Applied Mathematics and Nonlinear Sciences

<https://www.sciendo.com>

Design of software-defined network experimental teaching scheme based on virtualised Environment

Heng He^{1,2,†}, Yazhou Song^{1,2}, Tianzhe Xiao³, Haseeb Ur Rehman³, Lei Nie^{1,2}¹ School of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China² Hubei Province Key Laboratory of Intelligent Information Processing and Real Time Industrial System, Wuhan University of Science and Technology, Wuhan 430065, China³ International School, Wuhan University of Science and Technology, Wuhan 430065, China

Submission Info

Communicated by Juan Luis García Guirao

Received January 6th 2021

Accepted April 11th 2021

Available online May 25th 2021

Abstract

Aiming to address the shortage of experimental resources, the high cost of large-scale deployment of hardware experimental environment and the difficulty for students to get started in the software-defined network (SDN) course, this article proposes an SDN experimental teaching scheme based on the virtualised environment, and gives a specific experimental scheme design. The scheme utilises virtualisation technology to build a SDN experimental environment quickly, uses a lightweight network simulation platform – that goes by the name of Mininet – to build the SDN network and uses open-source controller Floodlight for centralised control of the SDN network. The scheme is mainly divided into three phases: basic, improvement and synthesis. In the basic phase, experimental projects mainly include the study of SDN basic concepts and the use of relevant tools; in the improvement phase, experimental projects mainly include the use of SDN flow table, group table, etc; in the synthetic phase, we design two innovative experimental projects that use computational intelligence technology to achieve efficient load balancing and accurate malicious attack detection. The difficulty of each phase is increasing. The constantly evolving levels of difficulty allow the individual needs of students with different levels to be met, thereby improving the effect of SDN experimental teaching and cultivating innovative SDN talents.

Keywords: software-defined network; virtualization; experimental teaching; computational intelligence; Floodlight; Mininet
AMS 2010 codes: 68M10, 90B18.

[†]Corresponding author.Email address: heheng@wust.edu.cn

1 Introduction

Software-defined network (SDN) is an emerging network innovation architecture proposed by Professor Nick McKeown based on SANE and Ethane [1]. In 2011, the Open Networking Foundation (ONF), as the leader and standard-setter, promoted SDN to gradually become an important topic in the research of new network architectures in recent years [2]. The core concept of SDN is to abstract the underlying network infrastructure from the application and adopt a new architecture that separates the control plane from the data plane. Further, SDN realises flexible scheduling of network traffic through open programmable interfaces, which provide a good platform for the innovation of SDN [3–6].

ONF proposes a typical logical SDN architecture. As shown in Figure 1, the tightly coupled network architecture in the logic of traditional network devices is divided into three planes: the data plane, the control plane and the application plane. The data plane contains various underlying network devices, and these are responsible for data storage and forwarding; in the control plane, the software-based centralised controller is deployed. The SDN controller is responsible for data forwarding, traffic scheduling, link bandwidth statistics and other functions of the basic plane by the OpenFlow protocol. The SDN controller also provides a unified network service to the application plane through the northbound interface; the application plane contains various software and realises the control and management of the network through the northbound interface.

Fig. 1 The typical logical SDN architecture.

Compared with traditional networks, SDN has the following advantages:

- (1) SDN uses the separation of control and forwarding to break the closed model of traditional equipment suppliers' software and hardware integration, which is conducive to optimisation at the global network.
- (2) SDN provides developers with a convenient programming interface, so that developers can create or use the corresponding SDN network according to actual needs and are no longer restricted by the traditional network.
- (3) SDN abstracts the actual physical network into multiple logical networks based on virtualisation technology, meeting the needs of different business scenarios flexibly.
- (4) SDN is an important application in the field of computational intelligence, monitoring network topology, link utilisation, traffic and other information in real time and optimising network quality through centralised mathematical statistical computation and simulation with the goal of discovering a better forwarding path for

the traffic to be adjusted.

