Journal of Network and Systems Management (2020) 28:605–626
https://doi.org/10.1007/s10922-020-09518-z

Application-Aware Firewall Mechanism for Software Defined Networks

Fahad N. Nife[1,2] · **Zbigniew Kotulski**[1]

Received: 28 June 2019 / Revised: 21 November 2019 / Accepted: 27 February 2020 /
Published online: 18 March 2020

Abstract
Software-Defined-Networking (SDN) has been recently arising as a new technology in the IT industry. It is a network architecture that hopes to provide better solutions to most of the constraints in contemporary networks. SDN is a centralized control architecture for networking in which the control plane is separated from the data plane, the network services are abstracted from the underlying forwarding devices, and the network's intelligence is centralized in a software-based directly-programmed device called a controller. These features of SDN provide more flexible, programmable and innovative network's architecture. However, they may pose new vulnerabilities and may lead to new security problems. In this paper, we propose the application-aware firewall mechanism for SDN, which can be implemented as an extension to the network's controller. In order to provide more control and visibility in applications running over the network, the system is able to detect network applications that may at some point affect network's performance, and it is capable to dynamically enforce constraint rules on applications. The firewall architecture is designed as four cooperating modules: the Main Module, the Filtering Module, the Application Identification Module, and the Security-Enforcement Module. The proposed mechanism checks the network traffic at the network, transport, and application levels, and installs appropriate security instructions down into the network. The proposed solution features were implemented and tested using a Python-based POX controller, and the network topology was built using Mininet emulation tool.

Keywords Software-Defined-Networking · Application-Aware Firewall · Packet Filtering · OpenFlow Protocol

✉ Fahad N. Nife
fnife@mion.elka.pw.edu.pl

Zbigniew Kotulski
zkotulsk@tele.pw.edu.pl

[1] Warsaw University of Technology, Warsaw, Poland

[2] Al-Muthanna University, Muthanna, Iraq

1 Introduction

It is hard to find a facet of our life that does not involve a computer system, at least on some levels. In modern business climate, computer networks have become a key element in providing important connections for an organization to run its applications and provide services. In today's world, there are thousands of applications that use the Internet to provide different services, such as online shopping, data transfer, instant messaging, or video conferencing. They use a wide range of different communication protocols. Knowledge of the application protocols running in the network is very important for network operators. With this knowledge, they are able to properly configure networks and provide fine-grained security.

For instance, identifying Layer 7 protocol lets network operators to sort traffic according to which application or application service the traffic is trying to reach, and what the specific contents of that traffic are. Instead of simply blocking all traffic on a particular port, they can use an application-aware firewall to generally accept traffic on that port but block any traffic that contains a known vulnerability (like an SQL injection attack or a malicious telnet command).

Over the last few years, SDN has been one of the trending topics in network technologies. It has received a considerable attention from, both, researchers and IT industry, believing that SDN can provide creativity, invention and best solutions to most chronic problems experienced by existing networks [1]. The key feature of the SDN model is the physical separation between the control plane and the data plane [2]. Whereas, the control plane of all network devices is migrated from the forwarding elements and grouped into a software-based device known as a "controller". Typically, SDN architecture encompasses three distinct planes: the data plane, the control plane, and the application plane.

The data plane involves network devices connected to each other [3]. This plane is responsible for data forwarding as well as gathering local information and statistics. The control plane contains one or more connected controllers. It strives to behave like a network operating system. It represents the brain of the network in which the network intelligence is centralized. The controller's comprehensive view of the entire network helps to make forwarding decisions for the underlying network elements. For further liberation, rapid innovation, and high scalability, network services are extracted from basic network elements; they are clearly presented as a separate application plane.

Typically, the SDN Controller handles traffic (Sect. 2.1) with respect to low level traffic identifiers (e.g., header field values) and physical identifiers (e.g., an interface identifier) [4]. Thus, the SDN Controller has no insight into relations between flows. Due to the lack of high-level traffic identifiers, it is not possible to directly and consistently specify policies for all flows belonging to a specific application or to define policies for specific persons that are the parties of communication.

Motivation: On the one hand, greater reliance on software, direct programming capability, and centralized logical network intelligence of the SDN-based

network can support rapid updating and provides different ways and opportunities to enhance and protect the network [5]. On the other hand, these features bring new vulnerabilities related to security, scalability, and flexibility [6, 7]. Furthermore, the primary design of the SDN architecture does not sufficiently take into account security requirements what makes security issues a real challenge. Thus, it is essential to build a robust security mechanism to protect the network from internal and external malicious activities while respecting the objective of the SDN network. In other words, one must build a mechanism with application-aware capabilities to inspect the network traffic, which can have a plane-like firewall that creates additional boundaries within the network, and which provides "defense-in-depth" usually considered as a good solution for networks with different levels of trust.

Contributions: In this paper, we use the advantages of the central control unit with its global network view, the benefit of sending the initial flow packets to the controller for routing information (reactive routing), and the SDN data plane programmability to propose a firewall mechanism able to recognize applications in order to ensure security in SDN networks. Such a firewall will act as an application based on the controller's API. Our proposed solution aims to set up application-based traffic filtering mechanism, which is implemented as a modular application running at the top of the controller. The method is based on converting each forward element to a defense point, making it easy to protect the network from external attacks as well as from internal malicious users. The network security policies are centralized in the controller, which is the unit responsible for deciding regarding how the switches should handle the packets. The proposed firewall can be implemented and deployed effectively on small, local, and lightly utilized networks, where it can detect network applications that may at some point affect network performance.

This paper is organized as follows. Section 2 describes the theoretical background of OpenFlow-based SDN networks and firewall technologies. The related work and state of the art of current SDN security solutions are presented in Sect. 3. Section 4 highlights the proposed system's architecture. Implementation and evaluation of the system are given in Sect. 5. Section 6 concludes the paper with a summary of its content and possible future enhancements.

2 Background

To realize our proposal system and link it with other existing network solutions, we start with a brief description of the OpenFlow-based SDN architecture and firewalls technologies. Understanding the main characteristics and functionality of these components plays a significant role when designing solutions to secure SDN networks.

2.1 OpenFlow/SDN Network

The OpenFlow-based SDN architecture can be described as the formation of several interrelated and collaborative planes, as shown in Fig. 1. SDN architecture

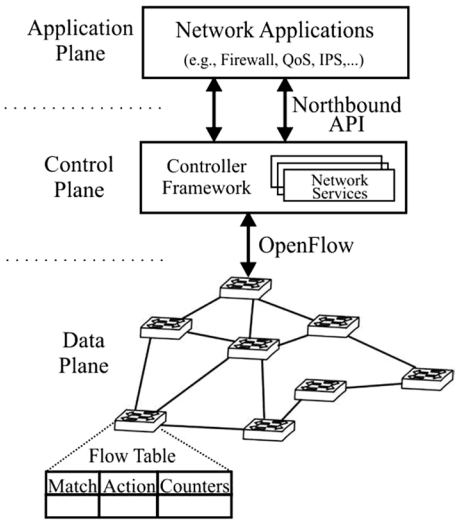

Fig. 1 OpenFlow/SDN architecture

consists primarily of three basic planes known as the data plane, the control plane, and the application plane. Each plane performs specific functions and takes advantage of information or services provided by other planes. Typically, two interfaces: the Northbound interface (NBI), and the Southbound Interface (SBI) are implemented by the control plane to communicate with the application and data planes, respectively.

