Journal of Korea Multimedia Society Vol. 18, No. 2, February 2015(pp. 189-198)
http://dx.doi.org/10.9717/kmms.2015.18.2.189

Large Flows Detection, Marking, and Mitigation based on sFlow Standard in SDN

Muhammad Afaq[†], Shafqat Rehman[††], Wang-Cheol Song[†††]

ABSTRACT

Despite the fact that traffic engineering techniques have been comprehensively utilized in the past to enhance the performance of communication networks, the distinctive characteristics of Software Defined Networking (SDN) demand new traffic engineering techniques for better traffic control and management. Considering the behavior of traffic, large flows normally carry out transfers of large blocks of data and are naturally packet latency insensitive. However, small flows are often latency-sensitive. Without intelligent traffic engineering, these small flows may be blocked in the same queue behind megabytes of file transfer traffic. So it is very important to identify large flows for different applications. In the scope of this paper, we present an approach to detect large flows in real-time without even a short delay. After the detection of large flows, the next problem is how to control these large flows effectively and prevent network jam. In order to address this issue, we propose an approach in which when the controller is enabled, the large flow is mitigated the moment it hits the predefined threshold value in the control application. This real-time detection, marking, and controlling of large flows will assure an optimize usage of an overall network.

Key words: Large Flows, Small Flows, sFlow, OpenFlow, Priority Marking, SDN, DDoS

1. INTRODUCTION

Studies [1] have shown that in data networks the majority of flows tend to be short, whereas the majority of packets belong to a few long-lived large flows. The short flows (small flows) are usually referred to latency-sensitive, bursty applications, such as VoIP and search results, whereas the long-lived flows (large flows) are often large transfers like back-end operations or backups.

The small and large flows phenomenon has been treated as an issue for network performance. Network resources are utilized depending on the constraints and requirements of a particular application. Large flows have the tendency to fill network buffers end-to-end and bring in substantial delay to the latency-sensitive small flows that in fact share the same buffers with large flows. This gives rise to the degradation of network performance. Furthermore, currently used hash-based multi-path routing techniques used in data networks can possibly hash various large flows onto one same link, whereas leaving other links free and causing lower quality network usage [2].

Therefore, it will be much suitable to deal with large flows differently than small flows. In order to this, detection, marking, and signaling of existence of large flows is required. Then with SDN, a traffic engineering module at the controller level can be informed to route large flows properly.

※ Corresponding Author : Wang-Cheol Song, Address: (690-756) 102 Jejudaehak-ro, Jeju, Republic of Korea, 690-756, TEL : +82-64-754-3656, FAX : +82-64-755-3620, E-mail : philo@jejunu.ac.kr
Receipt date : Dec. 30, 2014, Revision date : Feb. 11, 2015, Approval date : Feb. 16, 2015,

[†] Dept. of Computer Eng., Jeju National University (E-mail : afaq24@gmail.com)
[††] Dept. of Comp. Sci. & Eng., Air University, Islamabad, Pakistan (E-mail : shafqat.rehman@gmail.com)
[†††] Dept. of Computer Eng., Jeju National University (E-mail : kingiron@gmail.com)
※ This research was supported by the 2014 scientific promotion program funded by Jeju National University

Generally, the detection of large flows can be attained by means of periodically polling, such as Hedera [3]. The Hedara technique makes use of the five-second polling period. This degree of granularity gives rise to probable network congestion between polls. Since existing fast data networks are equipped with 10Gbps or even faster links, the possibility of dropping many packets between polling intervals because of late detection of large flows is too high. Also, there is a possibility that a short-lived large flow will stay in an unwanted path during its entire existence.

In this paper, we achieve the detection and marking of large flows in SDN systems using sampling technology, sFlow [4]. SDN systems make use of OpenFlow and sFlow enabled switches with an advanced software-based centralized controller [5] which enables network engineers to examine, predict, and regulate the behavior of the transmitted data. sFlow based sampling technology requires these OpenFlow switches to send samples of all flows to traffic analysis tools, such as sFlowTrend, sFlow-RT [6], Ganglia etc., which then determines the existence of large flows based on the samples.

In our proposed approach, SDN control application detects large flow and configures the virtual switch to mark its packets. Basically, the sFlow-RT controller receives a stream of sFlow measurements from the virtual switch and rapidly detects large flow in real-time and notifies the control application. The control application, by means of OpenFlow controller, instantly sets up an OpenFlow rule that matches the flow and directing the switch to mark the flow by assigning the IP type of service bits. By doing so, the traffic upstream of the switch will contain large flow being identified and marked, while small flows will be left unmarked.