In recent years, SDN has been introduced into the teaching of computer-related majors by an increasing number of colleges and universities [7, 8]. Princeton University, Columbia University and ETH Zurich have successively opened SDN courses. Some Chinese universities have also paid attention to the development of the future network field, but mainly through the establishment of relevant SDN laboratories. Therefore, related experimental resources are still characterised by a scarcity in supply, and the early stage construction of the course is difficult. Further, compared with the small-scale environment configuration of the laboratory, setting up a SDN course requires large-scale experimental deployment, and the cost of using hardware equipment is higher. Finally, as an emerging technology, the SDN course needs to combine various programming languages such as Java, Python and C in the experimental teaching methodology, and students need to be familiar with basic aspects of knowledge concerning Linux and the related networking functions. Therefore, it is difficult for students to commence their training [9–12]. Aiming at the above problems, this article proposes a SDN experimental teaching scheme based on the virtualised environment. Open-source virtualisation software such as Mininet and Open vSwitch is used to quickly and flexibly deploy an SDN experimental environment on a large scale. The scheme is divided into three phases: basic, improvement and synthesis. In the basic phase, experimental projects mainly include the study of SDN basic concepts and the use of relevant tools; in the improvement phase, experimental projects mainly include the use of SDN flow table, group table etc; in the synthetic phase, we design two innovative experimental projects which use computational intelligence to achieve efficient load balancing and accurate malicious traffic detection. Therefore, the scheme reduces the difficulty that students face in commencing their training with SDN and thereby increases their enthusiasm for learning SDN.

2 Virtualised experimental platform

Mininet is a lightweight network simulation platform developed by Professor Nick Meckown and his team based on the virtualisation technology of the Linux Container kernel. This platform can simulate an SDN experimental environment, and can also migrate the code of the virtualised experimental environment to the real network environment [13, 14]. Students can quickly build an SDN network on their own computers and perform related development tests. Compared with the hardware experimental environment, Mininet not only avoids the cost of expensive OpenFlow switches and other hardware equipment but also reduces the difficulty of large-scale deployment of SDN experimental environment; thus, this attenuation of the difficulty of using SDN networks – that is caused by Mininet – encourages the opening SDN courses in universities.

To effectively demonstrate the control process of the SDN controller to the network, the experiment can use Mininet to connect the remote controller, and then display the experimental effect through Mininet terminal commands or the visual management interface of the SDN controller. This article uses Floodlight as the remote controller. Floodlight is a Java-based open-source SDN controller authorised by Apache. Similar to NOX, POX and other controllers, Floodlight also adopts a hierarchical architecture and provides a wealth of application programming interfaces (APIs) that are the same as the commercial version, allowing developers to quickly migrate the code on Floodlight to other commercial versions of controllers. Floodlight is widely used in academic research and commercial projects owing to its good characteristics [15]. Also, Floodlight provides a visual management interface for the controller, through which students can view the switch information, the host information and the real-time network topology.

To summarise, the virtualised experimental platform used in the SDN teaching scheme consists of the Floodlight controller and the Mininet network simulation platform.

3 Design of the experimental teaching scheme

The SDN experiment teaching scheme can be divided into 3 experimental phases and 10 experimental projects. As shown in Figure 2, modular experiments can be carried out according to the actual situation faced by students. In the basic phase, the experimental projects include the use of basic tools such as Wireshark and Postman, as well as the installation and use of virtualised software such as Mininet and Floodlight; in the improvement phase, the experimental projects mainly include using Mininet command to create customised topologies, the analysis of OpenFlow packets through Wireshark and the learning of flow table and group table; in the synthetic phase, the experimental projects implement specific functions or mathematical models by the characteristics of SDN, such as SDN-based network load balancing and SDN-based malicious attack detection.

Fig. 2 The design of SDN experimental teaching scheme.

3.1 Basic phase

Table 1 Basic experimental projects.

Experiments	Brief introduction
Use of Wireshark, a packet capture analysis tool	Grasp the basic use of Wireshark. Master the use of filters to effectively improve the efficiency of capturing and analysing data packets.
Use of HTTP request testing tool Postman	Grasp the basic functions and usage of Postman. Master the method of API testing using Postman.
Installation and commonly used commands of Mininet	Grasp the basic concepts and functions of Mininet. Master the source code installation method of Mininet.
Installation and use of Floodlight	Grasp the background and basic architecture of Floodlight. Master the methods of installing and deploying Floodlight, be able to connect to Mininet and be able to independently solve problems during the experiment.