OpenFlow [8] is a widely used protocol in the Southbound in SDN framework. Open Networks Foundation (ONF) specifies OpenFlow as a standard communication protocol that defines the interaction between OpenFlow-enabled switches and the OpenFlow controller. Speaking broad, OpenFlow sets different messages that provide means for the remote controller to control the behavior of network switches by adding, deleting, or modifying flow table entries in these switches.

The control plan contains a central control software program, known as the OpenFlow-enable controller, has a network-wide view of the whole network and is responsible for making a forwarding decision. The data plan consists of OpenFlow-enable switches that are basic forwarding devices which utilize a simple "Match-Action" mechanism to perform flow-based packets forwarding according to their flow table. The flow table contains a set of flow entries (also called flow rules) each of which consists of headers fields, activity counters, priority, timeout, actions, cookie and flags.

In the OpenFlow 1.0.0 switch, the header fields (Match fields) are 12-tuples packet header to be matched against the incoming packet's header. Counters fields are updated only when packets are matched and used to keep track of each flow, table and port and provide statistics to the Controller. The action fields contain data, where there is a set of zero or more actions that should be applied to the matched packets.

For each flow entry added to the switch, the priority field is linked to determine which flow should be specified if more than one flow entry matches the incoming

packet, the idle timeout value refers to the time the input must be removed due to a lack of activity, and the hard time refers to when should remove the entry, regardless of the activity.

Each packet received by the switch, the packet's header fields is matched to the flow table entries. The lookup process begins with the first flow table entry till the last flow table entry. If a match occurs, the relevant actions in the corresponding flow entry are performed on the packet. Otherwise, the packet being dropped or the packet_in message containing the packet (or only 128 bytes of the packet) is sent to the controller for processing.

Depending on its policy rules, the controller will decide how the switch should handle this packet, then send it back to the corresponding switch using the packet_out message. According to that decision, the switch will handle the packet and the controller may modify the switch's flow table by adding new entry to help handling next similar packets [9].

2.2 Firewall

Firewall is a fundamental security mechanism that controls the flow of traffic between a trusted network (i.e., a corporate LAN) and an unreliable or public network (i.e., the Internet). It can be software-based solution, a hardware-based solution, or a combination of both, used to implement an enterprise security policy that governs network traffic [10]. Each firewall uses a database of policy rules that determine how the firewall handles inbound and outbound network traffic [11]. Firewalls can work (filter) at different levels of the TCP/IP protocol stack. Accordingly, there are different categories of the firewall: static packet filters, stateful firewalls, and application-level firewalls.

The packet filtering-based firewall (also known as a stateless firewall) works by allowing or dropping packets, based on their source or destination addresses or their port numbers. Each packet is maintained separately, and the firewall does not save the state of the packet to be used for processing the next packets in the same flow [12]. On the other side, stateful firewalls track the status of network connections when packets are filtered. They often store information about each traffic flow that passes through them in a table known as the state table. This state table consists of entries that represent the currently active connection session [13]. These firewalls are common because they are inexpensive, simple in operation and maintenance, and have a good throughput. However, these types of firewalls (stateless/stateful) do not needs to understand much about the traffic they are inspecting, since they filter packets basing on source and destination addresses and may look at UDP/TCP port numbers and flags. They are not smart enough to realize the application to prevent breaches and attacks occurring across the network.

Today most applications can be delivered via a web browser or through a computer program. The vast majority of these applications must have an Internet connection in order to function correctly. The application-aware firewall is a kind of technology that redefines the way the corporate networks are secured. The main

difference is that application-based firewalls can detect and restrict certain applications, whereas the traditional firewall can only block ports or addresses [14].

This is a major advance because the application blocking functionality is used to organize separate devices on the network. For example, most Web applications and websites are running over port 80, and an organization that uses a traditional firewall should allow the port to easily enable employees to access the Internet. If they want to block access to YouTube or prevent sending files via File Transfer Protocol (FTP) for example, they should use another part of technology, such as a web filter. Using the application-aware firewall, the organization can only restrict YouTube and identify and block malware detection in an attempt to disguise port 80 traffic.

3 Related Work

With regard to network security, many previous works and contributions have been presented that offer different ideas for securing the SDN environment [15]. Some studies have focused on building firewalls solutions, while others have dedicated efforts to provide traffic classification and application-aware framework attended for SDN-based networks.

3.1 Firewall Solutions in SDN

SDFS [16] introduces Stateful Distributed Firewall provided as a service in SDN. In order to ensure more reliability, SDFS automatically exchanges the state from global (controller) to the switches along the path of the traffic. In [17], the authors proposes a stateful firewall for SDN to be implemented in the data plane. In implementation, they make use of the OpenFlow protocol version 1.5.0 that enables TCP flag matching capabilities, and proactively install rules entries in the SDN switch's flow table. So, every switch maintain a local state of the active flows.

A lightweight Intrusion Prevention System (IPS) for SDN network is presented in the paper [18]. It is introduced to early detect and prevent a port scan attacks. The lifecycle of the suggested solution goes through three basic stages: Collection, Detection, and Prevention. The proposed system collect the required data by periodically requesting the flow's counter from the switches. This information is analyzed, and according to the predefined rules, the port scan is detected. Finally, it installs forwarding rules in the switches to react to the attack. A Cloud-Clustered Firewall [19] presents a novel algorithm to distribute the security policies into distributed SDN devices maintained by cloud-clustered firewall. The presented rule-placement algorithm implemented to provide a better performance and resolve the TCAM memory limit of SDN data plane.

In [20] the authors propose a new authentication system for the SDN network, which is used to verify the identity of a host upon connection to the network. The proposed mechanism denies the access of unauthorized hosts and defines different levels of privileges for each user.

