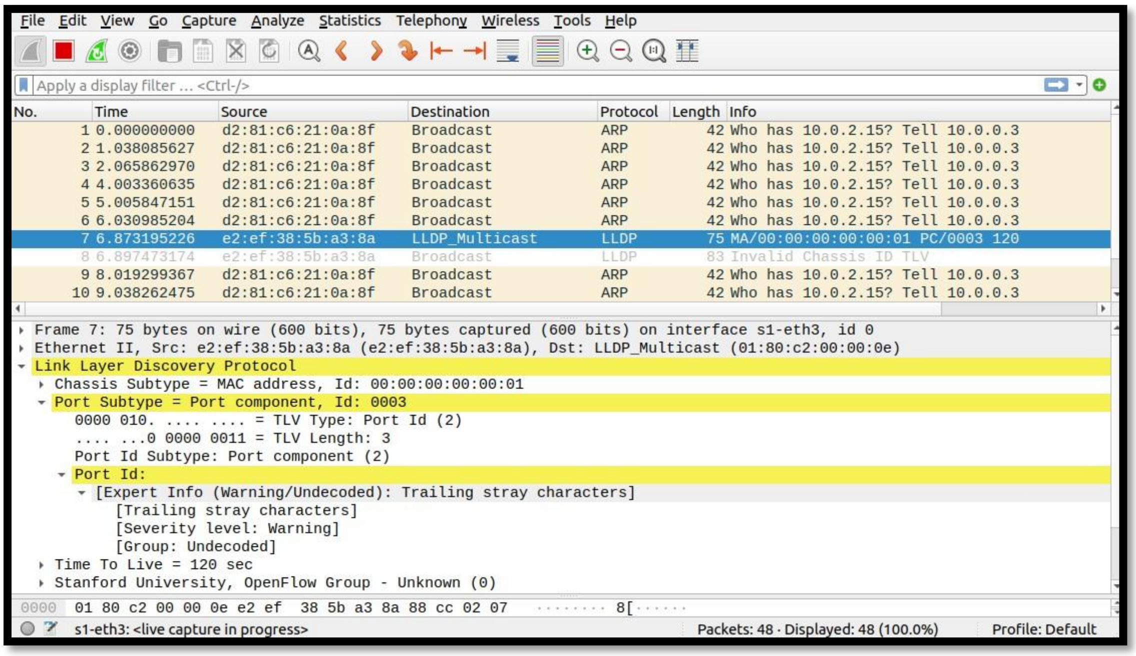


```
DDoS_FloodLight.pcapng

No.  Time           Source       Destination  Protocol  Length  Info
   1 0.000000000    10.0.2.15    10.0.2.15    OpenFl…      162  Type: OFPT_PACKET_IN[Malformed Packet]
   2 0.000104202    10.0.2.15    10.0.2.15    OpenFl…      168  Type: OFPT_PACKET_OUT[Malformed Packet]
   3 0.000140088    10.0.2.15    10.0.2.15    TCP           66  46370 → 6653 [ACK] Seq=97 Ack=255 Win=5813 L
   4 0.001890887    127.0.0.1    127.0.0.1    TCP           74  39482 → 4245 [SYN] Seq=0 Win=43690 Len=0 MSS
   5 0.001898645    127.0.0.1    127.0.0.1    TCP           54  4245 → 39482 [RST, ACK] Seq=1 Ack=1 Win=0 Le
   6 0.002009713    127.0.0.1    127.0.0.1    TCP           74  34318 → 4243 [SYN] Seq=0 Win=43690 Len=0 MSS
   7 0.002013853    127.0.0.1    127.0.0.1    TCP           54  4243 → 34318 [RST, ACK] Seq=1 Ack=1 Win=0 Le
   8 0.002051003    127.0.0.1    127.0.0.1    TCP           74  52544 → 4244 [SYN] Seq=0 Win=43690 Len=0 MSS

› Frame 1: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface lo, id 0
› Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
› Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15
› Transmission Control Protocol, Src Port: 46370, Dst Port: 6653, Seq: 1, Ack: 1, Len: 96
› OpenFlow 1.5
⌄ [Malformed Packet: openflow_v6]
  ⌄ [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
      [Malformed Packet (Exception occurred)]
      [Severity level: Error]
      [Group: Malformed]

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 c0   ········ ······E·

DDoS_FloodLight.pcapng          Packets: 801865 · Displayed: 801865 (100.0%)    Profile: Default
```

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

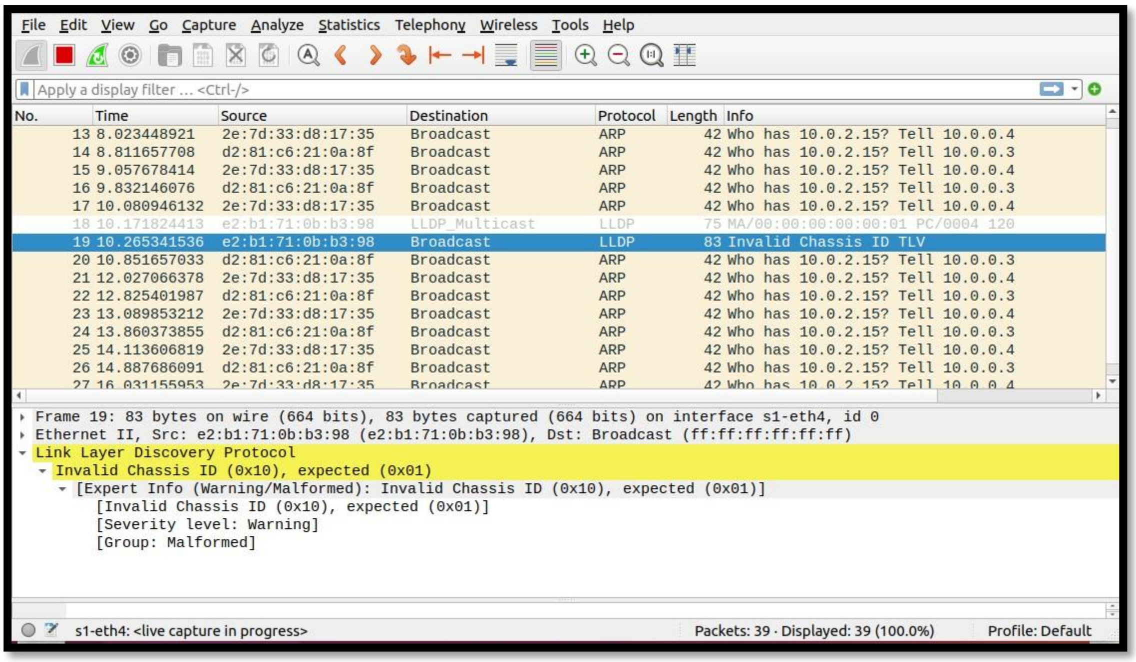
Figure C.4.1.11 & C.4.1.12: Wireshark – DDoS Attack – OpenFlow Malformed Packets – Central Interface – FloodLight Controller

Moreover, if we divert our attention to the interfaces of the compromised devices, we will see that it detected anomalies in the LLDP packets sent to our controller during DDoS attack.

