„Zeszyty Naukowe
Uczelni Jana Wyżykowskiego.
Studia z Nauk Technicznych”
2017 (6), s. 171–182.

Nevyan Neykov
University of Economics Varna, Bulgaria

Real-time detection and mitigation of flood attacks in SDN networksNevyan Neykov

Summary: Distributed Denial of Service (DDoS) flooding attack threats are becoming more and more relevant due to the advances in the Software Defined Networks (SDN). This rising trend creates an emerging need for defense mechanisms against such attacks. In order to address those issues the following paper focuses primarily on the implementation of an automatic real-time DDoS defense application based on sFlow technology. Initially we start by constructing a special flow, bound to a metric in order to capture traffic of interest. As soon as the flow reaches a certain predefined metric level, it is sent to an analyzer. Next we implement a detection algorithm based on the event handling capabilities of the sFlow-RT real-time analyser. Finally, the algorithm is tested with emulation network Mininet using network traffic, resulting in quick and effective DDoS attack mitigation.

Keywords: SDN; OpenFlow; OpenVSwitch; Mininet; sFlow; DDoS; flood attack; OpenStack

Wykrywanie w czasie rzeczywistym I łniwelowanie masowych ataków w sieciach SDN

Streszczenie: Zagrożenia w wyniku rozproszonej odmowy usługi (DDos) w przypadku masowego ataku stają się coraz bardziej możliwe z uwagi na rozwój programowalnych sieci (SDN). Ten rosnący trend powoduje konieczność tworzenia mechanizmów obronnych na wypadek takich ataków. W celu odniesienia sie do takich zagadnień, artykuł ten przede wszystkim skupia się na wdrażaniu automatycznych, działających w czasie rzeczywistym aplikacji obronnych DDoS, opartych na technologii sFlow. Wstępnie rozpoczynamy od stworzenia określonego przepływu, związanego z pomiarem w celu wychwycenia jak duże jest zainteresowanie eksploracją danych. Jak tylko przepływ osiągnie pewien wcześniej określony poziom, informacja zostaje wysłana do analityka. Następnie wdrażamy algorytm wykrywania, w oparciu o zdarzenie, który posiada funkcje analityka sFlow dzialajacego w czasie rzeczywistym. Na koniec testuje się algorytm przy wykorzystaniu emulacyjnej sieci Mininet, wykorzystującej eksplorację danych, co w rezultacie szybko i w efektywny sposób niweluje masowy atak DDoS.

Słowa kluczowe: SDN, OpenFlow, OpenVSwitch, Mininet, sFlow, DDoS; atak masowy,

1. Introduction

DDoS attacks are becoming more and more common nowadays. They are hard to block because of their distributed nature. With the advance of Software Defined Networks (SDN) the applications operating within cloud environments demand better protection against such types of attacks. OpenFlow is widely supported protocol which introduces flexible programming of network applications inside SDN. The protocol enables controller applications to specify a set of criteria for packet matching and perform actions to achieve network traffic engineering. From the application standpoint, OpenVSwitch (OVS) is an open-source switching software that uses OpenFlow, and is supported also by cloud platforms such as OpenStack, Mininet as virtual network emulation software and sFlow-RT for flow monitoring. This paper details a practical approach for real-time analysis, detection and mitigation of DDoS attacks using collaboration between the mentioned set of tools.

2. Literature review

Researchers[1] suggest a pure blocking mechanism for mitigation of DDoS attacks. The proposed algorithm incorporates features such as temporary increase of hosts' provided network bandwidth, time limiting the number of sent packets from a single IP address based on their unique IP time-stamp, as well as usage of echo request packets informing the source address to slow down the network transfer rate. The concept of privileged and unprivileged traffic channels, is also being introduced [2] with the aim to establish undisturbed communication between network hosts using privileged packets. The proposed lightweight solution creates a special privileged channel by using two additional negotiation packets between the sending and receiving hosts for each network round trip. In this scenario routers are being additionally programmed to give preferential treatment to the privileged packets. Being in privileged mode, the receiver sends capacity updates to the sender within a certain window of time, allowing for a possible switch between privileged and unprivileged channels. This way, by halting certain flows, the receiver controls sender capabilities, and could effectively limit them. In case of an unprivileged attack, both parties switch into privileged mode and filter out the attack by dropping its packets.