Table 2 Improvement experimental projects

Experiments	Brief introduction
Use Mininet to create a custom network topology	Master the method of using Mininet command to create minimal, linear and tree topologies. Master the method of using Python scripts to define topologies. Master the method of using the interactive interface to customise the topology.
Use Wireshark to analyse OpenFlow packets	Grasp the process of establishing a TCP connection between the OpenFlow switch and the controller. Master the method of configuring the OpenFlow version in the secure channel. Master the message interaction process between the OpenFlow switch and the controller.
OpenFlow flow table, group table, meter table learning	Master the basic knowledge of OpenFlow flow tables and flow table entries, matching rules and actions performed. Grasp the relevant knowledge of OpenFlow group table and meter table.
Use flow tables for simple access control	Master the use of Mininet to create custom network topologies and connect to the remote controller Floodlight. Master the use of Postman, the Floodlight visual interface and Java programs to deliver the flow table. Grasp flow table matching rules and switch operating mechanism.

3.2 Improvement phase

3.3 Synthesis Phase

Table 3 Synthesis experimental projects.

Experiments	Brief introduction
SDN-based network load balancing	Grasp the SDN load balancing method. Realise balanced scheduling of network traffic by issuing flow tables of specific matching domains.
SDN-based malicious attack detection	Grasp the malicious attack detection mechanism. Through link detection and data flow detection on OpenFlow switch ports, be able to accurately detect and defend against malicious attacks.

4 Description of the experimental teaching scheme

4.1 Experimental description

We design two innovative experimental projects in the synthesis phase: SDN-based network load balancing and SDN-based malicious attack detection. These projects mainly include grasping the mechanism of SDN load balancing and the malicious attack detection, the installation and commonly used commands of Mininet, the use of Mininet to create a custom network topology, the installation and use of Floodlight, the use of Wireshark, the

coding of Java programs to deliver OpenFlow flow tables etc.

Load balancing refers to processing the load received from the outside and distributing it to multiple servers to maximise the use of servers, improve data transmission efficiency and enhance network stability [16]. Malicious attack detection is an important part of network supervision. The basic idea is to use mathematical metrics to describe the network traffic pattern. If there is a deviation from the normal pattern far or close to the abnormal pattern, the network is considered to be under malicious attack [17]. The difference between SDN and traditional networks is that SDN separates the data plane from the control plane of the network, thereby achieving flexible and effective control of network traffic. The typical functions in the SDN architecture are load balancing and malicious attack detection, which are closely related to the stability of the entire network. Therefore, this scheme designs an SDN load balancing experiment based on dynamic forwarding rules and an SDN malicious attack detection based on the confidence-based filtering (CBF) method. The network topology of this experiment is shown in Figure 3; the two hosts – H1, H2 – and the five switches – S1, S2, S3, S4, S5 – constitute the network topology. H1 and H2 can communicate through three different paths: S1-S2-S5, S1-S3-S5, S1-S4-S5. The experiment develops a lightweight routing control application based on the northbound API to achieve SDN load balancing efficiently, based on dynamic forwarding path rules. Further, the experiment can accurately implement the malicious attack detection and defence base through the CBF method. These projects demonstrate the complete process of the SDN experimental teaching scheme.

Fig. 3 The network topology.

4.2 Experimental environment

In the experiment, Mininet is used to simulate the hosts and OpenFlow switches and to create the experiment network topology. The memory of Mininet is 1GB. Floodlight is used as a remote controller to communicate with the switches. Floodlight1.2 and Mininet2.3.0 are both mature and stable versions, which have abundant functions. They are deployed on an Ubuntu 16.04 version host with 8G memory. The specific environment is shown in Table 4.

4.3 Installation of Mininet and Floodlight

4.3.1 Mininet

There are three main methods to install Mininet. One is to directly download and install the Mininet VM image, and then open the Mininet virtual machine on VMware or VirtualBox. The second is to install the Mininet file package, then enter the command `sudo apt-get install mininet/precise-backports` in the terminal to

Table 4 Experimental environment.

Devices	Software environment	Hardware environment
Controller	Floodlight1.2	System: Ubuntu16.04
Switches (S1~S5)	Mininet2.3.0	CPU: 4core Memory: 8G
Hosts (H1, H2)	Mininet2.3.0	Disk storage: 40GB

install. The third is to obtain the source code of Mininet installation through GitHub. This method can enable the programmer to set the version of Open vSwitch during the process, and is therefore the recommended one. The specific installation process, commands and parameters are explained as follows:

- 1) Obtain the source code by the command `git clone git://github.com/mininet/Mininet`.
- 2) Find the installation directory by the command `cd mininet/util`.
- 3) Install by the command `mininet/util/install.sh -a`.