Table 1 Comparison of proposed approach with existing security mechanisms

Proposed solution	Problem approached	Behavior	Layer based	Extra memory	Capabilities exploited	Performance
Stateful Distributed Firewall [16]	Packet filtering	Hybrid (reactive/proactive)	Layer 4	Yes	Centralized control and distributed enforcement	High availability
Stateful Distributed Firewall [17]	Prevent internal and external attack	Proactive	Layer 4	Yes	Distributed control	High performance
IPS [18]	Prevent a port scan attacks	Reactive	Layer 2	Yes	Centralized control	High overhead
Cloud-Clustered Firewall [19]	Packet filtering	Proactive	Layer 4	Yes	Distributed control	High performance
IDS [21]	Malicious traffic inspection	Reactive	Layer 7	Yes	Centralized control	High performance
Application-aware [23]	Traffic identification	Reactive	Layer 7	No	Centralized control	High communication
DDoS detection [24]	Malicious traffic detection	Reactive	Layer 3, 4, 7	Yes	Centralized control	High overhead
Application-aware Data Plane [27]	Application classification	Reactive	Layer 3, 4, 7	Yes	Distributed control	High performance
Application-aware firewall	Prevent unwanted application	Reactive	Layer 3, 4, 7	No	Centralized control	High communication

Table 1 summarizes and compares the security functions (and system performance) proposed with other SDN-based security solutions known in the literature. The table clearly shows that the solutions currently available try to protect the network by inspecting the traffic packets up to Layers 2, 3 or 4 of ISO/OSI stack of protocols. Such solutions are not smart enough to be conscious of the application and fail to prevent Application Layer-based (per the Open System Interconnection (OSI) model) breaches and attacks occurring over the network. On the other hand, in this paper we provide a solution that inspects the packets based on Layer 7 characteristics; it is an application-aware firewall system that dynamically reconfigure security rules and redistributes them to the underlying devices. Layer 7 intrusion prevention system is introduced in [21] called SDN-Defense. Similar to our methodology, their system Leveraging the react behavior to inspect the first k packet at the controller side. However, our approach differs from their proposed system, yet they utilize snort [22] the open-source, and signature-based IDS in order to detect and prevent malicious flows. While we propose an application-aware firewall mechanism, the system is able to detect network applications that may at some point affect network performance.

3.2 Application-Aware and Traffic Classification Solutions in SDN

Previous efforts to classify the network traffic to provide application-aware solutions for SDN networks have been investigated in several works and project's deliverable. In [23] the authors come up with SDN-based Application-Aware approach capable of identifying YouTube Streaming. It is based on Deep Packet Inspection (DPI) and direct information collected from the applications. In contrast, our system inspects packet headers to do extract the application signature.

The distributed denial-of-service (DDoS) detection solution was introduced by Braga et al. [24]. The proposed idea uses flow features to classify network traffic flows as normal and abnormal. It extracts a set of flow characteristics and passes flow statistics as parameters for an unsupervised artificial neural network (e.g., Self-Organizing Maps). This system differs from our proposed system; it relies on an artificial neural network to detect any abnormal behavior. However, our solution relies on multiple classification methods, and therefore can be described as a multiple classifier.

Another work [25] represents a design of an SDN-based traffic classification platform for an enterprise network. The main difference to our work is that it focuses on port-based and deep packet inspection for traffic classification, while we classify network flows basing on port-based and deep packet inspection and additionally on information extracted from packet headers.

In [26], the authors presented a general framework for the end-host of SDN networks that are integrated into the NEAT transport structure. The framework provides communication interfaces between applications and external controllers. It also includes a system of expressive policies capable of meeting a wide range of requirements.

Extending the data plane devices to provide Application-aware processing in SDN appears in the paper [27]. In order to provide the application-aware capability, the data plane devices are extended to include new actions based on information from L4 to L7 of the packet header. Thus, the switch will be able to keep some application's logic locally (instead of limiting the application's logic to the controller only). It utilizes a new application table to maintain the application-specific packet-processing actions which are installed by the controller. Implementing the traffic classifier in the data plane side (flow entry classifiers) has the advantage of being relatively fast (no more controller-switch communications), but it may face many capacities and capability constraints. Because the processing occurs within the data plane, its capabilities are dependent on the particular switch implementation, which is often limited due to constraints of the ASICs on which it is built. Having classifiers located in the controller side, they are slower than their switch implementation, but they are likely to have better support for more features besides a network-wide view for traffic classification in networks. In addition, such an implementation brings the processing back to the data plane side which goes against the spirit of SDN architecture that leave the data plane devices simple match-action forwarding elements. However, to follow the main point of SDN, the traffic in our system is classified at the controller side.

In this paper, we have expanded our previous SDN-oriented stateful firewall mechanism [28, 29] to an application-aware firewall to protect SDN networks. It is implemented as an extension of the controller. The firewall functionality is implemented through four collaborating units: the Main Module, the Filtering Module, the Application Identification Module, and the Security-Enforcement Module. The latter presents a mechanism that checks network traffic at the application level, and dynamically installs the appropriate security guidelines down into the forwarding elements.

4 System Overview

4.1 System Architecture

The system architecture is depicted in Fig. 2. The functionality of the firewall application that runs on top of the controller is implemented by four modules.

- **Main Module:** The Main Module plays the role of coordinator and administrator of other units. It serves as a rendezvous point for other units so that they can interact with each other. This module listens to a Packet_in event on the controller side.
- **Application-Identification Module:** The Application-Identification Module performs traffic classification. The classification relies on different information of the traffic flows, including port numbers, application payloads, and statistical features of the flows. It adopts three methods for traffic identification processes to perform connection-level, packet-level, and flow-level classifications (Fig. 3). The first application detection approach requires to examine the