Moreover the aim and inner workings of an elaborate DDoS attack are revealed in [3] where botnets target pre-selected link routes, which deliver the vast majority of traffic to a specified area. Next botnets flood groups of routers with internal packets, without using the unwanted outside of the AS (autonomous system) traffic. Simultaneously the attacker monitors and reacts to network route changes by updating the route map of the next attack target servers. The successful launch of the attack depends on several conditions such as:

- keeping a very low rate of adversary flow-rates, to avoid triggering of the network protection mechanisms;
- the attack flow rate should also be higher than the target bandwidth divided by the maximum number of available flows on the link;
- per-flow rate should be less than the maximum flow rate which could be handled without triggering an alarm;
- even distribution of the aggregate attack traffic to multiple hosts, which are carefully chosen to be not too close (in less than 3 hops) to the target area.

Although DDoS attacks are inherent to the pure data-plane and impose tracking difficulties to the SDN controlling plane, some authors suggest traffic engineering mitigation techniques such as traffic re-routing. In this regard papers [4] and [5] reveal the inherent goal of Crossfire attack to cut off internet connectivity towards a specific geographic area (target area). Both papers propose solutions for SDN defense against the attack by rate-limiting malicious sources while keeping the network running, without causing any congestion. The proposed process moves through phases such as: monitoring, traffic load balancing by rerouting, recording DDoS link sources, detection and blockage of suspicious sources). Sources are assumed as suspicious, when, after being pushed away, they return to communicate via the same routes, and are considered as block candidates. In contrast, in case of route change the normal sources do not readjust themselves to use the same popular routes as before. Additionally to tame such elaborate attacks the authors [4] propose several stages of malicious flows detection such as: icmp packets and congested link monitoring where the detection routine tracks a congested link, isolates its source and destination addresses and changes its current route (route-mutation).

In the current paper we focus our research on the DDoS SYN flood attacks within the scope of SDN networks. We start by exploring mitigation mechanism differences between SDN and traditional networks in order to create a fully functional simulated DDoS network attack testbed.

Next we propose, implement and test an SDN defense mechanism against selected DdoS attacks.

During the preliminary research process in order to cover more-elaborated DDoS attacks such as CrossFire, we have also explored the potential integration of BGP protocol together with our proposed mitigation mechanism. While at time of writing technical limitations exist when integrating Quagga BGP routing protocol with MiniNet virtual networks in particular when using OpenVSwitch virtual routers, we foresee performing future tests using free simulation testbeds such as Cumulus VX .

3. Background

Control and data planes

Usually in SDN two planes exist: control – operated by the controller and data operated by switching and routing devices (fig.1). The controller is responsible for packets' logical management and can be programmed, while routers and switches are used for forwarding purposes only. Those features allow decoupling of the running network traffic. OpenFlow is one of the first SDN standards and by definition [13] is a programmable network, protocol is designed to manage traffic among both physical and virtual routers and switches. OpenFlow is also open-source and enables the controller to directly interact with switches and routers via flows.

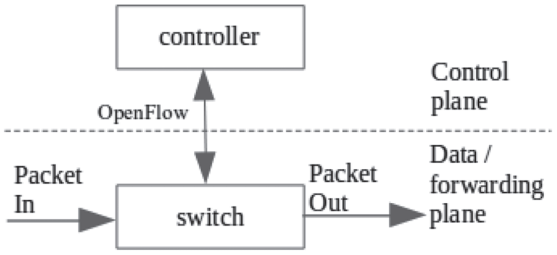

Fig. 1. Planes and communication

B. Packets, flows and tables

In order to perform packet forwarding, switches maintain flow tables consisting of flow entries. A flow record summarizes a set of packets that share common attributes. Each flow entry consists of match fields (the packet's incoming interface port, source and destination ports, ip and MAC addresses, type of networking protocol) statistics (cookie information tracking the flow, duration, what table the flow has to go to next, the size of the flow, counter usage log e.g. when packets match the flow rules this counter is increased, priority, idle and hard timeouts), as well as actions applied to the flow.