The remainder of the paper is organized as follows. An overview of sFlow technologies in relation with flow detection and marking is provided in Section 2 and Section 3. In Section 4, the system model is presented. In Section 5 and Section 6, large flows detection and marking, and large flows mitigation achieved from the proposed approach are explained respectively. The paper is concluded in Section 7.

2. FLOW DETECTION BASED ON sFLOW STANDARD

In order to communicate with the sFlow analytics engine, such as sFlow-RT [6], switches are configured to use sFlow protocol in the control plane. In addition to that, switches are also configured to use OpenFlow protocol to communicate with OpenFlow controller like OpenDaylight [7] or Floodlight [8] in the control plane. Control plane software like OpenFlow controller and sFlow use Open Northbound APIs to offer control functionality and summary statistics to SDN applications, e.g. DDoS Mitigation, Load Balancer, and Large Flow Marking etc.

As shown in Fig. 1, the OpenFlow protocol enables SDN controller to interact with forwarding plane and make adjustments to the network, such as gathering information of a network of switches and configure the forwarding trend of these switches. A graph based model of the network and an advanced routing algorithm to determine the flows path through the network is built by SDN

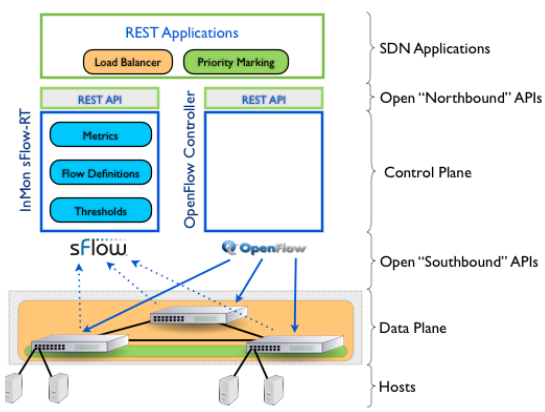


Fig. 1. Software Defined Networking (SDN) Stack.

controller. The OpenFlow protocol then adds the flow paths determined by the controller to the forwarding tables of the switches [8]. The sFlow standard is implemented in the switches/routers using a separate Application Specific IC (ASIC) which enables to continuously monitor application level traffic flows at wire speed on all interfaces simultaneously. sFlow and OpenFlow together provide an integrated flow monitoring system in which the OpenFlow controller can be used to define the flows which are to be monitored by sFlow. Furthermore, metrics from sFlow can be used as feedback by an SDN application to control the forwarding behavior in the switches.

The sFlow Agent is a software process that runs as part of the network management software within a device as shown in Fig. 2. It couples flow samples and interface counters into sFlow datagrams that are sent through the network to a sFlow collector. Sampling of packets is in particular carried out by the switching/routing ASICS, giving wire-speed performance. The condition of the forwarding/routing table entries attached with each sample packet is also recorded [9].