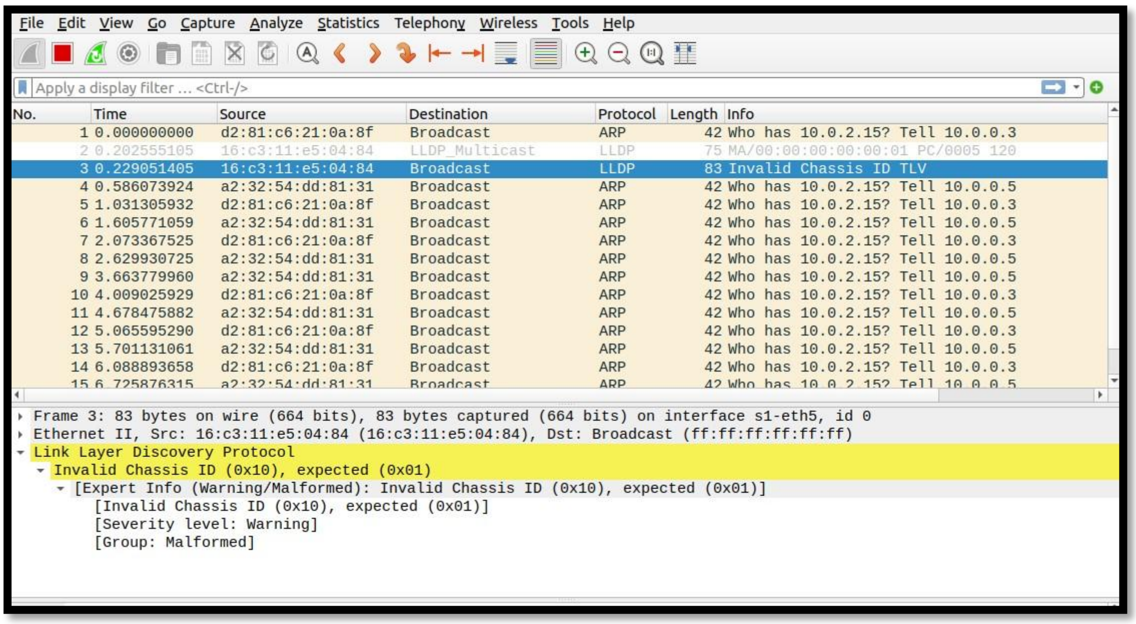


```
No.  Time           Source              Destination     Protocol  Length  Info
   1 0.000000000    d2:81:c6:21:0a:8f   Broadcast       ARP           42  Who has 10.0.2.15? Tell 10.0.0.3
   2 1.038085627    d2:81:c6:21:0a:8f   Broadcast       ARP           42  Who has 10.0.2.15? Tell 10.0.0.3
   3 2.065862970    d2:81:c6:21:0a:8f   Broadcast       ARP           42  Who has 10.0.2.15? Tell 10.0.0.3
   4 4.003360635    d2:81:c6:21:0a:8f   Broadcast       ARP           42  Who has 10.0.2.15? Tell 10.0.0.3
   5 5.005847151    d2:81:c6:21:0a:8f   Broadcast       ARP           42  Who has 10.0.2.15? Tell 10.0.0.3
   6 6.030985204    d2:81:c6:21:0a:8f   Broadcast       ARP           42  Who has 10.0.2.15? Tell 10.0.0.3
   7 6.873195226    e2:ef:38:5b:a3:8a   LLDP_Multicast  LLDP          75  MA/00:00:00:00:00:01 PC/0003 120
   8 6.897473174    e2:ef:38:5b:a3:8a   Broadcast       LLDP          83  Invalid Chassis ID TLV
   9 8.019299367    d2:81:c6:21:0a:8f   Broadcast       ARP           42  Who has 10.0.2.15? Tell 10.0.0.3
  10 9.038262475    d2:81:c6:21:0a:8f   Broadcast       ARP           42  Who has 10.0.2.15? Tell 10.0.0.3

› Frame 7: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface s1-eth3, id 0
› Ethernet II, Src: e2:ef:38:5b:a3:8a (e2:ef:38:5b:a3:8a), Dst: LLDP_Multicast (01:80:c2:00:00:0e)
⌄ Link Layer Discovery Protocol
  › Chassis Subtype = MAC address, Id: 00:00:00:00:00:01
  ⌄ Port Subtype = Port component, Id: 0003
      0000 010. .... .... = TLV Type: Port Id (2)
      .... ...0 0000 0011 = TLV Length: 3
      Port Id Subtype: Port component (2)
    ⌄ Port Id:
      ⌄ [Expert Info (Warning/Undecoded): Trailing stray characters]
          [Trailing stray characters]
          [Severity level: Warning]
          [Group: Undecoded]
  › Time To Live = 120 sec
  › Stanford University, OpenFlow Group - Unknown (0)

0000  01 80 c2 00 00 0e e2 ef  38 5b a3 8a 88 cc 02 07   ········ 8[······

s1-eth3: <live capture in progress>     Packets: 48 · Displayed: 48 (100.0%)     Profile: Default
```

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time          Source              Destination     Protocol  Length Info
   13 8.023448921   2e:7d:33:d8:17:35   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.4
   14 8.811657708   d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
   15 9.057678414   2e:7d:33:d8:17:35   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.4
   16 9.832146076   d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
   17 10.080946132  2e:7d:33:d8:17:35   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.4
   18 10.171824413  e2:b1:71:0b:b3:98   LLDP_Multicast  LLDP      75 MA/00:00:00:00:00:01 PC/0004 120
   19 10.265341536  e2:b1:71:0b:b3:98   Broadcast       LLDP      83 Invalid Chassis ID TLV
   20 10.851657033  d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
   21 12.027066378  2e:7d:33:d8:17:35   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.4
   22 12.825401987  d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
   23 13.089853212  2e:7d:33:d8:17:35   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.4
   24 13.860373855  d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
   25 14.113606819  2e:7d:33:d8:17:35   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.4
   26 14.887686091  d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
   27 16.031155953  2e:7d:33:d8:17:35   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.4

› Frame 19: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface s1-eth4, id 0
› Ethernet II, Src: e2:b1:71:0b:b3:98 (e2:b1:71:0b:b3:98), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▾ Link Layer Discovery Protocol
  ▾ Invalid Chassis ID (0x10), expected (0x01)
    ▾ [Expert Info (Warning/Malformed): Invalid Chassis ID (0x10), expected (0x01)]
        [Invalid Chassis ID (0x10), expected (0x01)]
        [Severity level: Warning]
        [Group: Malformed]

s1-eth4: <live capture in progress>          Packets: 39 · Displayed: 39 (100.0%)        Profile: Default
```

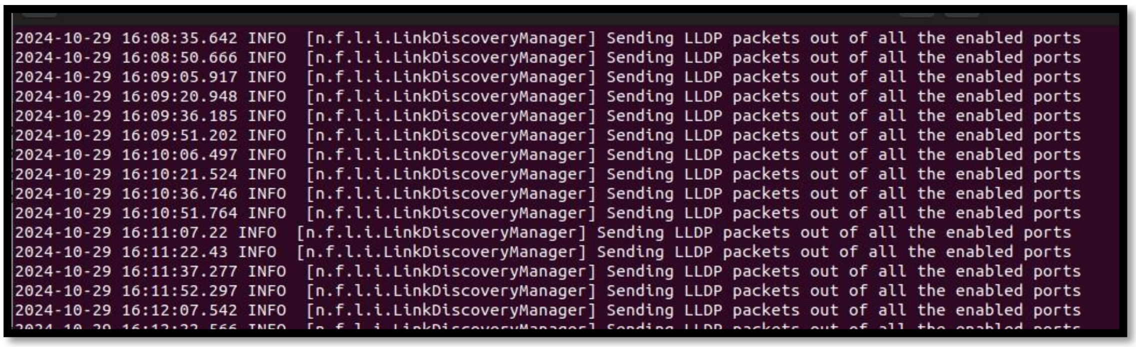

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time          Source              Destination     Protocol  Length Info
    1 0.000000000   d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
    2 0.202555105   16:c3:11:e5:04:84   LLDP_Multicast  LLDP      75 MA/00:00:00:00:00:01 PC/0005 120
    3 0.229051405   16:c3:11:e5:04:84   Broadcast       LLDP      83 Invalid Chassis ID TLV
    4 0.586073924   a2:32:54:dd:81:31   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.5
    5 1.031305932   d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
    6 1.605771059   a2:32:54:dd:81:31   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.5
    7 2.073367525   d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
    8 2.629930725   a2:32:54:dd:81:31   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.5
    9 3.663779960   a2:32:54:dd:81:31   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.5
   10 4.009025929   d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
   11 4.678475882   a2:32:54:dd:81:31   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.5
   12 5.065595290   d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
   13 5.701131061   a2:32:54:dd:81:31   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.5
   14 6.088893658   d2:81:c6:21:0a:8f   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.3
   15 6.725876315   a2:32:54:dd:81:31   Broadcast       ARP       42 Who has 10.0.2.15? Tell 10.0.0.5

› Frame 3: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface s1-eth5, id 0
› Ethernet II, Src: 16:c3:11:e5:04:84 (16:c3:11:e5:04:84), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▾ Link Layer Discovery Protocol
  ▾ Invalid Chassis ID (0x10), expected (0x01)
    ▾ [Expert Info (Warning/Malformed): Invalid Chassis ID (0x10), expected (0x01)]
        [Invalid Chassis ID (0x10), expected (0x01)]
        [Severity level: Warning]
        [Group: Malformed]
```