When a switch receives its first data packet from a host, it performs a lookup operation at its flow table, using the packet's header characteristics. In case there is no matching entry, the packet flow is forwarded via OpenFlow protocol to the controller for a decision on what action is to be applied further, and is returned to the data path. Actions can be forwarded to another port, controller or drop (fig.2).

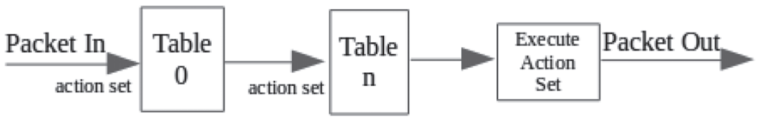

Fig. 2. Packets, flows and tables

4. Tools

The following is a quick summary of the main tools used in the lab setup.

A. OpenVSwitch

OpenVSwitch is OpenFlow supporting open-source virtual switch, designed to enable network automation through programmatic extension while supporting standard interfaces and protocols [14]. It is specifically tailored to function in virtualized environments. In cloud infrastructure systems such as OpenStack, hypervisors have the ability to bridge traffic between virtual machines and the outside world. In single hosted Linux environments this function is carried by the built-in bridge, which is fast and reliable, while OpenVSwitch targets multi-server virtualization deployments. OpenVSwitch support s both configuring and migrating configuration and network states between instances, as well as allowing network control systems such as NetFlow, IPFIX, and sFlow to respond and adapt to environment changes. Connections between various hosts and the switch are realized through ports (s1-eth1, s1-eth2, s1-eth3), where s1 is the name of the switch, and ‚s1-ethX' are the network interfaces. Ports connect the host network interface to the switch via the virtual bridge (fig.3).

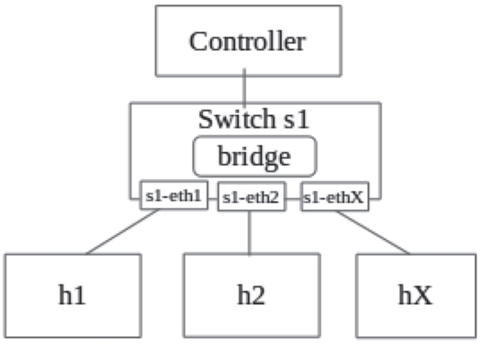

Fig. 3. OpenVSwitch architecture

B. Mininet

MiniNet is a network emulator which can create virtual kernel or user-space switches, controllers and hosts that are able to communicate over the emulated network via virtual link pairs [15]. MiniNet supports OpenFlow protocol and has built in OpenVSwitch controller. Emulated networks run real Linux network applications and provide Linux kernel and networking stack for development purposes. Because of its inherent features, the code, which runs on Mininet, could with minimal changes be transferred directly to hardware switches.

C. sFlow

sFlow is a technology for traffic monitoring of virtual and real networks[8], and defines traffic sampling mechanisms, implemented in sFlow agents. Agents are attached to network switches and carry traffic measurement data to a common sFlow Collector. During the process of data gathering, a stream of packet headers is sampled and decoded to extract fields of interest, later forming a flow record cache. Then each incoming flow record is looked up inside the flow cache in order to be updated or saved. Based on protocol information such as FIN flag, a timeout, inactivity, or when the cache is full, records are flushed from the cache and sent to traffic analysis application. The monitor, decode, hash, flow cache and flush functionality are moved out of the switch to a central sFlow analyzer (collector), which can run on a server with relatively abundant resources, thus safely supporting large numbers of requests without a risk of network destabilization.

