DOI: 10.18721/JCSTCS.14105
УДК 004.72

# ON PROGRAMMING AN APPLICATION TO MITIGATE DOS ATTACK USING OPENDAYLIGHT CONTROLLER IN SOFTWARE-DEFINED NETWORKING

*C.D. Cajas Guijarro, D.O. Budanov*

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

Denial of Service (DoS) attacks try to deplete system resources by consuming bandwidth. In this paper the application using Software-Defined Networking (SDN) principles for DoS attack mitigation based on traffic monitoring in a network is proposed. The most important details about the programming aspects of the application using OpenDaylight (ODL) are explained. The application generates both proactive and reactive rules that should be installed in the network devices. Therefore, it is possible to have statistics of the flows and track possible anomalies such as an unexpected increase of the throughput in one or more of the flows. This allows to detect a DoS attack and mitigate it, installing the appropriate rules. Simulation results obtained with the application when using virtual switches in a network with a linear topology are presented.

**Keywords:** DoS attack, software-defined networking, OpenDaylight, controller, proactive rules, reactive rules, throughput.

**Citation:** Cajas Guijarro C.D., Budanov D.O. On programming an application to mitigate DoS attack using OpenDaylight controller in software-defined networking. Computing, Telecommunications and Control, 2021, Vol. 14, No. 1, Pp. 50–59. DOI: 10.18721/JCST-CS.14105

# ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЯ ДЛЯ СНИЖЕНИЯ ВЛИЯНИЯ АТАКИ ТИПА ОТКАЗ В ОБСЛУЖИВАНИИ С ИСПОЛЬЗОВАНИЕМ КОНТРОЛЛЕРА OPENDAYLIGHT В ПРОГРАММНО-ОПРЕДЕЛЯЕМЫХ СЕТЯХ

*К.Д. Кахас Гихарро, Д.О. Буданов*

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

Атаки типа «отказ в обслуживании» (Denial of Service, DoS) нацелены на истощение системных ресурсов за счет генерации большого количества запросов. В данной работе представлено приложение, использующее принципы программно-определяемых сетей для снижения влияния DoS-атаки, основанное на мониторинге трафика в сети. Приведено объяснение наиболее важных аспектов программирования приложения с использованием платформы OpenDaylight. Предлагаемое в работе приложение создает как проактивные, так и реактивные правила, которые могут быть установлены в сетевых устройствах. Это позволяет реализовать сбор статистики о потоках в сети и отслеживание аномалий, таких, например, как неожиданное увеличение трафика в одном или нескольких потоках. Таким образом, становится возможным обнаружить DoS-атаку и снизить ее влияние на функционирование сети, установив соответствующие правила. Результаты моделирования разработанного приложения при использовании виртуальных коммутаторов представлены для сети с линейной топологией.

**Ключевые слова:** DoS атака, программно-определяемые сети, OpenDaylight, контроллер, проактивные правила, реактивные правила, пропускная способность.

**Ссылка при цитировании:** Cajas Guijarro C.D., Budanov D.O. On programming an application to mitigate DoS attack using OpenDaylight controller in software-defined networking // Computing, Telecommunications and Control. 2021. Vol. 14. No. 1. Pp. 50–59. DOI: 10.18721/JCSTCS.14105

## Introduction

Software-Defined Networking (SDN) is a recent networking paradigm that has modified the way to work and study networks [1]. SDN implementsthe decoupling of the data and control planes. The forwarding decisions are taken by a centralized controller instead of the switches in traditional networks. Due to the switches do not perform the intelligence of the network, they can be cheaper and on the side of the controller this centralization allows to make better network decisions [2].

Despite the main SDNs advantages, there are still security attacks in this kind of network. One of the most popular kinds of such attacks is Denial of Service (DoS) [3]. This paper presents the steps of programming an application that can detect and mitigate DoS attack exploiting ICMP drop using the OpenDaylight controller.