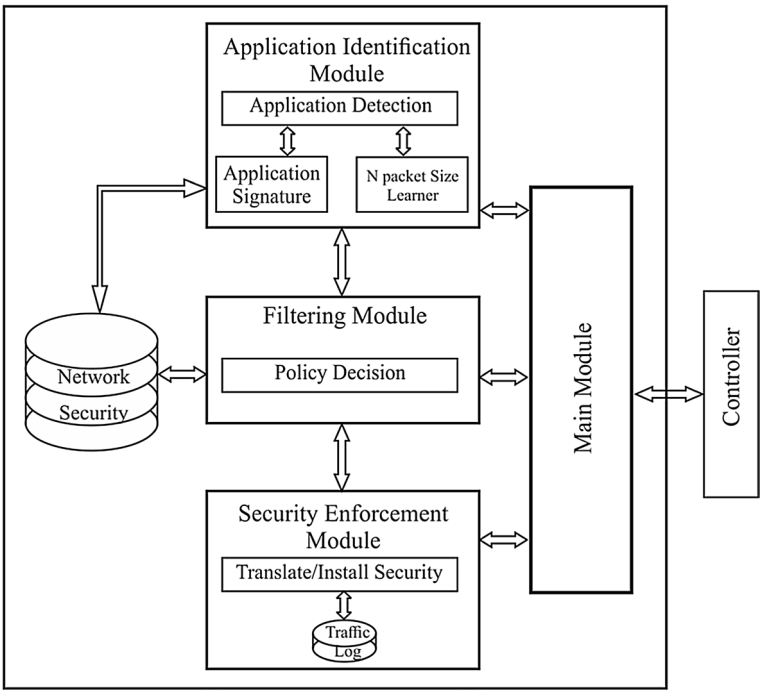

Fig. 2 The proposed system architecture for SDN

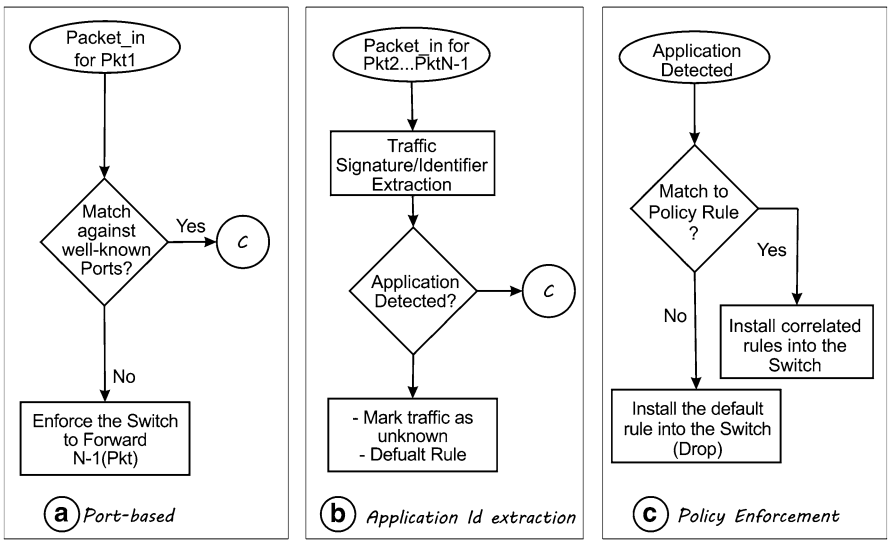

Fig. 3 Packet processing

Transport Layer header of the packet (e.g. TCP/UDP port numbers) to learn the source and destination ports in order determine the application. Using a set of well-known ports, it maps the flow to the applications as defined by IANA (Internet Assigned Numbers Authority). This approach is accurate and could be effective for many well-known applications that use registered port numbers. However, it fails with some applications that use dynamic port negotiation instead of standard port numbers. Aceto et al. study [30] reported that port-based classification is capable of identifying about 30% of network application flows correctly. However, we apply it as an attempt to reduce packet processing delay caused by the following applied stage. If the flow is not positively identified in this stage, then it goes to other identification methods.

The second method inspects the packet payload seeking for well-known keywords (a signature) that uniquely identify the protocol. Usually the signature is a regular expression located in the payload of the Application Layer. For instance, HTTP packets begin with URL instruction followed by the protocol model. In case the signature is detected, this method matches it against the list of available signatures. The third method adopts the flow features identification. It is based on analyzing the flow basing on the size and direction of each packet of the first 'N' packets in the flow as well as the source and destination ports and IP addresses to detect an application identity.

Our work relies more on the needs for prior knowledge of the protocols (supervised). Simply, by using primary network data, our algorithm classifies traffic into distinct protocols basing on the correlations between packet features. In this sense, our approach differs from the Hyunchul Kim et al. [31], (that is unsupervised), yet they are similar in spirit, both suggest using the sizes of the first ten packets per session as an identifier of the protocol. Most of the flow characteristics used are inspired by the features used in [31–33]. The features include: a protocol, source and destination ports, transferred bytes, the number of packets without Layer 4 (TCP/UDP) payload, number of TCP packets with FIN, SYN, RSTS, PUSH, ACK, URG, inter-arrival time statistics, and the size of the first ten packets.

- **Filtering Module:** The third unit is the Filtering Module; it is used to provide granular control over what traffic leaves or enters the network. The Filtering Module compares the determined application against a predefined application identification table. That is a hash table which contains match/id entries. Known applications are assigned as specific patterns in the application identification table. The match fields include a set of ports for well-known applications to be used by the first method (that is port-based), and a set of predefined signatures and identifiers that uniquely define the applications to be used by the second and third methods. Flows that could not be classified by the three processes are categorized as unknown.
- **Security-Enforcement Module:** The Security-Enforcement Module is used to enforce the required firewall services according to policy rule sets. It is used to translate and install the predefined action related to the identified application in the corresponding switch, using the OpenFlow modification message. The rules are removed from the switches using OpenFlow timeout mechanism. The abso-

lute timeout means that if no packets reach the flow for a specific duration (called Idle timeout), then the flow is removed from the device.

The firewall modules maintain two lists: (1) a list of filtering rules to identify applications, and (2) a list of firewall rules for security policies on the entire network. The former list is implemented as a hash table. The latter list, for implementation simplicity, is implemented as a CSV file that contains a multi-line of comma-separated values representing the network policy, as described later in Sect. 4.2. Both lists are stored on an SDN controller and are accessed by the firewall modules.

4.2 Firewall Policy

In order to achieve the core functionality of the network, that is packets delivery, the behavior of network's elements can be defined by two types of policies:

- **Routing policy** to specify the path of the packet to traverse the network from ingress to egress.
- **Firewall policy** to describe whether those packets are permitted or dropped.

Firewall policies are modeled as a list of priority rules. This is compatible with the matching tables (Flow table) in the OpenFlow switches, in which each rule is a tuple of matching fields and a binary decision field serving as the action field. The decision field has one bit to specify if this rule drops or permits the packet (this is also supported by OpenFlow).

In this paper, we are primarily concerned with building an application-aware firewall for small localized SDN networks. Therefore, we have basically adopted a simple implementation of how to places the rules in the switches to accomplish a specific firewall policy.

In our centralized firewall policy, the switches are named using indexes, see the relevant notation that is listed in Table 2. Each switch has a set of ports. Some of the ports are used to connect the switches to each other, the other ports may be

Table 2 Firewall policy notation

Symbol	Description
S_i	The switches index in the network, switch i
$P_{i,j}$	The policy attached to switch i ingress port j
$R_{i,j,k}$	A single rule where $R_k \in P_{ij}$
$m_{i,j}$	The matching field of $R_{i,j,k}$ rule
$d_{i,j}$	The decision of the $R_{i,j,k}$ rule
$t_{i,j}$	The priority of the $R_{i,j,k}$ rule

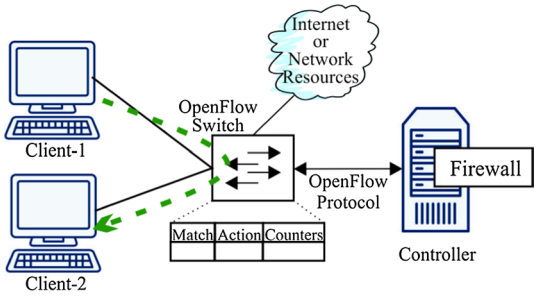

Fig. 4 The layout of the experimental testbed

the entry/exit points for the network. All firewall rules are placed on the ingress switches to ensure the least network traffic.

4.3 Example Scenario

Now, let us discuss the function of the firewall application, and how it interacts with the controller and the OpenFlow-enabled switches. Suppose, we have a network that relies on the SDN architecture, as shown in Fig. 4, and initially, all flow tables in the OpenFlow-enabled switches are empty. Assume that Client 1 wants to communicate with Client 2. Therefore, Client 1 transfers the packets addressed to Client 2. When the first packet (say pkt1) is reached to the switch (say S1), and because there is no rule in the flow table of S1 to process such a flow, pkt1 (full packet) is encapsulated in the Packet_in message and forwarded to the Controller. On the controller side, a Packet_in event will trigger the event listener in the Main Module, which responds by parsing the packet and invoke other modules. The application-identification module attempts to identify the application, it checks the packets information against a list of firewall rules, Sect. 3.2. It works by allowing or dropping packets, basing on their source or destination addresses or their port numbers.