D. Sflow-RT

sFlow-RT[11][12] is an open-source product that has an embedded OpenFlow controller, allowing monitoring and flows insertions to OpenFlow supporting switches. It can listen on certain events of interest, raise triggers and apply traffic handling rules to a particular controller. Event handling in sFlow consists of three steps:

1. *Set up a flow to capture traffic parameters of interest.*
2. *Create a metric to summarize a flow, coupled with a threshold.*
3. *Raise an event and attach an event handler, when a certain threshold level is reached.*

E. Integrated hybrid OpenFlow[12]

In order to integrate our testbed with Mininet, we have to clarify that Mininet flow forwarding rules are internally working with two tables. Table 0 has a rule to look up rules from table 1, which implement equal-cost multipath (ECMP) forwarding [18]. ECMP forwarding forms when a routing table contains multiple next-hop addresses for one destination with same preference and metric values.

By default, all traffic is handled by the switch's normal switching and routing functions without any intervention from the controller. OpenFlow rules are then used to override the normal forwarding behavior for the selected flow. This is achieved by adding high priority rules to table 0. The current setup via the file (leafandspine-hybrid.js) emulates the hybrid OpenFlow NORMAL action by rewriting it to jump to table 1 that contains the ECMP forwarding rules. Such an approach is extremely scalable and robust with a minimal overhead associated with maintaining OpenFlow connections between the controller and the switches. Moreover the network will still continue to forward traffic if the controller fails. Because the switches have fully populated forwarding tables, packet_in events are never sent to the controller.

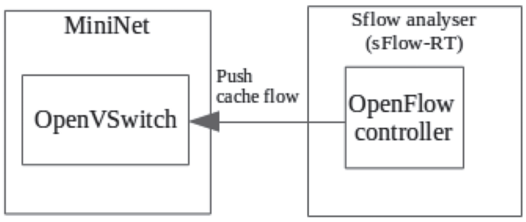

Fig. 4. Traffic monitoring

5. Implementation

The software used in the experimental setup is carefully chosen, based on the support of the OpenVSwitch controller. For the networking testbed we will be using Mininet installed on a VirtualBox[16] emulated machine running Ubuntu Linux[6] together with the real-time network analyzer sFlow-RT.

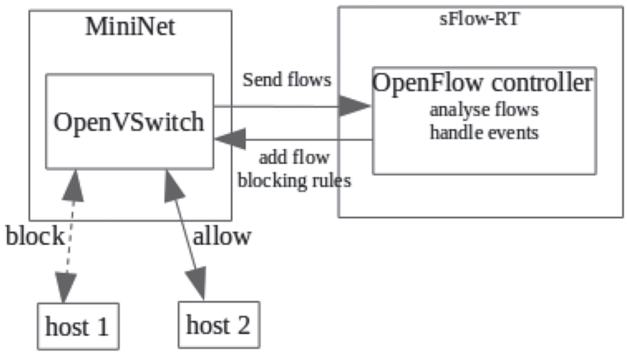

Fig. 5. Event handling

In order for sFlow-RT to analyze and react to traffic changes, it has to be configured to work together with the existing network (fig.4, fig.5). This is done by installing sampling agent(s) and attaching them to the switches we would like to monitor. Then we create a logical bridge port (sFlow) which copies and sends sFlow traffic from a particular switch (s1), sampled by sFlow agent/s (eth0) to sFlow collector (127.0.0.1). On OpenFlow supporting switches this can be achieved running the following commands from the s1 OpenVSwitch:

- to connect OpenVSwitch to OpenFlow controller: `ovs-vsctl set-controller s1 tcp:10.0.0.1:6633`
- to connect OpenVSwitch to sFlow-RT analyzer : `sudo ovs-vsctl -\- --id=@sFlow create sFlow agent=eth0 target=\"127.0.0.1:6343\" sampling=2 polling=20 -\- -\- set bridge s1 sFlow=@sFlow`

Next we do additional refinements to the setup, due to the fact that MiniNet is a virtual networking environment emulator. Initially the virtual network topology has to be exported in order to be understood by sFlow. For this reason we install an Apache server to provide a directory /var/www/html that can be remotely retrieved via HTTP protocol by sFlow: `sudo apt-get install apache2`