In the command `mininet/util/install.sh -a`, the `-a` parameter means complete installation, including Mininet VM, Open vSwitch dependencies, OpenFlow, Wireshark splitter and POX etc. These tools will be installed in the home directory.

4.3.2 Floodlight

Before installing Floodlight, the experiment requires the installation of the JDK. Floodlight1.2 version requires JDK version 1.8 or higher. First, we download the JDK8 compressed package file corresponding to the Linux version on the Oracle official website and decompress it; then, we set the JDK environment variables and update the profile file; finally, we need to check whether the installation is successful using the command `java -version`.

Floodlight is downloaded and compiled mainly by the command `git clone git://github.com/floodlight/floodlight.git`. After the installation, we use the command `java -jar target/floodlight.jar` to run Floodlight. To compile, we need to execute the command `ant eclipse` before importing to Eclipse. The command `ant eclipse` needs to be executed before importing the Floodlight project into Eclipse.

4.4 SDN-based network load balancing

The flow table is an abstraction of the data forwarding function of the network device, and integrates the configuration information of each plane in the network. Therefore, rich data forwarding rules can be used. OpenFlow flow table delivery is divided into active delivery and passive delivery. Passive delivery means that when the switch receives a message that does not match the flow table record, it will forward the message to the controller, and the controller will make the decision. Active delivery means that the controller actively delivers the flow table information to the switches. There are four main methods of delivery:

- 1) Use Floodlight visual interface to operate experiments through web pages.
- 2) Use the REST API, through the command `curl` or through the Postman tool.
- 3) Use the REST API, through writing application programs (Java and Python).
- 4) Use local API to implement Floodlight module programs (Java).

This experiment uses the OpenFlow flow table module provided by Floodlight REST API to write a Java program to actively deliver the flow table and dynamically change the forwarding rules by sending a custom flow table to the switches S1 and S5. The experimental process is shown in Figure 4. In the first T seconds, we set the flow table matching domain $ipv4\_src = 10.0.0.1$, $ipv4\_dst = 10.0.0.2$, $inport = 1$, flow table survival time $hard\_timeout = T$, action set $output = 2$ and deliver it to S1. Further, we set the flow table matching domain $ipv4\_src = 10.0.0.2$, $ipv4\_dst = 10.0.0.1$, $inport = 1$, flow table survival time $hard\_timeout = T$, action set $output = 2$ and deliver it to S5. When H1 pings H2, the data packet from H1 to S1 will match the actively delivered flow table and forward the data packet from S1 to S2. Since there is no corresponding flow table on S2 and S5, the Floodlight controller will deliver the passive flow table to make the data packet pass and finally make

the data packet reach H2. Similarly, the corresponding flow table is also delivered in the second T seconds and the third T seconds to realise the different paths of data packets in different periods. The experiment distributes concentrated traffic to different switches to achieve SDN load balancing. The number of packets on each port in S1 is monitored by Wireshark to verify the experimental effect. The experimental results are shown in Figures 5 and 6. We use the command *H1 ping H2* to simulate normal user access, and set $T = 10$ s. Figure 5 describes the number of packets in S1 without load balancing. We find that the system always selects the path S1-S2-S5 by default, because the packets always come out from port eth2 of S1. Further, Figure 6 indicates the number of packets in S1 with load balancing, based on dynamic forwarding rules. The packets come out from different ports every 10 s, which proves that the system is dynamically changing forwarding path rules to efficiently achieve load balancing.

Fig. 4 The process of dynamically changing forwarding path rules.

Fig. 5 The number of packets in S1 without load balancing.

4.5 SDN-based malicious attack detection

Dou et al. [18] proposed the CBF method based on packet filtering to detect malicious attack packets. The CBF method selects six attributes: Total Length, TTL, Protocol Type, Source IP Address in the IP header and Flag and Destination Port Number in the TCP header, which make up a total of 15 attribute pairs in 2 pairs. The higher frequency of the attribute pairs' value indicates the corresponding high confidence level. Malicious attack packets are filtered by calculating the CBF score of the packet to determine the legitimacy of the packet.

Confidence values for single attributes are calculated as:

$$\text{Conf}(B_i = b_{i,j}) = \frac{N(B_i = b_{i,j})}{N_n} \quad (1)$$

Fig. 6 The number of packets in S1 with load balancing.

where $i = 1, 2, 3, \dots, n, j = 1, 2, 3, \dots, m_i$.