In case the application is not determined for some reason (e.g., traffic with unknown port number), the next approaches are performed.

Here the other two methods are invoked, and the OpenFlow-enabled switch is instructed to mirror the first 10 packets to the application detection function. The first 10 packets carry valuable information to recognize the applications as well as it ensures that the controller/firewall does not get overwhelmed of information passed to it when processing new type of applications.

Application signature and Packet size learner are methods implemented to determine the application that issued traffic on the network. Application signature inspects the payloads of the packets looking for the well-known keywords that identify the application.

As the features of known applications are mapped to specific patterns in the application identification database. Packet size learner extracts the features of the first 10 packets. The extracted signature is compared to the content of the database. If it matches occur, then the application of the traffic is detected. If this stage also fails

(i.e., unknown flow), the default rules for drop or permit with further inspections are applied. After then the Filter module applies the appropriate action to the flow.

Finally, the Security-Enforcement Module translates the policy rules related to the matched signature to be installed by the Controller into the corresponding switch using the OpenFlow modification message. Accordingly, the switch will handle the remaining packets in the flow basing on the taken and installed decision.

5 Implementation and Performance Evaluations

5.1 Implementation

We validate and evaluate our proposed system by utilizing the logical experimentation setup delineated in Fig. 4 to build the entirely virtualized test environment. The testbed consists of the following virtual machines (VMs):

1. The application-aware firewall is implemented as a modular application running on the top of the POX SDN controller framework [34]. The controller is installed on the Ubuntu Desktop 18.04 LTS VM.
2. To implement the switch functionality, Ubuntu Desktop 18.04 LTS VM runs the network emulator Mininet Network [35] to create Open vSwitch (OVS) [36]. This OpenFlow-enabled switch is used at L2 / L3 layers and provides a connection to the upper layers. It helps to link different entities for the purpose of testing.
3. Finally, two client nodes are represented by Ubuntu Desktop 18.04 LTS VMs.

The Controller and each Client got a dedicated interface connected to the OVS switch directly. The firewall controller VM node is connected to the OVS switch (Open vSwitch 2.9.0) node via a 1 Gbps link, while the clients VM nodes are connected to the OVS switch node via 100 Mb/s links. The controller and OVS switch communicate via an OpenFlow 1.0.0 (the only version supported currently by the POX controller).

This setting will certainly not show any bottleneck, since the capacity of the link for the firewall is many times greater than the overall traffic, which is unusual in real-life scenarios. However, the main concern here is to observe how much time is required by the proposed firewall to process the packet_in and then install packet_mod down into the switch for new connections.

For this experiment, we selected the list of default ports basing on information from IANA database [37]. The dataset which consists of labeled flows with a rich set of features is prepared from the public data that the MAWI Working Group has gathered [38].

Table 3 The simulation settings parameters

Parameter	Value	Value
Tool	D-ITG [39]	Ping
Protocol	UDP	ICMP
Sender port, destination port	9400, 9500	–
Packet size	64, 128, 256, 512, 768, 1024, 1280, 1470 bytes	56 bytes
Concurrent streams per second	50 , 100, 150 , 200, 250, 300, 350, 400	25, 50, 100, 150, 200, 250, 300
Firewall rules size	500	25, 50, 100, 250, 500, 1000

5.2 Performance Evaluations

In order to prove effectiveness, to present behavior, and to analyze the system's performance, different scenarios are conducted and tested. The experiment settings of the used testing tools are presented in Table 3.

As the firewall is expected to handle a vast number of traffic packets and maintain a large number of firewall rules, in the first experiment we evaluate the performance of the system by focusing on the scaling firewall rules and checking out how it could affect the generated latency. The number of firewall rules varies from 25 to 1000 rules.

In this test, one Client runs a script that uses the ping utility, which is implemented to send multi ICMP packets simultaneously to the other Client. The relations

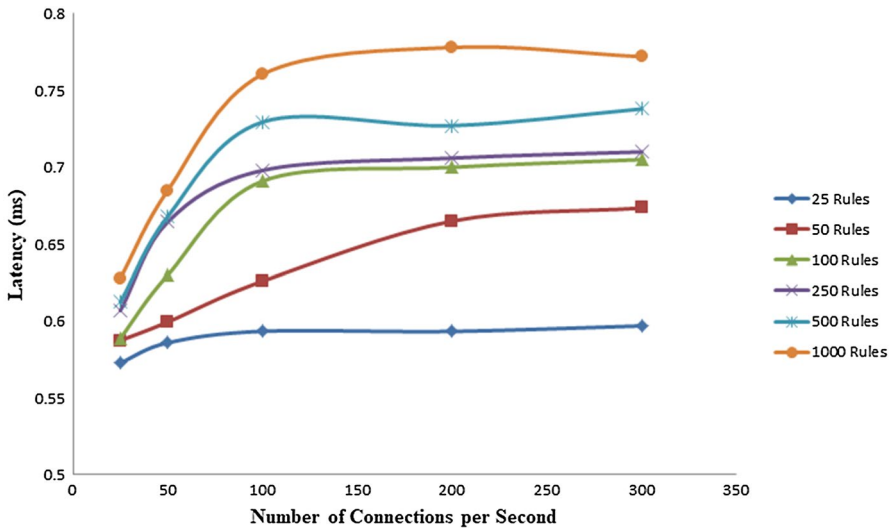

Fig. 5 The relation between the number of concurrent flows, number of firewall rules and an average latency

between the number of flows, number of firewall rules and an average latency are shown in Fig. 5, where we can observe that when the number of firewall rules is small (e.g., 25 rules), the average latency is still small for different numbers of arrival rates of flows. If there are 50 rules in the firewall base, the average latency is increased linearly according to the number of access rate for the new traffic.

After that, we increase the rule base size of the firewall to be 100, 250 and 500, respectively. We note that the average latency shows sudden increases under higher rates of flow arrival. Furthermore, we observe that the average latency stops the continuous rise and stays almost steady when the simultaneous connections are greater than 100 parallel connections. This could be because of many new traffic streams may match the rule entries in the flow table in the switch.

The following series of experiments shows the proposed system effectiveness. To evaluate the overall system performance, we observe and compare the difference in throughput in two different testbed setup settings. In the first setting, the OVS switch is connected to the POX controller acting as a Layer 3 learning switch (i.e., a baseline for the comparison). In the second configuration, we run the application firewall with the POX controller and deactivate the l3_learning switch module.

In both cases, we use Distributed-Internet Traffic Generation (D-ITG) tool [39], where one Client creates and sends the same number of concurrent UDP flows over the same time period, while the other Clients' nodes receive traffic flow and report the results.