Then we run Mininet, specify the address of collector and controller, as well as the type and size of the custom network we are creating and at the same time we export the network topology as JSON[12] file to the sFlow collector

```
   sudo ./leafandspine.py –-collector=127.0.0.1 --controller=127.0.0.1
--leaf=1 --spine=1 --fanout 3
    --topofile=/var/www/html/topology.json
```

(note: the addresses of collector/s and controller can differ)

In the script provided by sFlow-RT the number of leaf and spine switches, as well as the number of hosts per leaf can be modified. In leaf and spine architecture (fig.6) all the switches are interconnected within the network fabric. This minimizes latency and bottlenecks because each packet has to only travel to spine and another leaf switch to reach its endpoint.

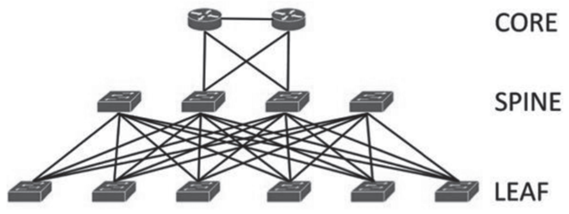

Fig. 6. Leaf and spine topology[1]

Next we start the real-time sFlow analyzer to monitor the collected traffic data samples: ./sFlow-rt/start.sh and periodically check whether Mininet topology is loaded into sFlow-RT http://localhost:8008/topology/json, and if not we push it using Curl: curl -H "Content-Type: application/json" -X PUT --data @/var/www/html/topology.json http://localhost:8008/topology/json

DDoS flood attack

In this type of network attack (fig.7), the attacker uses a command and control network to instruct large numbers of compromised systems to send traffic to a designated target aiming to overwhelm the target infrastructure and deny access to its legitimate users.

[1] http://searchdatacenter.techtarget.com/definition/Leaf-spine leaf-spine architecture

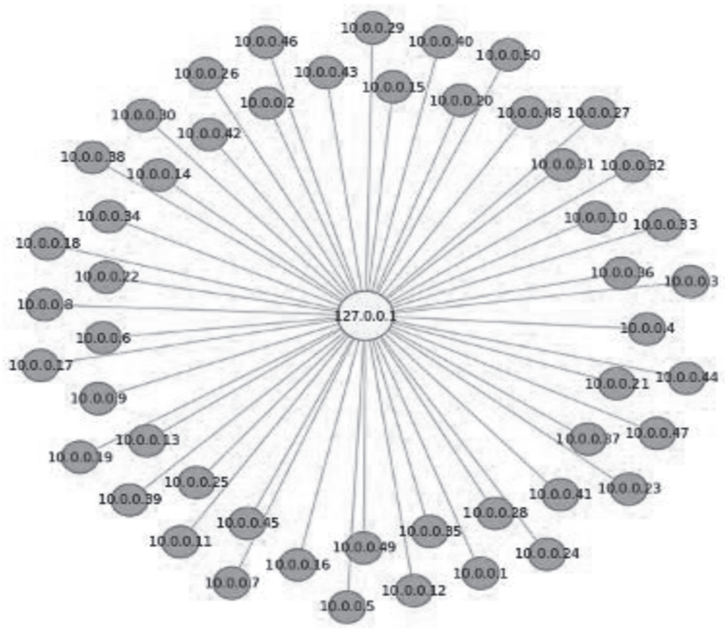

Fig. 7. Visualization of the simulated network attack topology using HP VAN SDN controller [19]