There are several ideas developed below to mitigate DoS attacks in SDN. In the paper [3] B.H. Lawal and A.T. Nuray proposed an SDN application that collects information using sFlow software system, so that the controller handles the network decisions according to this software. In the case of adaptability, the controller must not depend on other network software to make decisions (or at least has minimal dependence), an exception to the OpenFlow switches. So, the idea is that the own controller can estimate the throughput of the traffic flows of the network. Another implementation is presented in [4] by R. Kandoi and M. Antikainen. They proposed a configuration and tuning of parameters of the rules such as the timeout value. This idea will help to defend against the DoS attack. But if the attack can vary or be carried out from different sources, the bandwidth can still be compromised and overwhelmed. Also, this solution may require to repeatedly request rules for previously known flows adding overflow in the communication between the controller and the switches.

We propose a solution based only on the use of controller capabilities. The basic idea is to track flows in the SDN network and estimate the throughput of each flow, installing rules that help make such estimation. If a flow is higher than a threshold, the application detects that and mitigates this possible DoS attack. The basic notions of the application and tests of its functionality are described in [5]. This paper focuses on the key aspects of programming the application and tests the refreshing time that it takes to estimate the throughputs of the flows and detect a possible DoS attack.

## Software-Defined Networking

As mentioned before, SDN is a networking architecture that makes the split of the control and switching planes [1]. The switches and routers must be extremely efficient at switching and must reduce their intelligence to a minimum. The management of the control plane is done in a device called controller (Fig. 1).

The controller executes software modules and bundles that establish the network’s functionality and assemble the rules that must be installed on them. The controller uses protocols (e.g. OpenFlow) to communicate with the switches.

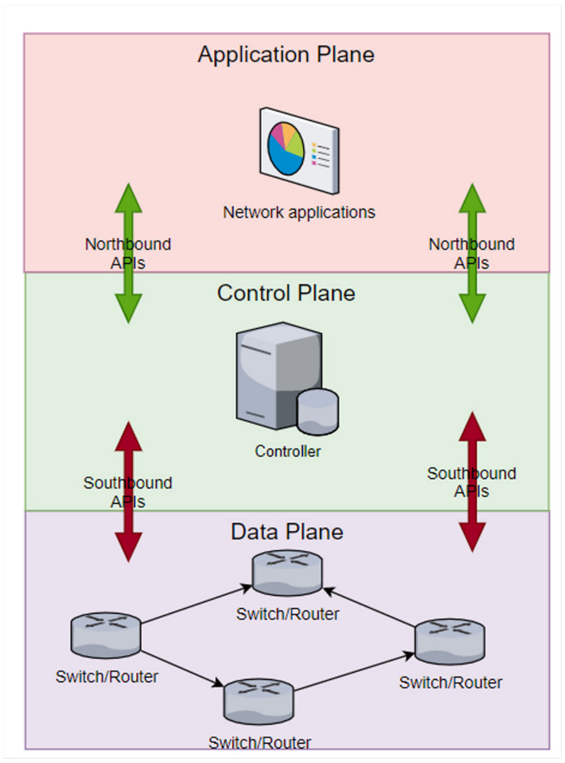


Fig. 1. SDN architecture

## OpenDaylight

OpenDaylight (ODL) is a modular open-source platform that customizes and automates networks of any size and scale [6]. One of the most common uses of ODL is software-defined networking. The main layers of the OpenDaylight architecture are the Controller Platform Layer and the Service Abstraction Layer (SAL) that are shown in Fig. 2. As shown in Fig. 2, southbound (SB) plugins communicate with network devices and northbound (NB) plugins allow communication with applications that use the controller.

Controller Platform Layer contains the modules that provide essential functionalities. The modules of this layer define the operation of a network according to which of they are selected by the controller. Our application to mitigate the DoS attack resides in this layer.

Model Driven Service Abstraction Layer (MD-SAL) uses the idea of data providers and data consumers in the modules. MD-SAL connects consumers to providers and supports data adaptation between them. This allows modules to communicate with each other without minding which protocols are being used by the controller and network devices [7]. OpenDaylight uses OGSI system architecture which uses the MD-SAL [8]. MD-SAL is a shared layer for the northbound and the southbound APIs and the data structures used in different modules and components of an SDN controller. MD-SAL achieves the communication between different plugins from different modules regardless of the layer (NB or SB) due to a common layer [9]. The data structures and the creation of plugins implemented in applications are modeled using Yang language [10]. The generation of the module API is carried out after the compilation of the Yang models. MD-SAL ensures the framework to support:

• Subscriptions to publish and listen notifications. Service that is generated in providers when data in the data store is changed.