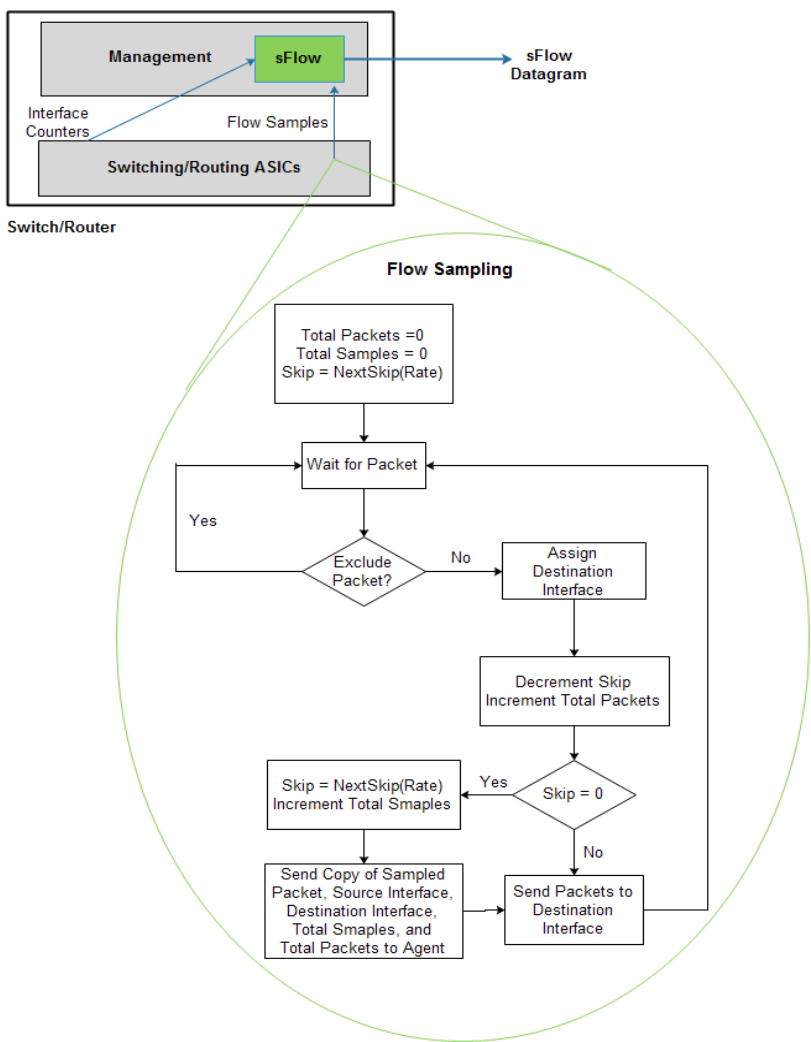


Fig. 2. sFlow Agent Embedded in Switch/Router.

3. sFLOW SAMPLING TECHNOLOGY

From wide range of devices, such as physical switches, virtual switches (OVSes), hosts, etc., traffic samples can be collected using sFlow. It is possible to configure sFlow monitoring on all interfaces of the device with little overhead. The sampling rate for each link can be determined according to the monitoring policy.

Random sampling is performed by sFlow agents in network devices according to the already set sampling rate. It is, therefore, possible to use them to monitor high speed networks with considerable accuracy. The sampled data is sent as UDP packets to the specified host and port where sFlow collector software computes summary statistics and possibly display the result graphically [8].

Fig. 3 shows the basic elements of the sFlow system. sFlow Agents throughout the network continuously send a stream of sFlow Datagram to a central sFlow collector where they are analyzed by an analytics engine to produce a rich, real-time, network-wide view of traffic flows [9].

In order to process sFlow packets received from the network, an extensively used tool named sFlow-RT [6] is used. It offers real-time monitoring ability into Software Defined Networks. sFlow-RT sits in the control plane of the SDN stack. It changes the datagram received into summary statistics or actionable metrics on the flows as specified by the user. A set of packets with a common property constitutes a flow of traffic known as the flow key which is observed within a period of time. Fields from the packet header usually specify the flow key. These fields can be IP source and destination addresses and TCP/UDP port numbers. Metrics which are programmatically approachable through Northbound APIs are usually represented as flow names. Any computer language that supports HTTP request messages (Python, Java script, Perl, bash etc.) can be used to fetch metrics from sFlow-RT. Since JSON encoded text based results can be easily read and broadly supported by programming tools, sFlow-RT statistics can be fetched in JSON format.

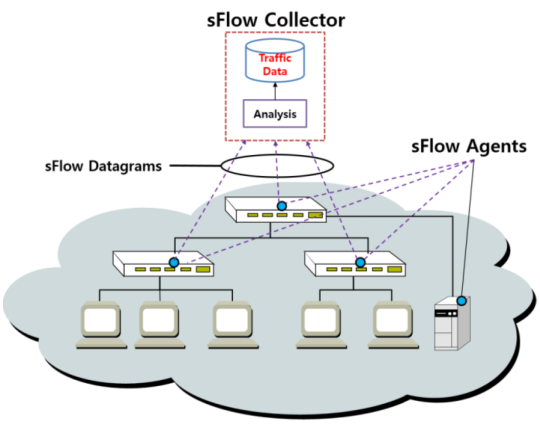


Fig. 3. sFlow Agents and Collector.

4. SYSTEM MODEL

Our system model is based on a testbed using free Mininet network emulation software. Mininet uses Linux containers and Open vSwitch to allow realistic virtual networks of hosts and switches to be constructed using a virtual machine [10].