An ICMP DDoS flood attack can be realized using the command: ping -f 10.0.0.2 ran from several MiniNet hosts. The following is a sample output from the sFlow analyzer on a network under attack:

```
   ./start.sh
INFO: Listening, OpenFlow port 6633
INFO: Listening, sFlow port 6343
INFO: OF: connected to 127.0.0.1:59165 using OF 1.3
INFO: OF: connected to 127.0.0.1:59168 using OF 1.3
INFO: OF: connected to 127.0.0.1:59167 using OF 1.3
INFO: OF: connected to 127.0.0.1:59166 using OF 1.3
INFO: OF1.3: 127.0.0.1:59168 = datapath 0000000000000001
INFO: OF1.3: datapath 0000000000000001 added to
INFO: Starting the Jetty [HTTP/1.1] server on port INFO: Star-
ting com.sFlow.rt.rest.sFlowApplication application
INFO: Listening, http://localhost:8008
INFO: mark.js started
INFO: Lead and spine hybrid mode enabled
INFO: [“11”,”2048”,”000400000002”,”000400000001”,”1”,”10.0.0.2”,
”10.0.0.1”]
```

```
   INFO: blocking {"priority":500,"idleTimeout":20,"hardTi-
meout":3600,"match":{"in_port":"4","dl_type":2048,"ip_proto-
":17,"nw_src":"10.0.0.1"},"actions":[]}
INFO:["9","2048","000400000001","000400000002","1","10.0.0.1","1
0.0.0.2"]
INFO: blocking {"priority":500,"idleTimeout":20,"hardTime-
out":3600,"match":{"in_port":"3","dl_type":2048,"ip_proto-
":17,"nw_src":"10.0.0.2"},"actions":[]}
```

From the log it can be seen that the sFlow controller starts and listens to OpenFlow and sFlow flows. Ports and data paths (addresses of the OpenVSwitch) are properly recognized and added to sFlow. Then the monitoring server on http://localhost/8008 and our DDoS detection and mitigation script (mark.js) are started. In case of a custom SYN flood event from the log, we see how the script mark.js adds to the controller a new blocking flow (i.e. rule without action) for the particular traffic source.

When performing tests with simulated SYN flood traffic, averaging 48 Mbits/sec., Fig. 8 shows that in a matter of 2-3 seconds the system blocks the traffic due to its real-time proactive response.

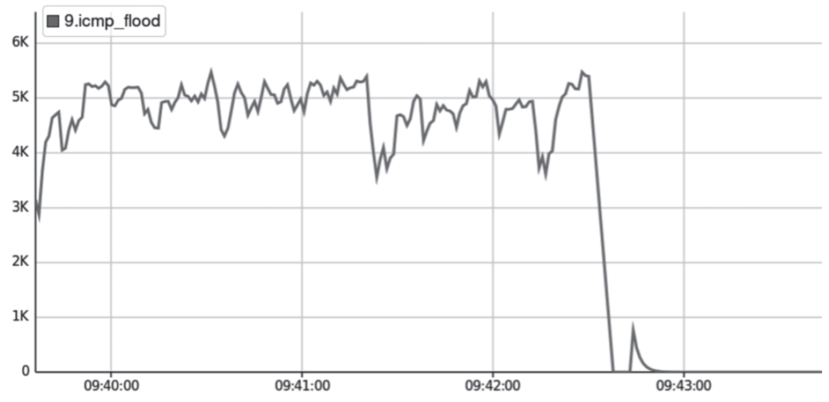

Fig. 8. Real-time monitoring of network interface traffic under attack

Conclusions and future work

Advances in software defined networks (SDN) provide researchers with access to abundant information about hosts and the surrounding routing network topologies. At the same time, when compared to conventional networks, explorations of complex network attacks within SDN environments built entirely of virtual appliances create major technical challenges.

The motivation behind the current paper was to develop and validate practical real-time protection against a particular DDoS syn flood attack within a simulated SDN environment. We demonstrate a simulated SYN flood attack (seen in the generated output chart), as well as the point where the controller uses our detecting script to proactively filter out the SYN flood type of traffic, effectively proving that when a new attack is being launched, it could be immediately detected and filtered. As a result, the suggested approach protects the link channel, allowing normal traffic to reach the network.

The paper also presents novelties in the development approach of flow processing and filtering mechanisms such as the usage of sFlow-RT embedded OpenFlow handling capabilities, without the help of an additional SDN controller such as Floodlight.