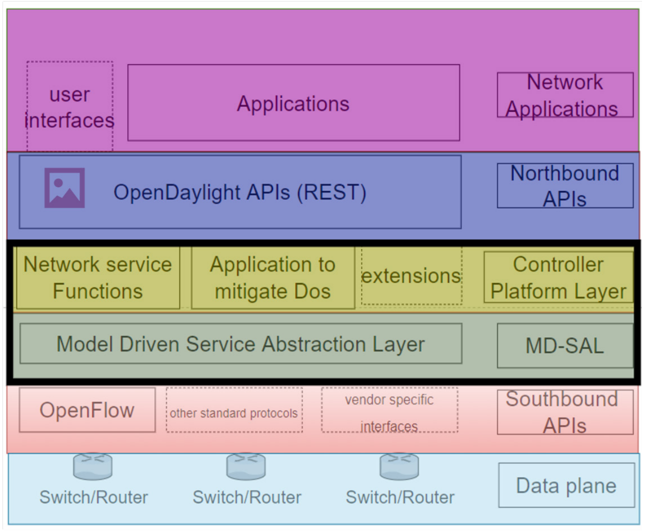


Fig. 2. OpenDaylight architecture

• Datastore. The providers and consumers use the data store of the MD-SAL to store data. This storing enables the exchange of data between providers and consumers. This data store can be split into clusters [11].

• Remote Procedure Call (RPC). It is a procedure that is used when a consumer receives notifications to get data from providers.

The proposed application to mitigate the DoS uses the DataBrokerService, so a simple Yang model was developed to interact with the data store using reading and writing transactions, and the following notification services:

• PacketProcessingService: used to process arriving packets at the controller.
• Ipv4PacketListener: used to process and decode the arriving IP packets at the controller.
• OpenDaylightInventoryListener: used to get related information about the OpenFlow switches.

## Programming the application

The proposed application is deployed as an ODL plugin that sends a set of rules to the OpenFlow switches [12] so that they forward every IP packet to the controller and the destination port. The installation of both proactive and reactive rules is carried out to monitoring the flows that are in the network. A rule in SDN is a primitive that establishes how a packet that ingress to the switch is handled. A rule has a key that is used as an identifier. Besides, the rule saves information about how many packets (or bytes) have matched the rule. So, with the information of the key and the number of bytes, it is possible to estimate the throughput of each flow at a certain time. The proactive rules are generated when the controller populates them in the flow table before any packets arrived. These rules are usually installed when the notification about the connection between the controller and the switches is established. The switches do not know the location of the hosts at the beginning of the communication. So it is necessary to install these proactive rules at the beginning to avoid the loss of the first packets. These proactive rules will make flooding considering the functionality of a switch. On the other hand, the reactive rules are installed when the first packets arrive to the switch. The firsts packets in the controller allow knowing the location of the hosts with their respective IP address and input port. Therefore, it is possible to install a reactive rule to make forwarding of data without generating flood and sending data only to the respective output port.

The following classes from the SDN Hub Project [13] were reused while implementing the application:

- **InventoryUtils:** This class possesses methods to decode the headers of the packets received at the controller. Also, this class allows getting values relative to the information of the packets such as Datapath ID, input ports and output ports.
- **GenericTransactionUtils:** This class has methods to interact with the data store using reading and writing transactions.

There were additional files added to the base project [13]. After that Maven was used [14]. To add services such as RPC, access to the datastore and subscription to notifications several pom.xml files were modified. Files added to the project:

- PacketParsingUtils.java: This file contains the classes with methods for extracting information from the packet headers such as the value of source or destination IP addresses.
- rules.java: In this file, the actions corresponding to establish the proactive and reactive rules are implemented.
- application.java: In this file, the proposed application to detect and mitigate DoS is deployed, besides, the rules to drop flows that are attacking the network.

**Key ideas to generate reactive and proactive rules.** Both proactive and reactive rules are generated according to [6]. The basis is to use the same ideas that both rules allow the forwarding of the data in the network considering the functionalities of the two-layer switch. Once the application starts, it registers itself to receive notifications. The *OnNodeUpdated* notification is invoked when every switch registers to the controller. This notification has information about the switch that is registered to the controller, e.g. ID of the switch, ports, etc. With the information of this notification, the *install_proactive_rules* method is invoked. The idea is to make flooding for, as discussed previously, hosts' identification in a network. The matching conditions of the proactive rules are established considering only input forwarding ports and the IP protocol. In the output actions the method *install_proactive_rules* takes each forwarding port as input and the other forwarding ports as output for every port of the switches in every switch, e.g. if the switch has five forwarding ports, the method *install_proactive_rules* will install five proactive rules.