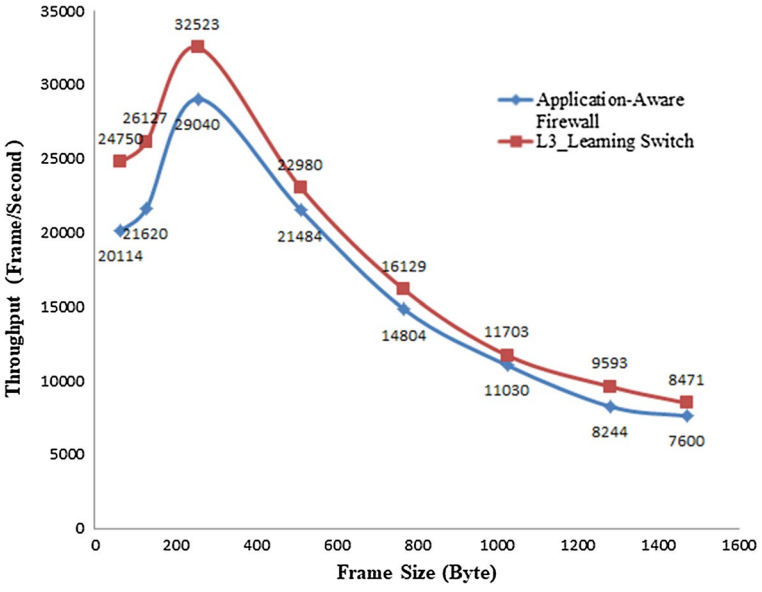

Fig. 6 Frame forwarding rates at different frame sizes

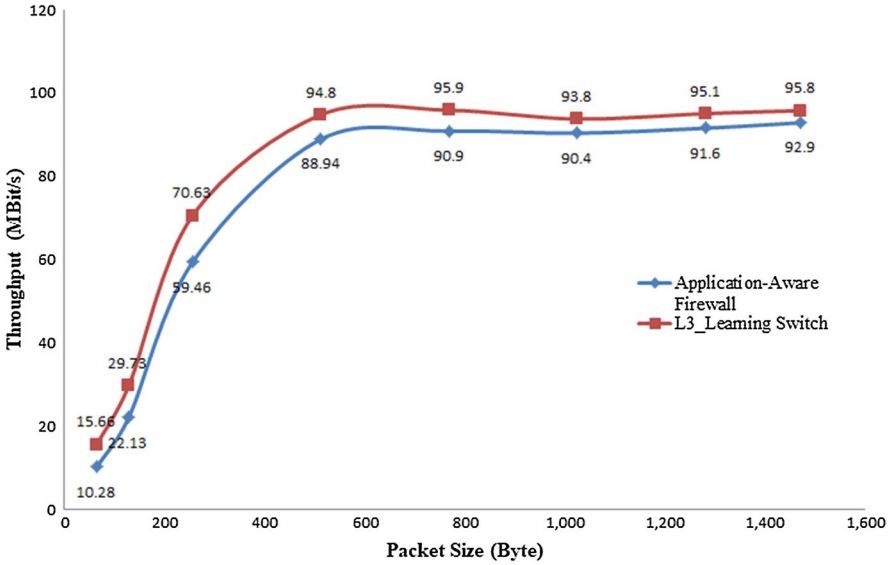

Fig. 7 Throughput at different frame sizes

To determine the maximum throughput, the frames-per-second sending rate is altered to find the highest rate for a given frame size. The Ethernet frame sizes used in the tests are shown in Table 3.

Each frame rate lasts for 30 s, with a controller killed and topology restarted between trials. We repeated each trial 5 times and the mean values of the results is calculated and considered. Our methodology approach is based on Sections 9.1 and 26.1 of [40]. However, it differs from [40] in two ways: we kill the controller and do restart the topology rather than wait for 60 seconds between trials, and we consider a unidirectional flow rather than a bi-directional flow. The relation between the frames per second performance against frame size for both test configurations are highlighted in Figs. 6 and 7 shows the throughput in Mbps against the frame size.

As expected, the learning switch controller throughput outperforms the firewall (which requires more computations) in all frame sizes. Both configurations begin with low throughput at frame size 64 bytes, and then their performance is rapidly increased within the 128–256-bytes range, reaching the peak in performance with 256–512-byte frames. There is no obvious indication for the cause of this behavior, although it was common in both configurations. In the range of 512–1470-bytes, the throughput stops the continuous rise and remains almost steady.

In general, the behavior and the performance of the learning switch controller do not appear to be significantly different with the firewall. The highest difference in throughput between them is reached with a frame size 256-bytes, while the lowest was at 1470-byte frame size.