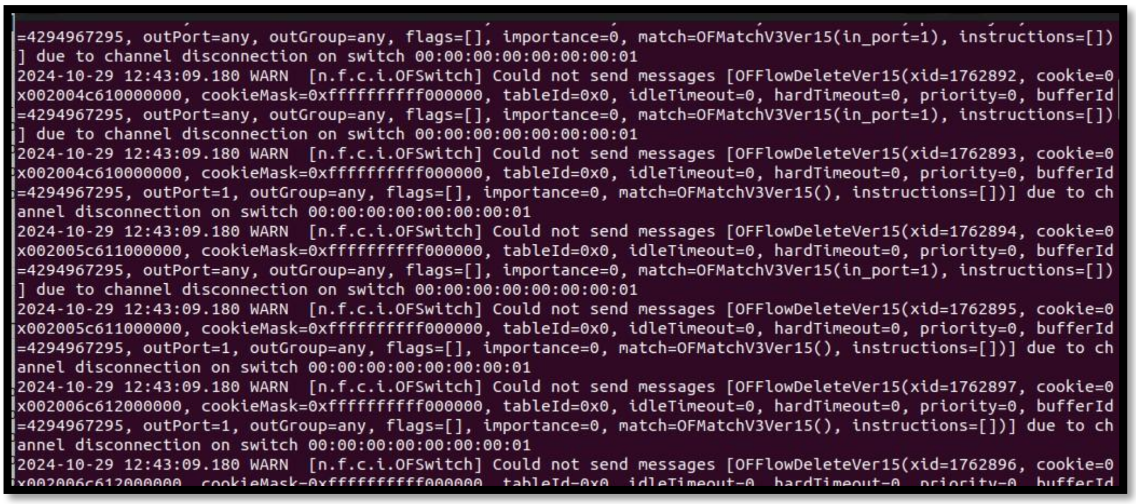
Figure C.4.1.13 - C.4.1.15: Wireshark – DDoS Attack – OpenFlow Malformed Packets – Origin – Compromised Devices

From the aforementioned illustrations, it seems that Wireshark has detected the flood DDoS attack with the OpenFlow protocol version 1.5 with security errors as malformed. The destination port is 6653 is used due to the fact that it is the default service port for the FloodLight SDN Controller.

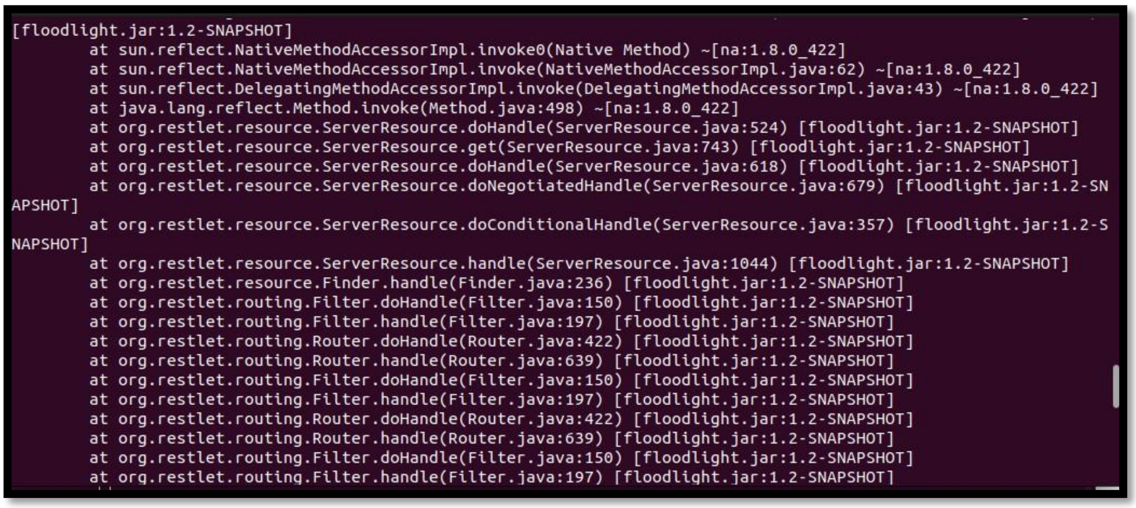
When the attempt was made to refresh the controller, its performance was dropped dramatically with it being slowed down. From the sheer amount of the packets sent, it caused the controller to show timeout error (server overload), as seen below:



```
2024-10-29 16:08:35.642 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:08:50.666 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:09:05.917 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:09:20.948 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:09:36.185 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:09:51.202 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:10:06.497 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:10:21.524 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:10:36.746 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:10:51.764 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:11:07.22 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:11:22.43 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:11:37.277 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:11:52.297 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2024-10-29 16:12:07.542 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
```