Confidence values for attribute pairs are calculated as:

$$Conf(B_{i_1} = b_{i_1, j_1}, B_{i_2} = b_{i_2, j_2}) = \frac{N(B_{i_1} = b_{i_1, j_1}, B_{i_2} = b_{i_2, j_2})}{N_n} \quad (2)$$

where $i_1 = 1, 2, 3, \dots, n, i_2 = 1, 2, 3, \dots, n, j_1 = 1, 2, 3, \dots, m_1, j_2 = 1, 2, 3, \dots, m_2$.

In Eqs (1) and (2), n is the number of the attributes under consideration in the method, B_i is the i -th attribute in the packet, m_i is the number of values which attributes B_i can have, $b_{i, j}$ is the j -th value of attribute B_i , N_n is the total number of packets in the packet flow in one time interval, $N(B_i = b_{i, j})$ is the number of packets whose attribute B_i has value $b_{i, j}$ in this packet flow in one time interval, $N(B_r = b_{r, x}, B_s = b_{s, y})$ is the number of packets whose attribute B_r has value $b_{r, x}$ and attribute B_s has value $b_{s, y}$ in this packet flow in one time interval.

The CBF score is calculated as:

$$Score(p) = \frac{\sum_{k=1}^d W(B_{k_1}, B_{k_2}) Conf(B_{k_1} = p(k_1), B_{k_2} = p(k_2))}{\sum_{k=1}^d W(B_{k_1}, B_{k_2})} \quad (3)$$

In formula 3, d is the total number of the attribute pairs, $W(B_{k_1}, B_{k_2})$ is the weight of the attribute pair (B_{k_1}, B_{k_2}) , $p(k_1)$ is the value of B_{k_1} in the data packet and $Conf(B_{k_1} = p(k_1), B_{k_2} = p(k_2))$ is the $Conf$ of the attribute pair $(A_{k_1} = p(k_1), A_{k_2} = p(k_2))$.

In this experiment, H1 sends TCP data packets through the hping3 tool to H2, to simulate malicious attacks. The hping3 is a packet generation and analysis tool for the TCP/IP protocol, which can be used to simulate SYN flood attacks, ICMP flood attacks and UDP flood attacks etc. [19] The attack density refers to the number of attack packets sent per second. We design the low-rate malicious attack experiment by the density of 100 packets per second and high-rate malicious attack experiment by the density of 10^6 packets per second. The specific function is realised by writing the Java program to monitoring network topology, link utilisation, traffic and other information in real time. In the experiment of low-rate malicious attacks, H1 sends a large number of attack packets to S1 from the 3rd second, as shown in Figure 7. At the 5th second, the traffic in port 1 exceeds the attack threshold. The system detects malicious attacks and enters the defence state, and the attack packets sent by H1 will be intercepted. In the experiment of high-rate malicious attacks, H1 also sends a large number of attack packets to S1 from the 3rd second, as shown in Figure 8. At the 14th second, the traffic in port 1 exceeds the attack threshold. The system detects malicious attacks accurately and enters the defence state, and the attack packets sent by H1 will be intercepted.

Fig. 7 Real-time traffic of S1 under low-rate attacks. The attack density is 100 packets per second.

Fig. 8 Real-time traffic of S1 under high-rate attacks. The attack density is 106 packets per second.

5 Conclusion

Endeavouring to address the scarcity in experimental resources, the high cost of large-scale deployment of hardware experimental environments and the difficulty that students face in commencing work on the SDN course, this article proposes the design of an SDN experimental teaching scheme that is based on the virtualised environment. The scheme that is studied in this article uses virtualised experimental platforms, such as Mininet simulation platform and Floodlight controller, to design basic, improvement and synthesis phases' experimental programs and describe the experimental process of SDN-based network load balancing and SDN-based malicious attack detection, thereby enriching the experimental resources of SDN courses and reducing the deployment cost of the experimental environment; further, the individual needs of different students are fully considered in this article. As a result of the continuous improvement of the SDN course system, we will, in future, consider relying on the cloud platform to meet the needs of more complex experimental environments.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 61602351), the National Natural Science Foundation of China (Grant No. 61802286), the Ministry of Education Industry-University Cooperation Collaborative Education Project of China (Grant No. 201902182010), the Undergraduate Teaching Research Project of Wuhan University of Science and Technology (Grant No. 2020 × 047) and the Graduate Teaching Research Project of Wuhan University of Science and Technology (Grant No. Yjg201719).