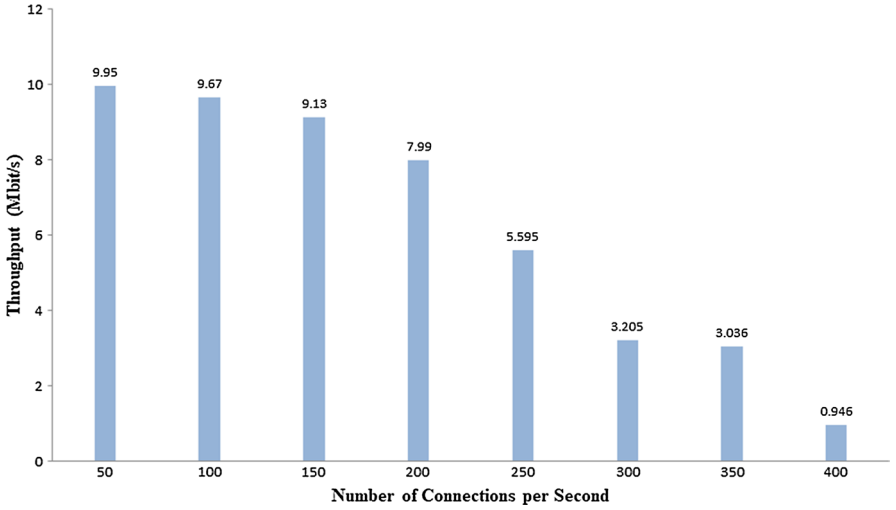

Fig. 8 The relation between the amount of traffic and the maximum throughput

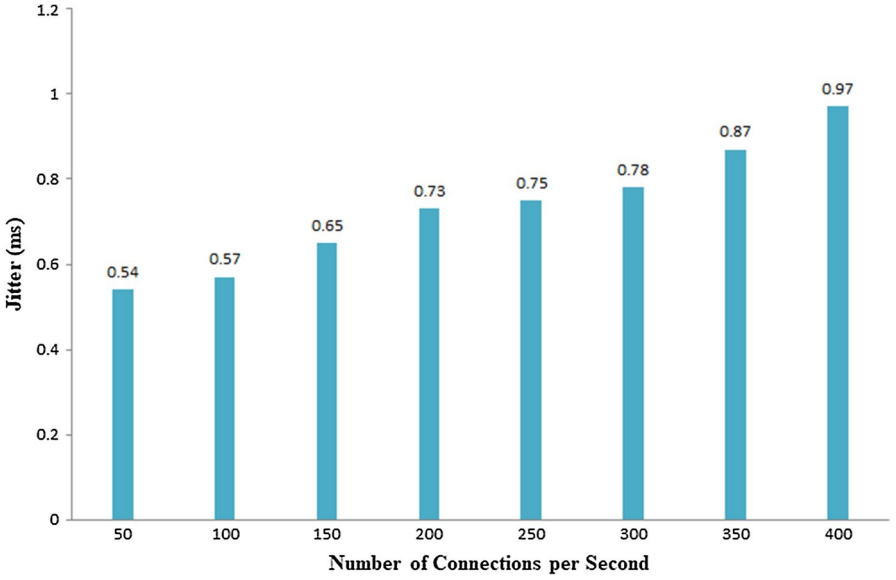

Fig. 9 The relation between the amount of traffic and the average jitter

One more test regarding the throughput is performed to show the relation between the amount of concurrent flows and the system throughput, Fig. 8. In this test, Client 1 sends simultaneous connections with randomly distributed packet size between 64- and 1470-bytes. Figure 8 clearly shows that the throughput is very high with the small number of simultaneous flows, for example, between 50

and 150 connections. However, the rapid decrease of throughput is observed with more connections, for example, more than 200 connections.

Next, we focused on the packet delay jitter, where the relation between the amount of traffic and the average jitter is observed.

Also, in this test, Client 1 sends simultaneous connections with random sizes. Figure 9 shows that the average jitter is increasing slowly as the number of connections is increasing.

6 Conclusion

Securing SDN networking is the key to the success of such a technology, but it is also a significantly challenging task. In this paper, tried to solve the problem of how to provide application-aware filtering capability for SDN controller using the fact that the SDN Controller handles traffic with respect to low level traffic identifiers and to physical identifiers. However, this makes that the Controller has no insight into the relations between flows. Such a lack of high-level traffic identifiers makes that it not possible to directly and consistently specify policies for all flows belonging to a specific application or to define policies for specific persons that are the parties of communication.

In this paper, we proposed an application-aware firewall approach for securing OpenFlow-based SDN networks. The solution does not impose any change on the SDN model in terms of design and behavior. Moreover, it tries to enhance programmable network's security and simplifies security management.

This proposed solution attempts to protect the small localized network by detecting applications such as FTP or blocking streams according to the predefined policy. The security policy is centralized in the controller side, the firewall application above the controller inspects the traffic flows and enforces the required policy, and the switches act as a distributed checkpoint that implements the controller instructions. Rather than using a gateway firewall with a low performance and a single point of failure, multi plane-like firewall creates layers of defense; "defense-in-depth" can be used successfully for networks with different levels of trust.

The future work could increase the level of security provided to the network by building a system that can examine the contents of the packets to prevent any malicious or somehow undesirable content. The system could be integrated with quantum steganography-based authentication protocol that authenticates an embedded secret message [41]. The next step of the studies should be an improvement of the system by building a Layer 7 firewall with an unsupervised mechanism based on machine learning to identify the network traffic and to block unauthorized and malicious packets. Our plan for improvement of the system is making it user-aware and application-aware, and as an effect, using application- and users-based security policies, rather than continuing the traditional methods that are based on ports and protocols. Also, we will strive to integrate the system with a soft-methods-based security tool (a multi-tier trust and reputation system) which is a reasonable enhancement of a firewall in widespread and untrusted networks (see, e.g., [42]).

References

1. Cox, J.H., Chung, J., Donovan, S., Ivey, J., Clark, R.J., Riley, G., Owen, H.L.: Advancing Software-Defined Networks: a survey. IEEE Access **5**, 25487–25526 (2017). https://doi.org/10.1109/ACCESS.2017.2762291
2. Singh, S., Jha, R.K.: A Survey on Software Defined Networking: architecture for next generation network. J. Netw. Syst. Manag. **25**(2), 321–374 (2017). https://doi.org/10.1007/s10922-016-9393-9
3. Rawat, D.B., Reddy, S.R.: Software Defined Networking architecture, security and energy efficiency: a survey. IEEE Commun. Surv. Tutor. **19**(1), 325–346 (2017). https://doi.org/10.1109/COMST.2016.2618874
4. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
5. Kreutz, D., Ramos, F.M.V., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-Defined Networking: a comprehensive survey. Proc. IEEE **103**(1), 14–76 (2015). https://doi.org/10.1109/JPROC.2014.2371999
6. Heller, B., Sherwood, R., McKeown, N.: The controller placement problem. In Proc. 1st ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Helsinki, Finland, pp. 7–12, (2012)
7. Kreutz D., Ramos F. M. V., and Verissimo P.: Towards secure and dependable software_defined_networks. In: Proc. 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, China, pp. 55–60, (2013)
8. The Open Networking Foundation, OpenFlow Switch Specification (2014)
9. Astuto, B.N., Mendonca, M., Nguyen, X.N., Obraczka, K., Turletti, T.: A survey of Software-Defined Networking: past, present, and future of programmable networks. IEEE Commun. Surv. Tutor. **16**, 1617–1634 (2014)
10. Sharma, R. K., Kalita, H. K., Issac, B.: Different firewall techniques: a survey. In: 5th IEEE ICCCNT, Hefei, Anhui, China, pp. 1–6, (2014)
11. Cheng, Y., Wang, W., Wang, J., Wang, H.: FPC: a new approach to firewall policies compression. Tsinghua Sci. Technol. **24**(1), 65–76 (2018). https://doi.org/10.26599/TST.2018.9010003
12. Duan, Q., Al-shaer, E.: Traffic-aware dynamic firewall policy management: techniques and applications. IEEE Commun. Mag. **51**(7), 73–79 (2013). https://doi.org/10.1109/MCOM.2013.6553681
13. Trabelsi, Z.: Teaching stateless and stateful firewall packet filtering: a hands-on approach. In: Proc. 16th Colloquium for Information Systems Security Education, Florida, USA, pp. 95–102, (2012)
14. Krit, S., and Haimoud, E.: Overview of firewalls: Types and policies: Managing windows embedded firewall programmatically. In: 2017 International Conference on Engineering & MIS (ICEMIS), Monastir, pp. 1–7, (2017). https://doi.org/10.1109/ICEMIS.2017.8273003
15. Sahay, R., Meng, W., Jensen, C.D.: The application of Software Defined Networking on securing computer networks: a survey. J. Netw. Comput. Appl. **131**, 89–108 (2019). https://doi.org/10.1016/j.jnca.2019.01.019
16. Zeineddine A., and El-Hajj W.: Stateful Distributed Firewall as a Service in SDN. In: 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, 2018, pp. 212-216 (2018). https://doi.org/10.1109/NETSOFT.2018.8460126