Since the default behavior of the Floodlight OpenFlow Controller is to offer elementary connectivity which can be selectively overridden using the Static Flow Pusher API, so it was selected for the testbed. By doing so, simple performance optimizing applications can be developed since they do not require to be concerned with sustaining connectivity and are open to emphasis on implementing optimizations. Besides, in order to collect sFlow samples and to implement large flow marking application, sFlow-RT controller is used.

As shown in Fig. 4, the system model consists of a linear topology with four virtual switches. Each virtual switch is connected to a single host. Moreover, each virtual switch is further connected to Floodlight OpenFlow Controller and sFlow-RT controller. sFlow is configured on each switch to capture packets according to a specified sampling

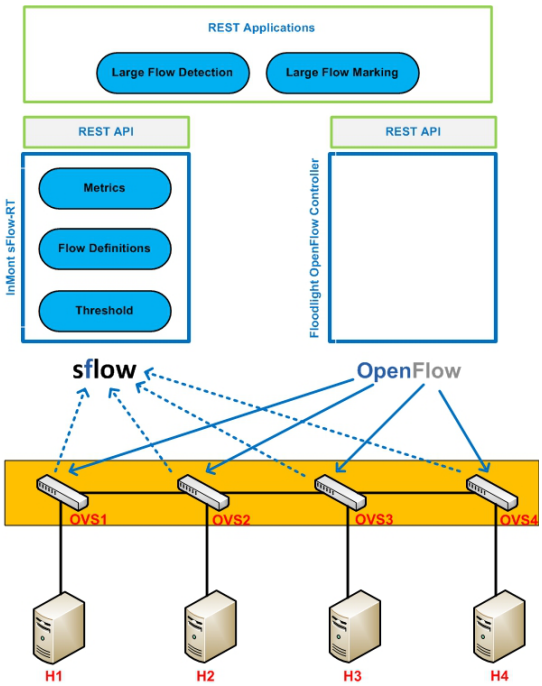


Fig. 4. System Model.

rate. Samples are sent in the form of measurement datagram by each switch or agent to sFlow-RT which is real-time analytics engine. The stream of measurement datagram received from sFlow in real-time is processed by sFlow-RT which then provides real-time summary statistics to control application through northbound REST APIs.

Flows have to be defined in order to get them detected and marked accordingly. A flow is defined using name, keys, value, and optionally filter attributes. Fig. 5 shows the large and small flows defined in our Java Script developed control application for flows detection and marking. The filter for small flows has been set to IP TOS value of 00000000 (decimal 0), whereas the filter for large

Link Speed	Large Flow	Sampling Rate	Polling Interval
10 Mbit/s (Mininet)	>= 1 Mbit/s	1-in10	20 seconds
100 Mbit/s	>= 10 Mbit/s	1-in100	20 seconds
1 Gbit/s	>= 100 Mbit/s	1-in1000	20 seconds
10 Gbit/s	>= 1 Gbit/s	1-in10,000	20 seconds
40 Gbit/s	>= 40 Gbit/s	1-in40,000	20 seconds
100 Gbit/s	>= 10 Gbit/s	1-in100,000	20 seconds

Fig. 6. Threshold Values for Large Flows.

flows has been set to IP TOS value of 10000000 (decimal 128).

sFlow is set for large flow detection by defining a large flow as a flow consuming 10% of the link bandwidth for one second as shown in the table in Fig. 6. Since our testbed is based on Mininet and according to the table the link speed given for Mininet is 10Mbit/s, so a threshold of 1Mbit/s (10% of the link bandwidth) has been defined in our JavaScript control application. This threshold is set so that any flow that consumes 10% or more of the link bandwidth is recorded as a large flow.

Fig. 7 shows the part of JavaScript developed control application which is responsible for flows detection and marking. sFlow measurements from the switches are sent to the sFlow-RT real-time analytics engine. If any flow is greater than the predefined threshold value then sFlow-RT controller records or detects that flow as a large flow in real-time, and immediately installing an OpenFlow rule that matches the flow and instructing the switch to mark the flow by setting the IP type of service bits.

```
jsonPut(rt,'/flow/tos0/json',{"value":'bytes',"filter":'iptos=00000000',"t":'1'},
        function() {}
);
jsonPut(rt,'/flow/tos128/json',{"value":'bytes',"filter":'iptos=10000000',"t":'1'},
        function(){}
);
```