References

- [1] Benzekki K, Fergougui A, Elbelrhiti A. Software-defined networking (SDN): A survey[J]. *Security & Communication Networks*, 2016, 9(18): 5803-5833.
- [2] Zhu D, Zhang D, He B, et al. Design and implementation of an experimental teaching scheme for the development of SDN northbound applications[C]. In: *Proc. of the 14th International Conference on Computer Science & Education (ICCSE)*, IEEE Computer Society, Washington, DC, USA, 2019, 731-736.
- [3] Li J, Li D, Yu Y, et al. Towards full virtualization of SDN infrastructure[J]. *Computer Networks*, 2018, 143(OCT.9): 1-14.
- [4] Das T, Sridharan V, Gurusamy M. A Survey on controller placement in SDN[J]. *IEEE Communications Surveys and Tutorials*, 2020, 22(1): 472-503.
- [5] He M, Varasteh A, Kellerer W. Towards a flexible design of SDN dynamic control plane: an online optimization approach[J]. *IEEE Transactions on Network and Service Management*, 2019, (99):1-1.
- [6] Chica C, Imbachi C, Botero F. Security in SDN: a comprehensive survey[J]. *Journal of Network and Computer Applications*, 2020, 159:102595.
- [7] Matias J, Garay J, Toledo N, et al. Toward an SDN-enabled NFV architecture[J]. *IEEE Communications Magazine*, 2015, 53(4): 187-193.
- [8] Caicedo V, Prieto Y, Pezoa E, et al. A novel framework for SDN teaching and research: a Chilean University case study[J]. *IEEE Communications Magazine*, 2019, 57(11): 67-73.
- [9] Hwang J, Ramakrishnan K, Wood T. NetVM: high performance and flexible networking using virtualization on commodity platforms[J]. *IEEE Transactions on Network and Service Management*, 2015, 12(1): 34-47.
- [10] Cosgrove S. Teaching software defined networking: It's not just coding[C]. In: *Proc. of 2016 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE Computer Society, Washington, DC, USA, 2017, 139-144.
- [11] Bima E, Inalegwu C, Folorunso A, et al. Software defined network (SDN) based virtual laboratory: an implementation[C]. In: *Proc. of International Conference on Information Communication Technology and Its Applications (ICTA)*. IEEE Computer Society, Washington, DC, USA, 2016, 232-235.
- [12] Alcorn J, Melton S, Chow E. SDN on-the-go (OTG) physical testbed[C]. In: *Proc. of 2017 IEEE Conference on Dependable and Secure Computing*. IEEE Computer Society, Washington, DC, USA, 2017, 202-208.
- [13] Tadros N, Rizk M, Mokhtar B. Software defined network-based management for enhanced 5G network services[J]. *IEEE Access*, 2020(8): 53997-54008.
- [14] Jawaharan R, Mohan M, Das T, et al. Empirical evaluation of SDN controllers using Mininet/Wireshark and comparison with Cbench[C]. In: *Proc. of IEEE 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE Computer Society, Washington, DC, USA, 2018: 1-2.
- [15] Bholebawa I, Dalal U. Performance analysis of SDN/OpenFlow controllers: POX Versus Floodlight[J]. *Wireless Personal Communications*, 2018, 98(2): 1679-1699.
- [16] Jantila S, Chaipah K. A security analysis of a hybrid mechanism to defend DDoS attacks in SDN[J]. *Procedia Computer Science*, 2016, 86: 437-440.
- [17] Haddadi M, Beghdad R. A confidence interval based filtering against DDoS attack in cloud environment: a confidence interval against DDoS attack in the cloud[J]. *International Journal of Information Security and Privacy (IJISP)*, 2020, 14(4): 42-56.
- [18] Dou W, Chen Q, Chen J. A confidence-based filtering method for DDoS attack defense in cloud environment[J]. *Future Generation Computer Systems*, 2013, 29(7): 1838-1850.
- [19] Bhakthavatsalam K, Shanmugam T, Malarkodi B. Prevention of a SYNflood attack using ExtremeXOS modular operating system[J]. *International Journal of Internet Technology and Secured Transactions*, 2017, 7(4):364.

This page is intentionally left blank