The testbed setup uses one switch to guard one link, but the proposed solution can be easily scaled out and applied to any OpenVSwitch installation including those inherent to OpenStack Neutron in order to provide an automated solution for effective cloud based flood attacks mitigation. The current work could be further expanded with respect to advanced DDoS attack mitigation techniques that target Crossfire types of attacks. Approaches in this direction include sending blocking messages to neighboring BGP routers, as well as with the help of SDN Traceroute project [20], to trace and re-route congested traffic of interest, and block re-occurring persistent route attempts that occur within short periods of time.

Bibliography

[1] S. Singh, R. A. Khan and A. Agrawal, „Prevention mechanism for infrastructure based Denial-of-Service attack over software Defined Network," *International Conference on Computing, Communication & Automation*, Noida, 2015, pp. 348-353.

[2] A. Yaar, A. Perrig and D. Song, "SIFF: a stateless Internet flow filter to mitigate DDoS flooding attacks," *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, 2004, pp. 130-143.

[3] M. S. Kang, S. B. Lee and V. D. Gligor, "The Crossfire Attack," *2013 IEEE Symposium on Security and Privacy*, Berkeley, CA, 2013, pp. 127-141.

[4] A. Aydeger, N. Saputro, K. Akkaya and M. Rahman, "Mitigating Crossfire Attacks Using SDN-Based Moving Target Defense," *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, Dubai, 2016, pp. 627-630.

[5] Gkounis, D., Kotronis, V. and Dimitropoulos, X., 2014. Towards defeating the crossfire attack using SDN. arXiv preprint arXiv:1412.2013.

[6] Helmke, Matthew. Ubuntu Unleashed 2016 Edition: Covering 15.10 and 16.04. Sams publishing, 2016.

[7] Jakub, Libosvar and Rodolfo, Alonso. "Tired of iptables based security groups? Here's how to gain tremendous speed with OpenVSwitch instead!", URL: https://www.openstack.org/assets/presentation-media/Austin-Summit-SG-firewall-Presentation-v2.3.pdf, 2016.

[8] Phaal, Peter, and Ben Pfaff. "sFlow OpenFlow Structures" Specification. sFlow. Org, 2014.

[9] Nugraha, Muhammad, Paramita Isyana, Musa Ardiansyah, Choi Deokjai, Cho Buseung. "Utilizing OpenFlow and sFlow to Detect and Mitigate SYN Flooding Attack." Journal of Korea Multimedia Society 17.8, pp. 988-994, 2014.

[10] McKeown, Nick, et al. "OpenFlow: enabling innovation in campus networks." ACM SIGCOMM Computer Communication Review 38.2, pp. 69-74, 2008.

[11] Phaal, Peter, and Marc Lavine. "sflow version 5." Specification. sFlow. Org, 2004.

[12] Phaal, Peter, "Mininet integrated hybrid OpenFlow testbed" URL: http://blog.sflow.com/2014/04/mininet-integrated-hybrid-openflow.html, 2017.

[13] OpenFlow Switch SpecificationVersion 1.3.1, URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf, 2012.

[14] OVS-ofctl OpenFlow Switch Management Commands, http://OpenVSwitch.org/support/dist-docs/ovs-ofctl.8.txt , 2016.

[15] Team, Mininet. "Mininet Overview." URL: http://mininet. org/overview/, 2017.

[16] Oracle, V. M. "VirtualBox." User Manual. URL: https://www.virtualbox.org/manual/UserManual.html, 2017.

[17] JSON (JavaScript Object Notation) data-interchange standard. URL: http://www.json.org/, 2017.

[18] ECMP URL: https://en.wikipedia.org/wiki/Equal-cost_multi-path_routing , 2017.

[19] HP VAN SDN controller trial version, URL: https://marketplace.saas.hpe.com/sdn/content/sdn-controller-free-trial, 2017.

[20] Wang, Y., Bi, J. and Zhang, K., 2017. A tool for tracing network data plane via SDN/OpenFlow. Science China Information Sciences, 60(2), p.022304.