The idea of reactive rules is to track the flows of the network and reduce the overhead that flooding implies. These rules are generated when the firsts IP packets arrive at the controller. The idea is to save the input port and the source IP address in a map called *IP_table*. When other packets arrive at the controller, the input port and the source IP address are matched for every element of the *IP_table*. If there is no coincidence, these elements are saved in the *IP_table*. Considering the destination IP address of the packet, this element is searched in the map *IP_table*. If the element is found, it is possible to know the output port and the reactive rule can be created. The matching fields will be the IP protocol, input port, source IP address, destination IP address. The output actions are to send the packet to the corresponding output port. It is necessary to consider that the reactive rules will have higher priority than the proactive rules.

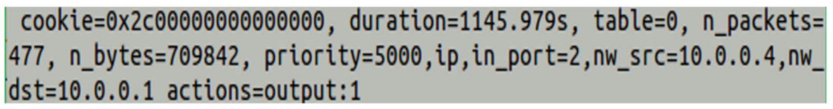
**Key ideas to detect and mitigate DoS attacks.** The idea to detect a DoS attack is considering the scenario where both proactive and reactive rules are installed in every switch. Once the application is initialized, an object *time* of the class *Timer* is created. The method *schedule* is configured in the object *time*. This method has the following prototype: *public void schedule*(*TimerTask task*, *long delay*, *long period*), where the argument *task* is the process to detect and mitigate DoS attack, the *delay* will be set to 0 and the *period* is initialized with the value of the variable *refreshing_time*. It means that this process will be executed every *refreshing_time* seconds. After, that object *data* is created. This object is an instance created by the method *Runtime.getRunTime().exec*(*"sudo ovs-ofctl dump-flows -OOpenFlow13 S1"*) execution. The object *data* allows writing the command *sudo ovs-ofctl dump-flows -OOpenFlow13 S1* in the controller. This command allows getting all the rules installed in the switch S1. Where:

*ovs-ofctl*: Command line to monitor OpenFlow switches

*dump-flows*: Print of all rules of the switch

*-OOpenFlow13*: Protocol OpenFlow 1.3

*S1*: Switch ID S1



```
 cookie=0x2c00000000000000, duration=1145.979s, table=0, n_packets=
477, n_bytes=709842, priority=5000,ip,in_port=2,nw_src=10.0.0.4,nw_
dst=10.0.0.1 actions=output:1
```

Fig. 3. Example of a rule

An example of the results of the following command is presented in Fig. 3.

Where:

*cookie*: Identifier of the rule

*duration*: Time that the rule has been installed

*table*: Identifier of the table where the rules are saved

*n_packets*: Number of packets that were matched the rule

*n_bytes*: Number of bytes that were matched the rule

*priority*: Value that shows the urgency to apply a rule. Higher values imply more priority

*ip*: Match field –IP protocol

*in_port*: Match field – input port

*nw_src*: Match field – source IP address

*nw_dst*: Match field – destination IP address

*actions*: Forwarding the packets to the output.

The idea is to save in every switch the values of the cookie to know if the rules belong to the developed application (reactive rules). The fields of *in_port*, *nw_src*, *nw_dst* are gotten from the object *data* and saved in an object called *identifier*. This object is useful to know the kind of flow. Besides, the value *n_bytes* can be gotten from the object *data*. This value is necessary to know the current number of bytes that matching the packets. The values of *identifier* and *n_bytes* are saved as a tuple. This process is done for every rule in the list of the application reactive rules in every switch. The *n_bytes* is updated according the expression *n_bytes* = *n_bytes* – *n_bytesi*, where *n_bytesi* is the number of bytes of the previously saved tuple.

Therefore, the value of *n_bytes/refreshing_time* is the throughput of each flow. All these values are saved in an array called *rules_number_bytes*. Then every element of the array is compared to the value of the variable *threshold* (value that can be set by the programmer). If the value is higher than the *threshold*, the method *install_drop_rules* is invoked.