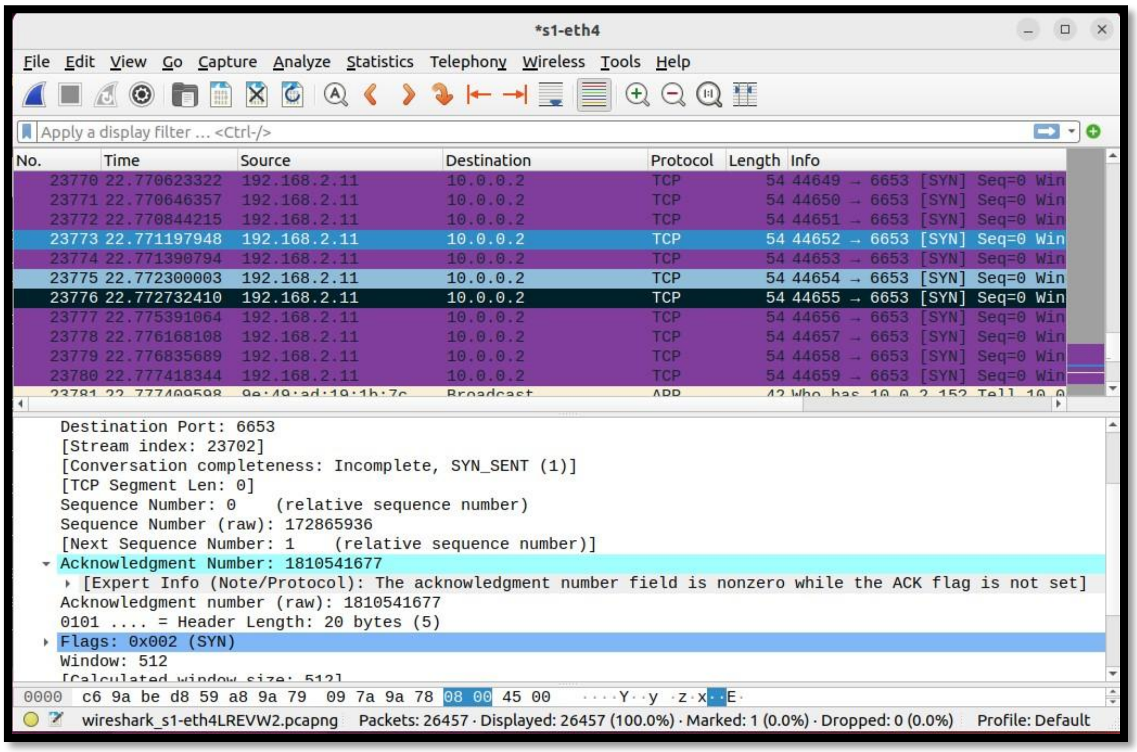
```
=4294967295, outPort=any, outGroup=any, flags=[], importance=0, match=OFMatchV3Ver15(in_port=1), instructions=[])
] due to channel disconnection on switch 00:00:00:00:00:00:00:01
2024-10-29 12:43:09.180 WARN  [n.f.c.i.OFSwitch] Could not send messages [OFFlowDeleteVer15(xid=1762892, cookie=0
x002004c610000000, cookieMask=0xffffffffff000000, tableId=0x0, idleTimeout=0, hardTimeout=0, priority=0, bufferId
=4294967295, outPort=any, outGroup=any, flags=[], importance=0, match=OFMatchV3Ver15(in_port=1), instructions=[])
] due to channel disconnection on switch 00:00:00:00:00:00:00:01
2024-10-29 12:43:09.180 WARN  [n.f.c.i.OFSwitch] Could not send messages [OFFlowDeleteVer15(xid=1762893, cookie=0
x002004c610000000, cookieMask=0xffffffffff000000, tableId=0x0, idleTimeout=0, hardTimeout=0, priority=0, bufferId
=4294967295, outPort=1, outGroup=any, flags=[], importance=0, match=OFMatchV3Ver15(), instructions=[])] due to ch
annel disconnection on switch 00:00:00:00:00:00:00:01
2024-10-29 12:43:09.180 WARN  [n.f.c.i.OFSwitch] Could not send messages [OFFlowDeleteVer15(xid=1762894, cookie=0
x002005c611000000, cookieMask=0xffffffffff000000, tableId=0x0, idleTimeout=0, hardTimeout=0, priority=0, bufferId
=4294967295, outPort=any, outGroup=any, flags=[], importance=0, match=OFMatchV3Ver15(in_port=1), instructions=[])
] due to channel disconnection on switch 00:00:00:00:00:00:00:01
2024-10-29 12:43:09.180 WARN  [n.f.c.i.OFSwitch] Could not send messages [OFFlowDeleteVer15(xid=1762895, cookie=0
x002005c611000000, cookieMask=0xffffffffff000000, tableId=0x0, idleTimeout=0, hardTimeout=0, priority=0, bufferId
=4294967295, outPort=1, outGroup=any, flags=[], importance=0, match=OFMatchV3Ver15(), instructions=[])] due to ch
annel disconnection on switch 00:00:00:00:00:00:00:01
2024-10-29 12:43:09.180 WARN  [n.f.c.i.OFSwitch] Could not send messages [OFFlowDeleteVer15(xid=1762897, cookie=0
x002006c612000000, cookieMask=0xffffffffff000000, tableId=0x0, idleTimeout=0, hardTimeout=0, priority=0, bufferId
=4294967295, outPort=1, outGroup=any, flags=[], importance=0, match=OFMatchV3Ver15(), instructions=[])] due to ch
annel disconnection on switch 00:00:00:00:00:00:00:01
2024-10-29 12:43:09.180 WARN  [n.f.c.i.OFSwitch] Could not send messages [OFFlowDeleteVer15(xid=1762896, cookie=0
```



```
[floodlight.jar:1.2-SNAPSHOT]
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) ~[na:1.8.0_422]
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62) ~[na:1.8.0_422]
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) ~[na:1.8.0_422]
        at java.lang.reflect.Method.invoke(Method.java:498) ~[na:1.8.0_422]
        at org.restlet.resource.ServerResource.doHandle(ServerResource.java:524) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.resource.ServerResource.get(ServerResource.java:743) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.resource.ServerResource.doHandle(ServerResource.java:618) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.resource.ServerResource.doNegotiatedHandle(ServerResource.java:679) [floodlight.jar:1.2-SN
APSHOT]
        at org.restlet.resource.ServerResource.doConditionalHandle(ServerResource.java:357) [floodlight.jar:1.2-S
NAPSHOT]
        at org.restlet.resource.ServerResource.handle(ServerResource.java:1044) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.resource.Finder.handle(Finder.java:236) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Filter.doHandle(Filter.java:150) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Filter.handle(Filter.java:197) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Router.doHandle(Router.java:422) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Router.handle(Router.java:639) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Filter.doHandle(Filter.java:150) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Filter.handle(Filter.java:197) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Router.doHandle(Router.java:422) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Router.handle(Router.java:639) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Filter.doHandle(Filter.java:150) [floodlight.jar:1.2-SNAPSHOT]
        at org.restlet.routing.Filter.handle(Filter.java:197) [floodlight.jar:1.2-SNAPSHOT]
```

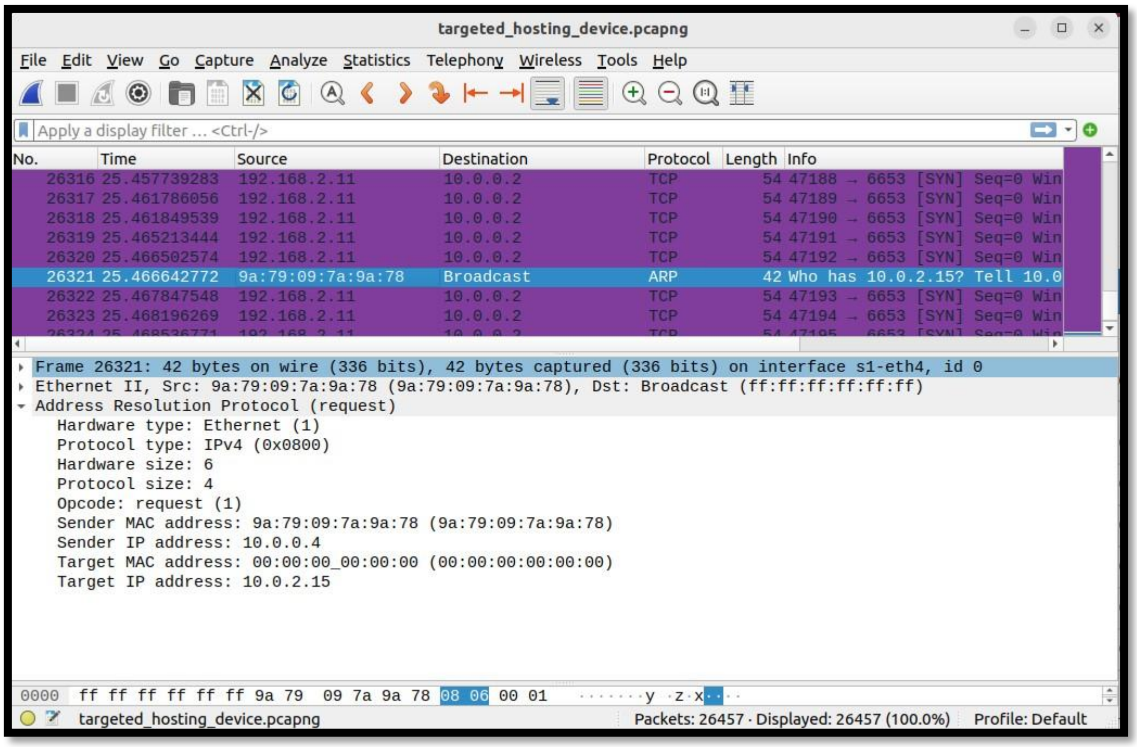
Figure C.4.1.16 – Figure C.4.1.18: FloodLight Controller – Error Logs

Additionally, Wireshark as a sniffing packet tool was capable of detecting the IP spoofed packets with the fraudulent source IP address “192.168.2.11” towards the destination IP address 10.0.0.2 of the hosting device h2 (Mininet).