17. Krongbaramee P., and Somchit Y.: Implementation of SDN Stateful Firewall on Data Plane using Open vSwitch. In: 2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE), Nakhonpathom, 2018, pp. 1–5, (2018). https://doi.org/10.1109/JCSSE.2018.8457354
18. Neu C. V., Tatsch C. G., Lunardi R. C., Michelin R. A., Orozco A. M. S.,and Zorzo A. F.: Lightweight IPS for port scan in OpenFlow SDN networks. In NOMS 2018 IEEE/IFIP Network Operations and Manag. Symposium, Taipei, Taiwan, pp. 1–6, (2018). https://doi.org/10.1109/NOMS.2018.8406313
19. Chang Y., and Lin T.: Cloud-clustered firewall with distributed SDN devices. In: 2018 IEEE Wireless Communications and Networking Conference (WCNC), Barcelona, pp. 1–5. (2018). https://doi.org/10.1109/WCNC.2018.8377305
20. Nife, F., Kotulski, Z.: Computer networks. CN 2018. Communications in computer and information science. In: Gaj, P., Sawicki, M., Suchacka, G., Kwiecień, A. (eds.) New SDN-Oriented authentication and Access Control Mechanism, vol. 860, pp. 74–88. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92459-5_7
21. Liu, C., Raghuramu, A., Chuah C., Krishnamurthy, B.: Piggybacking network functions on SDN reactive routing: a feasibility study. In Proc. ACM SOSR '17. Santa Clara, CA, USA, pp. 34–40, (2017). https://doi.org/10.1145/3050220.3050225
22. Snort: https://snort.org
23. Jarschel, M., Wamser, F., Höhn, T., Zinner, T., Tran-Gia, P.: SDN-based application-aware networking on the example of YouTube video streaming. In: 2013 Second European Workshop on Software Defined Networks (EWSDN), pp. 87–92. IEEE, (2013)
24. Braga, R., Mota, E., Passito, A.: Lightweight DDoS flooding attack detection using NOX/OpenFlow. In 2010 IEEE 35th Conference on Local Computer Networks (LCN), pp. 408–415, (2010)
25. Ng, B., Hayes, M., Seah, W.K.G.: Developing a traffic classification platform for enterprise networks with SDN: experiences amp; lessons learned. IFIP Netw. Conf. **2015**, 1–9 (2015)
26. Santos, R., Bozakov, Z., Mangiante, S., Brunstrom A., Kassler, A.: A NEAT framework for application-awareness in SDN environments. In: 2017 IFIP Networking Conference (IFIP Networking) and Workshops, Stockholm, pp. 1–2, (2017). https://doi.org/10.23919/IFIPNetworking.2017.8264887
27. Mekky, H., Hao, F., Mukherjee, S.: Application-aware Data Plane Processing in SDN. In: Proc. of the 3th Workshop on Hot Topics in Software Defined Networking, pp. 13–18, USA, (2014). https://doi.org/10.1145/2620728.2620735
28. Nife, F., Kotulski, Z.: Computer networks. CN 2017. Communications in computer and information science. In: Gaj, P., Kwiecień, A., Sawicki, M. (eds.) Multi-level Stateful Firewall Mechanism for Software Defined Networks, vol. 718, pp. 271–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59767-6_22
29. Nife, F., Kotulski, Z., Reyad, O.: New SDN-oriented distributed network security system. Appl. Math. Inf. Sci. **12**(4), 673–683 (2018). https://doi.org/10.18576/amis/120401
30. Aceto, G., Dainotti, A., de Donato, W., Pescape, A.: PortLoad: taking the best of two worlds in traffic classification. In: 2010 INFOCOM IEEE Conference on Computer Communications Workshops, pp. 1–5, (2010)
31. Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K.: Internet traffic classification demystified: myths, caveats, and the best practices. In Proc. ACM CoNEXT Conf., pp. 1–12, (2008). https://doi.org/10.1145/1544012.1544023
32. Auld, T., Moore, A.W., Gull, S.F.: Bayesian neural networks for internet traffic classification. IEEE Trans. Neur. Netw. (2007). https://doi.org/10.1109/TNN.2006.883010
33. Moore, A., Zuev, D.: Internet traffic classification using Bayesian analysis techniques. In ACM SIGMETRICS, pp. 50–60, (2005). https://doi.org/10.1145/1071690.1064220
34. Prete, L. R., Shinoda, A. A., Schweitzer, C. M., de Oliveira, R. L. S.: Simulation in an SDN network scenario using the POX Controller. In :2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, pp. 1–6, (2014). https://doi.org/10.1109/ColComCon.2014.6860403
35. de Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A., Prete, L. R.: Using Mininet for emulation and prototyping Software-Defined Networks. In: 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, pp. 1–6, (2014). https://doi.org/10.1109/ColComCon.2014.6860404
36. Pfaff, B., Pettit, J., Koponen, T., Jackson, E. J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., Casado, A.: The Design and Implementation of Open vSwitch.

In: Proc.12th USENIX Symposium on Networked Systems Design and Implementation, NSDI, pp. 117–130, (2015)
37. Internet Assigned Numbers Authority (IANA).http://www.iana.org/assignments/port-numbers
38. Fontugne, R., Borgnat, P., Abry, P., Fukuda, K.: MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. ACM CoNEXT 2010. Philadelphia, PA, (2010). http://mawi.wide.ad.jp/mawi/
39. Botta, A., Dainotti, A., Pescape, A.: A tool for the generation of realistic network workload for emerging networking scenarios. Comput. Netw. Int. J. Comput. Telecommun. Netw. (2012). https://doi.org/10.1016/j.comnet.2012.02.019
40. Bradner, S., McQuaid, J.: RFC2544-Benchmarking Methodology for Network Interconnect Devices. (1999) https://doi.org/10.17487/RFC2544
41. El-Latif, A.A.A., Abd-El-Atty, B., Hossain, M.S., Elmougy, S., Ghoneim, A.: Secure quantum steganography protocol for fog cloud internet of things. IEEE Access **6**, 10332–10340 (2018). https://doi.org/10.1109/ACCESS.2018.2799879
42. Konorski, J., Pacyna, P., Kołaczek, G., Kotulski, Z., Cabaj, K., Szałachowski, P.: A Virtualization-Level Future Internet Defense-in-Depth Architecture. *CCIS, Recent Trends in Computer Networks and Distributed Systems Security*, Part 1, vol.335, Springer-Verlag, Berlin Heidelberg, pp. 283–292, (2012). https://doi.org/10.1007/978-3-642-34135-9_29

Fahad N. Nife received the B.Sc. degree in Computer Science in 2006 from Al-Qadisya University, Iraq. He received his M.Sc. in Computer Science from Pune University, India. He is currently pursuing the Ph.D. degree in Software-Defined Networking at thWarsaw University of Technology, Poland. He is a lecturer at Al-Samawa University, Iraq. His main research interest is in Cryptography and Software-Defined Networking security.

Zbigniew Kotulski is a Professor at the Division of Cybersecurity at the Faculty of Electronics and Information Technology, Warsaw University of Technology, Poland. He received his M.Sc. in Applied Mathematics from Warsaw University of Technology and Ph.D. and D.Sc. degrees from the Institute of Fundamental Technological Research of the Polish Academy of Sciences. He is the author and co-author of five books and over 200 research papers on Applied probability, Cryptography, Cryptographic protocols and Network security.