**Key ideas to generate drop rules.** The basis of these rules is to drop the packets that belong to flows whose throughput is higher than the threshold. The matching fields of the drop rules are the input port, the source IP address, destination IP address and the IP protocol. The output actions are to drop the packets.

The flow diagram of the whole application is shown in Fig. 4.

## Experimental Results

The network shown in Fig. 5 was tested in Mininet emulator [15]. Functionality tests of the application are presented in [5]. The results provided in this section are the performance of the application while changing one of its critical value, the *refreshing_time*. The bandwidth of each link is 1 Mbps and the threshold was set to 500 Kbps. The idea is to change the refresh time in different scenarios to check performance values such as throughput, time to establish an optimal threshold that can detect a DoS attack. The attacker will be the host *00:00:00:00:04*, and it will attack the host *00:00:00:00:03*. The results of the values were calculated in the switch *openflow:3*.

As can be seen from the Table 1, the maximum value of the threshold is close to the value of the *refreshing_time*. This is explained by the fact the application checks the value of the throughput of each flow at each time interval, set by the value of *refreshing_time*. So, the maximum time to detect a DoS attack will be when the DoS attack begins as soon as the interval of *refreshing_time* begins.The throughput during the

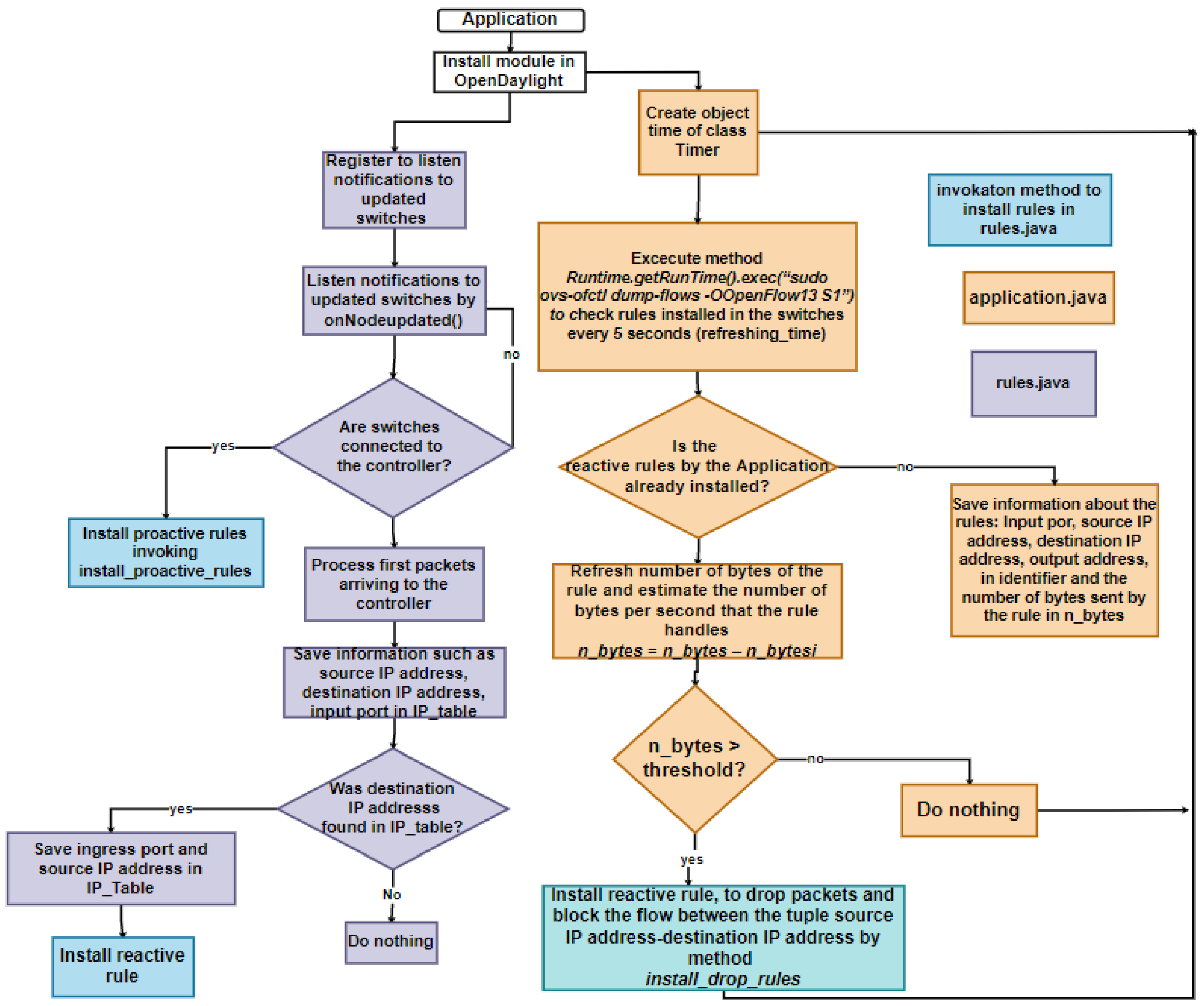

Application
Install module in OpenDaylight
Register to listen notifications to updated switches
Listen notifications to updated switches by onNodeupdated()
no
Are switches connected to the controller?
yes
Install proactive rules invoking install_proactive_rules
Process first packets arriving to the controller