*s1-eth4
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.   Time         Source        Destination  Protocol Length Info
23770 22.770623322 192.168.2.11  10.0.0.2     TCP      54 44649 → 6653 [SYN] Seq=0 Win
23771 22.770646357 192.168.2.11  10.0.0.2     TCP      54 44650 → 6653 [SYN] Seq=0 Win
23772 22.770844215 192.168.2.11  10.0.0.2     TCP      54 44651 → 6653 [SYN] Seq=0 Win
23773 22.771197948 192.168.2.11  10.0.0.2     TCP      54 44652 → 6653 [SYN] Seq=0 Win
23774 22.771390794 192.168.2.11  10.0.0.2     TCP      54 44653 → 6653 [SYN] Seq=0 Win
23775 22.772300003 192.168.2.11  10.0.0.2     TCP      54 44654 → 6653 [SYN] Seq=0 Win
23776 22.772732410 192.168.2.11  10.0.0.2     TCP      54 44655 → 6653 [SYN] Seq=0 Win
23777 22.775391064 192.168.2.11  10.0.0.2     TCP      54 44656 → 6653 [SYN] Seq=0 Win
23778 22.776168108 192.168.2.11  10.0.0.2     TCP      54 44657 → 6653 [SYN] Seq=0 Win
23779 22.776835689 192.168.2.11  10.0.0.2     TCP      54 44658 → 6653 [SYN] Seq=0 Win
23780 22.777418344 192.168.2.11  10.0.0.2     TCP      54 44659 → 6653 [SYN] Seq=0 Win
23781 22.777409598 9e:49:ad:19:1b:7c Broadcast ARP     42 Who has 10.0.2.15? Tell 10.0

    Destination Port: 6653
    [Stream index: 23702]
    [Conversation completeness: Incomplete, SYN_SENT (1)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 172865936
    [Next Sequence Number: 1    (relative sequence number)]
  ▾ Acknowledgment Number: 1810541677
    ▸ [Expert Info (Note/Protocol): The acknowledgment number field is nonzero while the ACK flag is not set]
    Acknowledgment number (raw): 1810541677
    0101 .... = Header Length: 20 bytes (5)
  ▸ Flags: 0x002 (SYN)
    Window: 512
    [Calculated window size: 512]

0000  c6 9a be d8 59 a8 9a 79  09 7a 9a 78 08 00 45 00   ····Y··y ·z·x··E·
```

wireshark_s1-eth4LREVW2.pcapng
Packets: 26457 · Displayed: 26457 (100.0%) · Marked: 1 (0.0%) · Dropped: 0 (0.0%)
Profile: Default


targeted_hosting_device.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.   Time         Source            Destination Protocol Length Info
26316 25.457739283 192.168.2.11      10.0.0.2    TCP      54 47188 → 6653 [SYN] Seq=0 Win
26317 25.461786056 192.168.2.11      10.0.0.2    TCP      54 47189 → 6653 [SYN] Seq=0 Win
26318 25.461849539 192.168.2.11      10.0.0.2    TCP      54 47190 → 6653 [SYN] Seq=0 Win
26319 25.465213444 192.168.2.11      10.0.0.2    TCP      54 47191 → 6653 [SYN] Seq=0 Win
26320 25.466502574 192.168.2.11      10.0.0.2    TCP      54 47192 → 6653 [SYN] Seq=0 Win
26321 25.466642772 9a:79:09:7a:9a:78 Broadcast   ARP      42 Who has 10.0.2.15? Tell 10.0
26322 25.467847548 192.168.2.11      10.0.0.2    TCP      54 47193 → 6653 [SYN] Seq=0 Win
26323 25.468196269 192.168.2.11      10.0.0.2    TCP      54 47194 → 6653 [SYN] Seq=0 Win
26324 25.468536771 192.168.2.11      10.0.0.2    TCP      54 47195 → 6653 [SYN] Seq=0 Win

▸ Frame 26321: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface s1-eth4, id 0
▸ Ethernet II, Src: 9a:79:09:7a:9a:78 (9a:79:09:7a:9a:78), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 9a:79:09:7a:9a:78 (9a:79:09:7a:9a:78)
    Sender IP address: 10.0.0.4
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.2.15

0000  ff ff ff ff ff ff 9a 79  09 7a 9a 78 08 06 00 01   ·······y ·z·x····
```

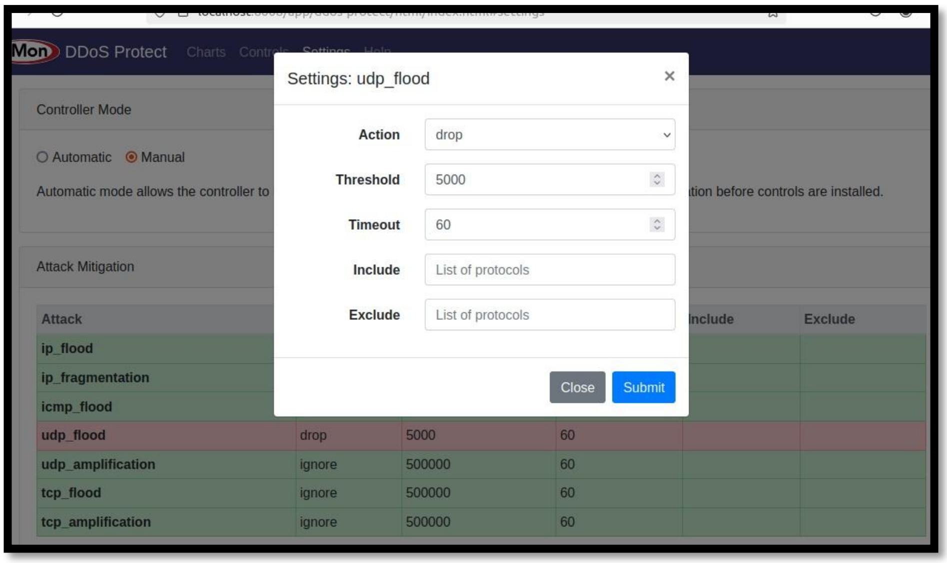
targeted_hosting_device.pcapng
Packets: 26457 · Displayed: 26457 (100.0%)
Profile: Default

Figure C.4.1.19 – Figure C.4.1.20: SFlow-RT InMon Packets ARP – IP Spoofed Hping3 – FloodLight SDN – Mininet

Wireshark detected the spoofed packets and displayed the message "acknowledgment number field is nonzero while the ACK flag is not set", which is a sign of a potential attack on a transmission protocol level. Due to the fact that the acknowledgement number is utilized in order to ensure whether the data were retrieved successfully or not, through the indication of the following sequence number. The attacker could have intentionally set the number to a non-zero and not set a flag on the grounds that intrusion detection and prevention systems might not have filters for logical correlation amidst the acknowledgment flag and the acknowledgment number, rendering them vulnerable to malicious data packets that could bypass the security as malformed and ignore them. Consequently, the actor will have successfully evaded the packet filtering. In order to truly prevent and mitigate such attacks, it is of vital importance to include Web Application Firewalls, limit the packet processing rate, and apply IPS/IDS systems with proper filtering rules.

Another solution for the aforementioned issue would be the implementation of the DDoS protection application that SFlow-RT entails, which is capable of limiting the rate of the incoming packets and identifying DDoS-related incidents.

From the DDoS-Protect, we have filters table on the IP addresses that need to be excluded from the access lists regarding the addresses that are allowed to communicate with the FloodLight Controller.



Controller Mode

Automatic [✓ Manual]

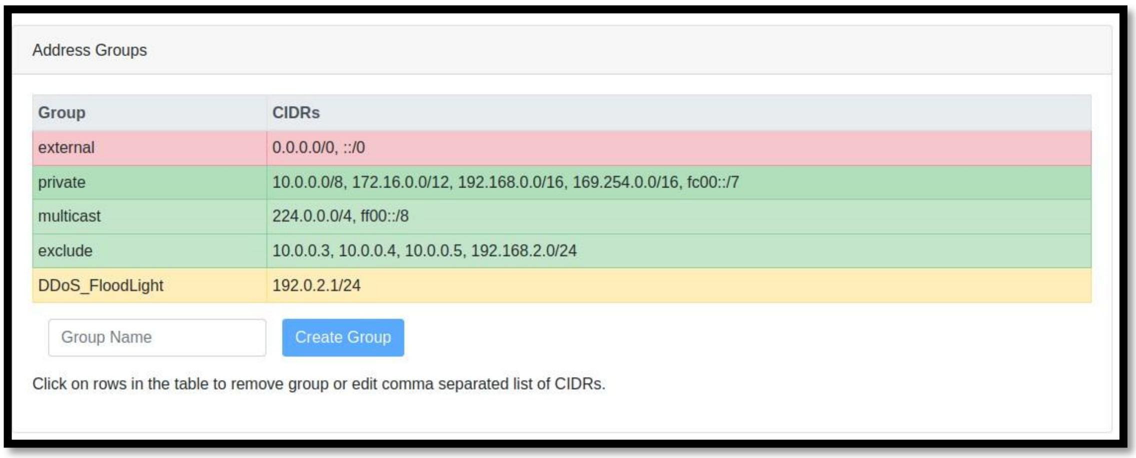
Automatic mode allows the controller to ... tion before controls are installed.