Save information such as source IP address, destination IP address, input port in IP_table
Was destination IP addresss found in IP_table?
yes
Save ingress port and source IP address in IP_Table
Install reactive rule
No
Do nothing
Create object time of class Timer
Excecute method Runtime.getRunTime().exec("sudo ovs-ofctl dump-flows -OOpenFlow13 S1") to check rules installed in the switches every 5 seconds (refreshing_time)
Is the reactive rules by the Application already installed?
no
Save information about the rules: Input por, source IP address, destination IP address, output address, in identifier and the number of bytes sent by the rule in n_bytes
Refresh number of bytes of the rule and estimate the number of bytes per second that the rule handles n_bytes = n_bytes – n_bytesi
n_bytes > threshold?
no
Do nothing
yes
Install reactive rule, to drop packets and block the flow between the tuple source IP address-destination IP address by method install_drop_rules
invokaton method to install rules in rules.java
application.java
rules.java

Fig. 4. Flow diagram of the whole application

Table 1

**Performance changes owing to refreshing time variations**

| Refreshing time, s | Throughput during the DoS attack, Kbps | Average time of processing ICMP packets, ms | Average time of processing ICMP packets, ms |
|---|---|---|---|
| 1 | 900.6 | 405.5 | 1.8 |
| 2 | 910.5 | 276.1 | 2.4 |
| 5 | 925.4 | 268.4 | 4.8 |
| 8 | 940.3 | 211.1 | 7.8 |
| 15 | 967.5 | 181.1 | 14.7 |
| 30 | 980.4 | 130.0 | 29.6 |

DoS attack is quite uniform, there is not a high difference and it has sense because this value is the maximum capacity of each link of the network.

It is possible to see the difference in the average time of processing ICMP packets. This occurs because with the small values of the *refreshing_time* a complete saturation is achieved due to the DoS

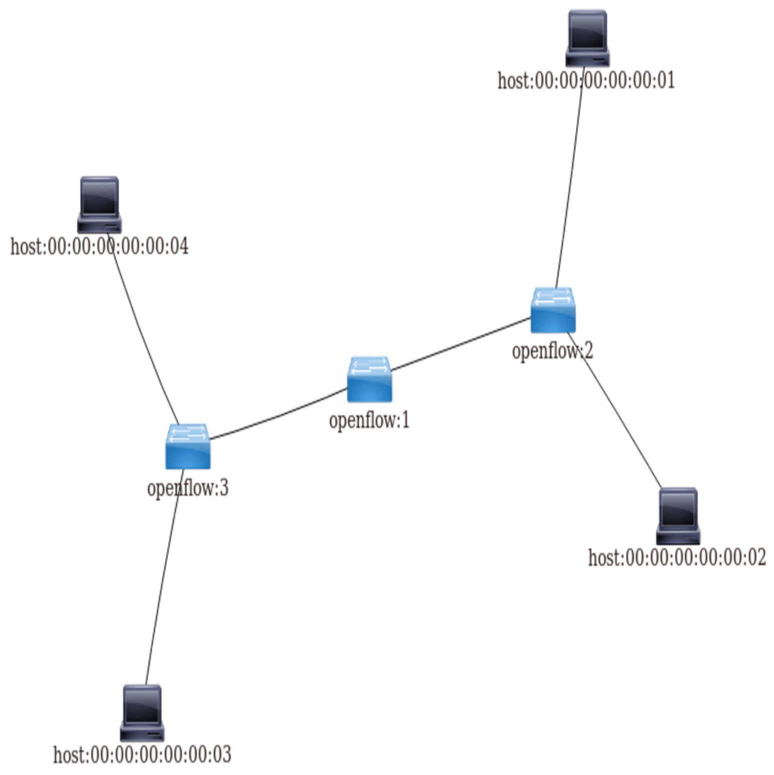


Fig. 5. Network topology tested in mininet

attack. At higher values of *refreshing_time*, due to some ICMP packets are discharged, the average time is reduced.

## Conclusion

In this article, the approach to program an application that can detect and mitigate the DoS flood attack has been proposed. The application was emulated on the SDN network using the OpenDaylight controller. The main ideas about the programming were to implement classes which describe and install proactive and reactive rules. These rules give the application an ability to function as a two-layer switch, track each flow and save statistics of each flow such as the number of bytes. This information will be used at each time interval to estimate the throughput and compare it to a threshold value to decide if there is a DoS attack. If an attack is detected the rules installation is performed to drop the flows whose throughputs are higher than the threshold. Tests to collect information about the performance changes due to the variation of refreshing time interval to detect a DoS attack were carried out.

## REFERENCES

1. **Kreutz D., Ramos F.M.V., Veríssimo P.E., Rothenberg C.E., Azodolmolky S., Uhlig S.** Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 2015, Vol. 103, No. 1, Pp. 14–76.

2. **Antikainen M., Aura T., Sarel M.** Attacking an SDN with a Compromised OpenFlow Switch. *Proceedings of the 19th Conference on Secure IT Systems*, 2014, Pp. 233–243

3. **Lawal B.H., Nuray A.T.** Real-time detection and mitigation of distributed denial of service (DDoS) attacks in software defined networking (SDN). *Proceedings of 26th Signal Processing and Communications Applications Conference*, 2018, Pp. 1–4.

4. **Kandoi R., Antikainen M.** Denial-of-Service Attacks in OpenFlow SDN Networks. *Proceedings of the International Symposium on Integrated Network Management* (*INM*), 2015, Pp. 1322–1326.

5. **Cajas C., Budanov D.** Mitigation of Denial of Services Attacks Using OpenDaylight Application in Software-Defined Networking. *Proceedings of the 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering* (*EIConRus 2021*), 2021, Pp. 260–265.

6. **Cajas C., Valdivieso C., Mejía D., Bernal I.** On programming an MP-TCP analyzer plugin using OpenDayLight Beryllium as the SDN controller. *Proceedings of the 2017 IEEE Second Ecuador Technical Chapters Meeting* (*ETCM*), 2017, Vol. 2, Issue 19, Pp. 1–6.

7. **Paliwal M., Shrimankar D., Tembhurne O.** Controllers in SDN: A Review Report. *IEEE Access*, 2018, Vol. 6, Pp. 36256–36270.

8. **Khattak Z. K., Awais M., Iqbal A.** Performance evaluation of OpenDaylight SDN controller. *Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems* (*ICPADS*), 2014, Pp. 671–676.

9. **Becerra F., Mejía D., Bernal I.** Solving MP-TCP's Shared Bottlenecks Using a SDN with OpenDayLight as the Controller. *Proceedings of the 2018 IEEE ANDESCON*, 2018, Pp. 1–6.

10. **Vilalta R., Via S., Mira F., Sanabria L., Martinez R., Casellas R., Munoz R., Alonso-Zarate J.** Control and management of a connected car using YANG/RESTCONF and cloud computing. *Proceedings of the 8th International Conference on the Network of the Future* (*NOF*), 2017, Pp. 120–122.

11. **Kim T., Myung J., Yoo S.** Load Balancing of Distributed Datastore in OpenDaylight Controller Cluster. *The IEEE Transactions on Network and Service Management*, 2019, Vol. 16, Issue 1, Pp. 72–83.