Settings: udp_flood

Action	drop
Threshold	5000
Timeout	60
Include	List of protocols
Exclude	List of protocols

Attack Mitigation

Attack				Include	Exclude
ip_flood					
ip_fragmentation					
icmp_flood					
udp_flood	drop	5000	60		
udp_amplification	ignore	500000	60		
tcp_flood	ignore	500000	60		
tcp_amplification	ignore	500000	60		



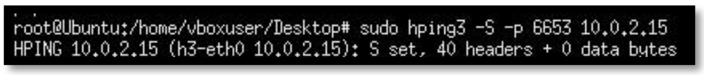
Address Groups

Group	CIDRs
external	0.0.0.0/0, ::/0
private	10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 169.254.0.0/16, fc00::/7
multicast	224.0.0.0/4, ff00::/8
exclude	10.0.0.3, 10.0.0.4, 10.0.0.5, 192.168.2.0/24
DDoS_FloodLight	192.0.2.1/24

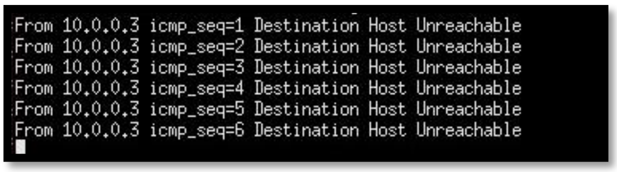
Click on rows in the table to remove group or edit comma separated list of CIDRs.

Figure C.4.1.21 – Figure C.4.1.22: SFlow-RT DDoS Protect – Access Control Groups

If the attacker attempts to perform DDoS from any of the compromised hosting devices (h3, h4, h5) then the DDoS-Protect will cut off any packets that come from these devices, as demonstrated below.



```
root@Ubuntu:/home/vboxuser/Desktop# sudo hping3 -S -p 6653 10.0.2.15
HPING 10.0.2.15 (h3-eth0 10.0.2.15): S set, 40 headers + 0 data bytes
```



```
From 10.0.0.3 icmp_seq=1 Destination Host Unreachable
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable
From 10.0.0.3 icmp_seq=3 Destination Host Unreachable
From 10.0.0.3 icmp_seq=4 Destination Host Unreachable
From 10.0.0.3 icmp_seq=5 Destination Host Unreachable
From 10.0.0.3 icmp_seq=6 Destination Host Unreachable
```

Therefore, we successfully thwarted the attempt to flood our SDN environment. However, these measures alone will not suffice, and this is why security policies, rules and automated configuration handling methods are also needed to mitigate such events.

Conclusion

From the entirety of the thesis, we have been thoroughly discussing and examining the architecture as well as the inner workings of the Software-Defined Networks. The new trends that surround it, their benefits and challenges have been analyzed not only in comparison to their predecessor but also when it concerns the relationships between the three layers and their major components. Afterward, the Software-Defined Networks were examined from the aspect of security. Meticulously reviewing its main susceptibilities and the countermeasures that can be taken, while providing the prominent security traits of the Software-Defined Networks that set it apart from the rest of the other network technologies. The three layers, mainly infrastructure, control and application layer were analyzed not only from a network but also from a security perspective with preventive and mitigation measures. The SDN concept even though it is relatively new to the fields of both networking and security, many have been researching it giving birth to technologies and methods, causing its rapid development and proving its worth as a solution for the problems that were caused by the traditional networks. All in all, on the grounds that the SDN still has room for improvement and since it has various applications in cloud computing and network virtualization, combined with its vast potential for the enhancement of network security, it will be an asset of vital importance, that will bring about a great change to many fields and due to the aforesaid fact, it is expected to draw tremendous amounts of attention.

Acknowledgements

Primarily, I would like to show appreciation to my advisor professor George Xylomenos, for his invaluable guidance, insightful feedback as well as his unfaltering support throughout the duration of my master’s thesis. His expertise and constructive criticism have motivated me to broaden my horizons and sharpen my critical thinking regarding my approach to understanding the network concept, Software-Defined Network. I am very fortunate to have had this opportunity, mainly to learn under his guidance.

Moreover, I would like to extend my sincere thanks to my colleagues, friends and family for their advice in every part of my research. Their encouragement and advice helped me to reach the answers that eluded me during the process of understanding the Software-Defined Networks.

References – Bibliography

- ♣ [1]. Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M. and Gu, G. (2012). A security enforcement kernel for OpenFlow networks. *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*. [online] doi:https://doi.org/10.1145/2342441.2342466.

- ♣ [2]. Zhou, Y., Li, H., Chen, K., Pan, T., Qian, K., Zheng, K., Liu, B., Zhang, P., Tang, Y. and Hu, C. (2021). Raze policy conflicts in SDN. *Journal of Network and Computer Applications*, [online] 199, pp.103307–103307. doi:https://doi.org/10.1016/j.jnca.2021.103307.

- ♣ [3]. Xia, W., Wen, Y., Foh, C.H., Niyato, D. and Xie, H. (2015). A Survey on Software-Defined Networking. *IEEE Communications Surveys & Tutorials*, [online] 17(1), pp.27–51. doi:https://doi.org/10.1109/comst.2014.2330903.

- ♣ [4]. ResearchGate. (n.d.). *Figure 1.Traditional Network versus SDN*. [online] Available at: https://www.researchgate.net/figure/Traditional-Network-versus-SDN_fig1_319876305.

- ♣ [5]. Sultan Almakdi, Aqsa Aqdus, Amin, R. and Alshehri, M.S. (2023). An Intelligent Load Balancing Technique for Software Defined Networking based 5G using Machine Learning models. *IEEE Access*, 11, pp.105082–105104. doi:https://doi.org/10.1109/access.2023.3317513.

- ♣ [6]. Xia, W., Wen, Y., Foh, C.H., Niyato, D. and Xie, H. (2015). A Survey on Software-Defined Networking. *IEEE Communications Surveys & Tutorials*, [online] 17(1), pp.27–51. doi: https://doi.org/10.1109/comst.2014.2330903.

- ♣ [7]. Exemplary Illustration of the Open Virtual Switch Architecture | P4.org (2016). *P4 and Open vSwitch - Open Networking Foundation*. [online] Open Networking Foundation. Available at: https://opennetworking.org/news-and-events/blog/p4-and-open-vswitch/.

- ♣ [8]. Jha, R.K. and Llah, B.N.M. (2019). Software Defined Optical Networks (SDON): proposed architecture and comparative analysis. *Journal of the European Optical Society-Rapid Publications*, 15(1). doi:https://doi.org/10.1186/s41476-019-0105-4.

- ♣ [9]. Sangodoyin, A., Sigwele, T., Pillai, P., Hu, Y.F., Awan, I. and Disso, J. (2018). DoS Attack Impact Assessment on Software Defined Networks. *Wireless and Satellite Systems*, pp.11–22. doi:https://doi.org/10.1007/978-3-319-76571-6_2.

- [10]. Ehab Al-Shaer and Saeed Al-Haj (2010). *FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures*. [online] ResearchGate.

- [11].https://www.researchgate.net/publication/247928775_FlowChecker_Configuration_analysis_and_verification_of_federated_OpenFlow_infrastructures

- [12]. Khurshid, A., Zou, X., Zhou, W., Caesar, M. and Godfrey, P. (n.d.). *USENIX Association 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13) 15 VeriFlow: Verifying Network-Wide Invariants in Real Time*. [online]
  Available at: https://www.usenix.org/system/files/conference/nsdi13/nsdi13-final100.pdf.

- [13]. Zhu, L., Karim, M.M., Sharif, K., Li, F., Du, X. and Guizani, M. (2019). SDN Controllers: Benchmarking & Performance Evaluation. *arXiv:1902.04491 [cs]*. [online] Available at: https://arxiv.org/abs/1902.04491.

- [14]. Ranjit Jhala and Schmidt, D. (2011). *Verification, Model Checking, and Abstract Interpretation*. Springer Science & Business Media.

- [15]. GeeksforGeeks (n.d.). *GeeksforGeeks | A computer science portal for geeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/.

- [16]. Open Networking Foundation. (n.d.). *Open Networking Foundation*. [online] Available at: https://opennetworking.org/.

- [17]. Cisco (2023). *Cisco - Global Home Page*. [online] Cisco. Available at: https://www.cisco.com/.