12. **Alsaeedi M., Mohamad M.M., Al-Roubaiey A.A.** Toward Adaptive and Scalable OpenFlow-SDN Flow Control. *IEEE Access*, 2019, Vol. 7, Pp. 107346–107379.

13. **Seetharaman S., Ramachandran A., Natarajan S.** SDNHub Opendaylight Tutorial, 2014.

14. **Xiong Z.-H., Yang Y.-Z.** Automatic updating method based on Maven. *Proceedings of the 9th International Conference on Computer Science & Education*, 2014, Pp. 1074–1077.

15. Mininet, Available: *http://mininet.org/* (Accessed: 07.03.2021).

*Received 10.03.2021.*

## СПИСОК ЛИТЕРАТУРЫ

1. **Kreutz D., Ramos F.M.V., Veríssimo P.E., Rothenberg C.E., Azodolmolky S., Uhlig S.** Software-Defined Networking: A Comprehensive Survey. Proceedings of the IEEE, 2015, Vol. 103, No. 1, Pp. 14–76.

2. **Antikainen M., Aura T., Sarel M.** Attacking an SDN with a Compromised OpenFlow Switch. Proceedings of the 19th Conference on Secure IT Systems, 2014, Pp. 233–243

3. **Lawal B.H., Nuray A.T.** Real-time detection and mitigation of distributed denial of service (DDoS) attacks in software defined networking (SDN). Proceedings of 26th Signal Processing and Communications Applications Conference, 2018, Pp. 1–4.

4. **Kandoi R., Antikainen M.** Denial-of-Service Attacks in OpenFlow SDN Networks. Proceedings of the International Symposium on Integrated Network Management (INM), 2015, Pp. 1322–1326.

5. **Cajas C., Budanov D.** Mitigation of Denial of Services Attacks Using OpenDaylight Application in Software-Defined Networking. Proceedings of the 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus 2021), 2021, Pp. 260–265.

6. **Cajas C., Valdivieso C., Mejía D., Bernal I.** On programming an MP-TCP analyzer plugin using OpenDayLight Beryllium as the SDN controller. Proceedings of the 2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM), 2017, Vol. 2, Issue 19, Pp. 1–6.

7. **Paliwal M., Shrimankar D., Tembhurne O.** Controllers in SDN: A Review Report. IEEE Access, 2018, Vol. 6, Pp. 36256–36270.

8. **Khattak Z.K., Awais M., Iqbal A.** Performance evaluation of OpenDaylight SDN controller. Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2014, Pp. 671–676.

9. **Becerra F., Mejía D., Bernal I.** Solving MP-TCP's Shared Bottlenecks Using a SDN with OpenDayLight as the Controller. Proceedings of the 2018 IEEE ANDESCON, 2018, Pp. 1–6.

10. **Vilalta R., Via S., Mira F., Sanabria L., Martinez R., Casellas R., Munoz R., Alonso-Zarate J.** Control and management of a connected car using YANG/RESTCONF and cloud computing. Proceedings of the 8th International Conference on the Network of the Future (NOF), 2017, Pp. 120–122.

11. **Kim T., Myung J., Yoo S.** Load Balancing of Distributed Datastore in OpenDaylight Controller Cluster. The IEEE Transactions on Network and Service Management, 2019, Vol. 16, Issue 1, Pp. 72–83.

12. **Alsaeedi M., Mohamad M.M., Al-Roubaiey A.A.** Toward Adaptive and Scalable OpenFlow-SDN Flow Control. IEEE Access, 2019, Vol. 7, Pp. 107346–107379.

13. **Seetharaman S., Ramachandran A., Natarajan S.** SDNHub Opendaylight Tutorial, 2014.

14. **Xiong Z.-H., Yang Y.-Z.** Automatic updating method based on Maven. Proceedings of the 9th International Conference on Computer Science & Education, 2014, Pp. 1074–1077.

15. Mininet [электронный ресурс] URL: http://mininet.org/ (дата обращения: 07.03.2021).

*Статья поступила в редакцию 10.03.2021.*

## THE AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

**Кахас Гихарро Карлос Давид**
**Cajas Guijarro C.D**
E-mail: cajasgcarlos@hotmail.com

**Буданов Дмитрий Олегович**
**Budanov Dmitry O.**
E-mail: budanov_do@spbstu.ru