- [18]. VMware (2019). *VMware – Cloud, Mobility, Networking & Security Solutions*. [online] VMware. Available at: https://www.vmware.com/.

- [19]. Cloudflare (2019). *Cloudflare*. [online] Cloudflare. Available at: https://www.cloudflare.com/.

- [20]. Oracle.com. (2024). *Oracle Virtual Cloud Network*. [online] Available at: https://www.oracle.com/uk/cloud/networking/virtual-cloud-network/.

- ♣ [21]. Handigol, N., Heller, B., Jeyakumar, V., Mazières, D. and Mckeown, N. (n.d.). *Where is the Debugger for my Software-Defined Network?* [online] Available at:https://www.scs.stanford.edu/~dm/home/papers/handigol:ndb-hotsdn.pdf.

- ♣ [22]. Haas, Z.J., Culver, T.L. and Sarac, K. (2021). Vulnerability Challenges of Software Defined Networking. *IEEE Communications Magazine*, 59(7), pp.88–93. doi:https://doi.org/10.1109/mcom.001.2100128.

- ♣ [23]. Shu, Z., Wan, J., Li, D., Lin, J., Vasilakos, A.V. and Imran, M. (2016). Security in Software-Defined Networking: Threats and Countermeasures. *Mobile Networks and Applications*, 21(5), pp.764–776. doi:https://doi.org/10.1007/s11036-016-0676-x.

- ♣ [24]. Yang M, Li Y, Jin D, Zeng L, Wu X, Vasilakos A (2015) Software-defined and virtualized future mobile and wireless networks: a survey. ACM/Springer Mob Netw Appl 20(1):4–18

- ♣ [25]. Dierks T (2008) The transport layer security (TLS) protocol version 1.2 [Online]. Available: http://tools.ietf.org/html/rfc5246

- ♣ [26]. Scott-Hayward S, O'Callaghan G, Sezer S (2013) Sdn security: a survey. In: IEEE SDN Future Networks and Services (SDN4FNS), pp 1–7

- ♣ [27]. ryu-sdn.org. (n.d.). *Ryu SDN Framework*. [online] Available at: https://ryu-sdn.org/.

- ♣ [28]. GitHub. (2021). *faucetsdn/ryu*. [online] Available at: https://github.com/faucetsdn/ryu.

- ♣ [29]. mininet.org. (n.d.). *Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet*. [online] Available at: https://mininet.org/.

- ♣ [30]. Heller B, Sherwood R, McKeown N (2012) The controller placement problem. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ACM, pp 7–12

♣ [31]. Alaa Taima Albu-Salih (2022). Performance Evaluation of Ryu Controller in Software Defined Networks. *Magallaẗ al-qādisiyyaẗ li-ʿulūm al-ḥāsibāt wa-al-riyāḍiyyāt,* 14(1). doi:https://doi.org/10.29304/jqcm.2022.14.1.879.

♣ [32]. Shin S, Porras P, Yegneswaran V, Fong M, Gu G, Tyson M (2013) FRESCO: Modular Composable Security Services for Software-Defined Networks. In: Proceedings of Network and Distributed Security Symposium, pp 1-16

♣ [33]. Amin, R., Hamza Aldabbas and Ahmed, N. (2024). Intrusion detection systems for software-defined networks: a comprehensive study on machine learning-based techniques. *Cluster Computing*. [online] doi:https://doi.org/10.1007/s10586-024-04430-6.

♣ [34]. Eliyan, L.F. and Di Pietro, R. (2021). DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges. *Future Generation Computer Systems,* 122. doi:https://doi.org/10.1016/j.future.2021.03.011.

♣ [35]. Conti, M., Gangwal, A. and Gaur, M.S. (2017). *A comprehensive and effective mechanism for DDoS detection in SDN*. [online] IEEE Xplore. doi:https://doi.org/10.1109/WiMOB.2017.8115796.

♣ [36]. Sebbar, A., Zkik, K., Boulmalf, M. and El Kettani, M.D.E.-C. (2019). New context-based node acceptance CBNA framework for MitM detection in SDN Architecture. *Procedia Computer Science*, 160, pp.825–830. doi:https://doi.org/10.1016/j.procs.2019.11.004.

♣ [37]. Open Networking Foundation. (2017). *Open Networking Technical Communities, Corporate Memberships*. [online] Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical.

♣ [38]. AbdelSalam, A.M., El-Sisi, A.B. and Reddy K, V. (2015). *Mitigating ARP Spoofing Attacks in Software-Defined Networks*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICCTA37466.2015.9513433.

- ♣ [39]. Iqbal, M., Iqbal, F., Mohsin, F., Rizwan, M. and Fahad Ahamd (2019). *Security Issues in Software Defined Networking (SDN): Risks, Challenges and Potential Solutions*. [online] Available at: https://www.researchgate.net/publication/338019274_Security_Issues_in_Software_Defined_Networking_SDN_Risks_Challenges_and_Potential_Solutions

- ♣ [40]. Lawal, B.H. and At, N. (2018). Improving Software Defined Network Security via sFLow and IPSec Protocol. *Anadolu University Journal of Science and* Technology-A Applied Sciences and Engineering, pp.1–2. doi:https://doi.org/10.18038/aubtda.421939.

- ♣ [41]. Wang, H., Xu, L. and Gu, G. (2015). *FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks*. [online] IEEE Xplore. doi:https://doi.org/10.1109/DSN.2015.27.

- ♣ [42]. Suh, J., Choi, H.-G., Yoon, W., You, T., Taekyoung, T., Kwon, quot; and Choi, Y. (n.d.). *Implementation of Content-oriented Networking Architecture (CONA): A Focus on DDoS Countermeasure*. [online] Available at: https://www.cl.cam.ac.uk/research/srg/netos/projects/netfpga/workshop/eurodev2010/suh/suh.pdf.

- ♣ [43]. Tootoonchian, A. and Ganjali, Y. (n.d.). *HyperFlow: A Distributed Control Plane for OpenFlow*. [online] Available at: https://www.usenix.org/legacy/event/inmwren10/tech/full_papers/Tootoonchian.pdf.

- ♣ [44]. Hock, D., Hartmann, M., Gebert, S., Jarschel, M., Zinner, T. and Phuoc Tran-Gia (2013). Pareto-optimal resilient controller placement in SDN-based core networks. *International Teletraffic Congress*. doi:https://doi.org/10.1109/itc.2013.6662939.

- ♣ [45]. Son, S., Shin, S., Vinod Yegneswaran, Porras, P. and Gu, G. (2013). Model checking invariant security properties in OpenFlow. doi:https://doi.org/10.1109/icc.2013.6654813.

♣ [46]. Mai, H., Khurshid, A., Agarwal, R., Caesar, M., Godfrey, P.B. and King, S.T. (2011). Debugging the data plane with anteater. *ACM SIGCOMM Computer Communication Review,* [online] 41(4), pp.290–301. doi:https://doi.org/10.1145/2043164.2018470.

♣ [47]. Ma, Y.-W., Chen, J.-L., Tsai, Y.-H., Cheng, K.-H. and Hung, W.-C. (2016). Load-Balancing Multiple Controllers Mechanism for Software-Defined Networking. *Wireless Personal Communications*, 94(4), pp.3549–3574. doi:https://doi.org/10.1007/s11277-016-3790-y.

♣ [48]. Skowyra, R., Lapets, A., Bestavros, A. and Kfoury, A. (n.d.). *Verifiably-Safe Software-Defined Networks for CPS*. [online] Available at: https://www.cs.bu.edu/fac/best/res/papers/hicons13.pdf.

♣ [49]. Ball, T., Bjørner, N., Gember, A., Itzhaky, S., Karbyshev, A., Sagiv, M., Schapira, M. and Valadarsky, A. (2014). VeriCon. *ACM SIGPLAN Notices*, 49(6), pp.282–293. doi:https://doi.org/10.1145/2666356.2594317.

♣ [50]. Canini, M., Venzano, D., Perešíni, P., Kostić, D., Rexford, J. and Epfl (n.d.). *A NICE Way to Test OpenFlow Applications*. [online] Available at: https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final105.pdf

♣ [51]. Nadjib Achir, Mauro, Ghamri, Y.M., Doudane Nazim and Agoulmine Ahmed Mehaoua (2001). *Active Networking System Evaluation: A Practical Experience*. [online] Available at: https://www.researchgate.net/publication/2400076_Active_Networking_System_Evaluation_A_Practical_Experience.

♣ [52]. Yeganeh, S. and Ganjali, Y. (2012). *Kandoo: a framework for efficient and scalable offloading of control applications*. [online] Semantic Scholar. doi:https://doi.org/10.1145/2342441.2342446.

♣ [53]. Perešíni, P. and Canini, M. (2011). Is your OpenFlow application correct? *Infoscience (Ecole Polytechnique Fédérale de Lausanne)*, pp.1–2. doi:https://doi.org/10.1145/2079327.2079345.

♣ [54]. Handigol, N., Heller, B., Jeyakumar, V., Mazières, D. and Mckeown, N. (n.d.). *Where is the Debugger for my Software-Defined Network?* [online] Available at: https://www.scs.stanford.edu/~dm/home/papers/handigol:ndb-hotsdn.pdf.

♣ [55]. Braga, R., Mota, E. and Passito, A. (2010). Lightweight DDoS flooding attack detection using NOX/OpenFlow. *IEEE Local Computer Network Conference*. doi:https://doi.org/10.1109/lcn.2010.5735752.

- ♣ [56]. Scirp.org. (2015). *Rigney, C., Willens, S., Rubens, A. and Simpson, W. (2000) Remote Authentication Dial in User Service (RADIUS). RFC 2865. - References - Scientific Research Publishing*. [online] Available at: https://www.scirp.org/reference/referencespapers?referenceid=1631785.

- ♣ [57]. Handigol, N., Seetharaman, S., Flajslik, M., Mckeown, N. and Johari, R. (n.d.). *Plug-n-Serve: Load-Balancing Web Traffic using OpenFlow*. [online] Available at: https://conferences.sigcomm.org/sigcomm/2009/demos/sigcomm-pd-2009-final26.pdf.

- ♣ [58]. Yehuda Afek, Anat Bremler-Barr and Shafir, L. (2017). Network anti-spoofing with SDN data plane. *International Conference on Computer Communications*. doi:https://doi.org/10.1109/infocom.2017.8057008.

- ♣ [59]. Heli Amarasinghe and Karmouch, A. (2016). SDN-Based Framework for Infrastructure as a Service Clouds. [online] pp.782–789. doi:https://doi.org/10.1109/cloud.2016.0108.

- ♣ [60]. Medium.com. (2024). *Restricted*. [online] Available at: https://medium.com/@dishadudhal/performance-evaluation-of-sdn-controllers-using-cbench-and-iperf-e9296f63115c.

- ♣ [61]. Ryait, D.K. and Sharma, Dr.M. (2020). To Eliminate the Threat of a Single Point of Failure in the SDN by using the Multiple Controllers. *International Journal of Recent Technology and Engineering (IJRTE)*, [online] 9(2), pp.234–241. doi:https://doi.org/10.35940/ijrte.b3433.079220.

- ♣ [62]. YADAV, J.S. and SAI, M.A. (2023). *Securing SDN Data Plane:Investigating the effects of IP SpoofingAttacks on SDN Switches and its Mitigation : Simulation of IP spoofing using Mininet*. [online] DIVA. Available at: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1782968&dswid=-5050

- ♣ [63]. Suleman, N.A., Mustafa, N.A., Kayani, R., Raza, A. and Saleem, N.A. (2023). REVIEW OF SECURITY ATTACKS ON SOFTWARE DEFINED NETWORKING. *Pakistan journal of scientific research*, 3(1), pp.60–80. doi:https://doi.org/10.57041/pjosr.v3i1.966.

- ♣ [64]. mvnrepository.com. (n.d.). Maven Repository: Search/Browse/Explore. [online] Available at: https://mvnrepository.com/.

- ♣ [65]. www.sciencedirect.com. (n.d.). *Floodlight Controller - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/computer-science/floodlight-controller

Use-Cases – SDN Open-Source Tools' Table

Open-Source Tools	Use-Case | Description - Usage
Mininet SDN Framework	It will be the platform that will provide a virtual testing environment for our Software-Defined Network | Use-Case: C.4
SDN Controller FloodLight	Java-Based OpenFlow Controller for SDN which will be utilized as one of the primary tools for Distributed Denial-of-Service Attack in our virtual SDN environment | Use-Case: C.4
Open Virtual Switch (OVS)	This is a multi-layer virtual switch supporting Apache 2.0 and it will be utilized to produce and manage our virtual switching devices | Use-Case: C.4
SFlow-RT	Real-Time analysis open-source tool, utilized for network traffic monitoring and metric, specifically for Software-Defined Network environment and it will be utilized to monitor the state of the controller | Use-Case: C.4

Installation Guide – Open-Source Tools (In Ubuntu/Kali Linux)

The use-case C.4 will be demonstrated in the Operating System Ubuntu (22.04 LTS), however it can be applied to the Kali Linux environment as well. The use-case is executed in Virtual Environment hosted by the VirtualBox (more specifically by the physical host machine, due to the hypervisor). For further details regarding the installation of the VirtualBox are found in the link description https://www.virtualbox.org/ as well as for the VMs Ubuntu and Kali Linux from the following links: https://medium.com/@maheshdeshmukh22/how-to-install-ubuntu-22-04-lts-on-virtualbox-in-windows-11-6c259ce8ef60, https://www.kali.org/get-kali/#kali-platforms.

The installation process is the same for either of the two OS, as follows:

- **Mininet SDN Framework:** Open a terminal instance after VirtualBox and a Kali Linux/Ubuntu virtual machine have been setup and afterwards execute the commands **sudo apt-get update -y** and **sudo apt-get install mininet -y**.

- **SDN Controller FloodLight:** In the terminal's command line, it is vital to install ant via the command **sudo apt install build-essential ant python2-dev openjdk-8-jdk maven git** and afterwards clone the FloodLight repository from github, **sudo git clone https://github.com/floodlight/floodlight**.

Then, head to the website of maven repository https://mvnrepository.com and install the jar files for the libraries libthrift ver 0.14.1 and netty-all ver 4.1.66.Final. Head to the terminal, and from there to the ~/floodlight/lib directory and there remove the jar files libthrift-0.9.0.jar, netty-all-4.0.31.final (it is not necessary for them to be in the aforementioned versions).

Build the file using gedit build.xml to ensure that the jar files have been removed and copy the downloaded jar files ro rhe directory ~/floodlight/lib via the commands **sudo cp libthrift-0.14.1.jar ~/floodlight/lib**, **sudo cp netty-all-4.1.66.Final.jar ~/floodlight/lib/**. Inside the floodlight directory (cd ..) check whether there are any issues with the jar versions and initialize, update the modules with the commands **sudo git submodule init**, **sudo git submodule update**. Furthermore, it is essential to ensure to execute git pull in order to fetch the latest updates from the master branch of the remote repository origin (floodlight) and merge them to our current branch. Clear the lingering files in the target sub-directory with **sudo ant clean** then compile the source Java code, and link the existing libraries according to the directives of the build.xml file via the command **sudo ant**. Finally, launch the floodlight SDN controller after everything have been setup correctly with the command **sudo java -jar target/floodlight.jar**.

- **Open Virtual Switch (OVS):** Open a new terminal instance and execute the commands **sudo apt-get update -y** and then **sudo apt-get install openvswitch-switch -y.**

- **SFlow-RT:** In the terminal execute wget in order to acquire the software package from inmon, **wget https://inmon.com/products/sFlow-RT/sflow-rt.tar.gz**, extract the file into a directory via **sudo tar -xvzf sflow-rt.tar.gz**. Afterwards, redirect to the website https://sflow-rt.com/download.php, which entails further instructions regarding application packages for SFlow-RT. With the command **./sflow-rt/get-app.sh sflow-rt [application_name]**, applications such as browse-metrics, ddos-protect can be installed in the current tool and after a restart **./sflow-rt/start.sh** the applications will be working properly.

Additional Files – Use-Case: It is noteworthy to mention that all the results of the captured packets via Wireshark found in the third section of the thesis are found in the file Wireshark_Findings_Use_Case_#1.zip.