Fig. 5. Flows Definition.

```
function getTopFlows(event) {
  jsonGet(rt,'/metric/' + event.agent + '/' + event.dataSource + '.' +
event.metric + '/json',
    function(metrics) {
      console.log("metrics: " + JSON.stringify(metrics));
      if(metrics && metrics.length == 1) {
        var metric = metrics[0];
        console.log("metric: " + JSON.stringify(metric));
        if(metric.metricValue > thresholdValue
           && metric.topKeys
           && metric.topKeys.length > 0) {
           var topKey = metric.topKeys[0].key;
           console.log("top key: " + topKey);
           record(event.agent,event.dataSource,topKey);
           mark(event.agent,event.dataSource,topKey);}}});}
```

Fig. 7. Flows Detection and Marking Function.

5. LARGE FLOWS DETECTION AND MARKING IN MININET-BASED TESTBED

In order to show large flows detection and marking in our Mininet-based testbed, a step-by-step procedure of the experiment we carried out is given as follows:

5.1 Configuring sFlow on Each Switch

sFlow Agents in network devices use random sampling according to the defined sampling rate and, therefore, can be used to monitor high speed networks (Gbps speeds and higher) with quantifiable accuracy. The sampled data is sent as UDP packets to the specified host and port where sFlow collector software computes summary statistics and possibly display the results graphically.

The following command line is used to configure sFlow on each virtual switch (OVSes):

sudo ovs-vsctl -- --id=@sflow create sflow agent =eth0 target=\"CollectorHost:6343\" sampling=10 polling=20 -- -- set bridge br0 sflow=@sflow

where CollectorHost is the host running sFlow-RT which is a widely used tool to process sFlow packets received from the network devices. Sampling rate is set to 10 which mean that among every 10 packets captured by agent, one will be sent to the collector. A counter polling interval of 20 seconds has been selected so that link utilization can be accurately tracked. Finally, the virtual switch (OVS) on which sFlow is configured is br0.

5.2 Starting Floodlight OpenFlow Controller

Virtual switches are configured to use OpenFlow protocols to communicate with Floodlight OpenFlow controller. The OpenFlow protocol enables SDN controller running on a server to gather topology information of a network of virtual switches and configure the forwarding behavior of these virtual switches. Fig. 8 shows the deployed linear network topology in Floodlight GUI.

5.3 Starting sFlow-RT Controller

As mentioned earlier that sFlow-RT enables re-

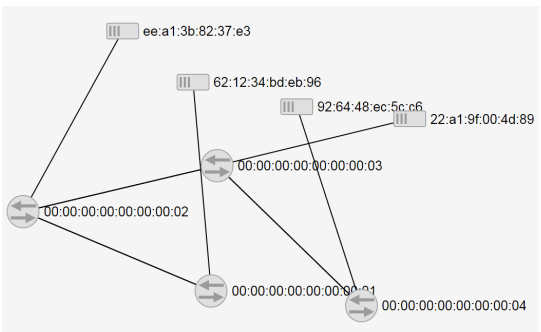


Fig. 8. Linear Topology in Floodlight GUI.

al-time visibility into software defined networks. sFlow-RT sits in the control plane of the SDN stack. It converts the received datagram into actionable metrics or summary statistics based on the flows defined by the user.

Any language that supports HTTP request messages (Perl, Python, Java, Java script etc.) can be used to retrieve metrics from sFlow-RT. sFlow-RT statistics can be retrieved in JSON format. JSON encoded text based results are easy to read and widely supported by programming tools. Following URL is used to retrieve JSON encoded metrics from sFlow-RT:

http://server:8008/metric/agents/metrics/json?filter

where server is the host running sFlow-RT, agents are semicolon separated list of host addresses or names, or ALL to include all hosts, metrics are comma separated list of metrics to retrieve, and filter is a filter to further restrict the hosts to include in the query.

5.4 Executing sFlow-RT's JavaScript-based Control Application

The sFlow-RT's control application that we developed in JavaScript is executed in order to detect and mark the elephant flows exceeding the threshold value. It is able to detect and mark both UDP and TCP flows. Several other terms, such as flow keys, value in bytes, filter, sFlow-RT controller, and Floodlight OpenFlow controller have also been defined in this control application. It is pertinent to mention here that this control application is implemented using node.js which employs asynchronous programming model and is optimized for very high performance I/O.

5.5 Generating Traffic from Different Hosts

The UDP or TCP traffic is generated by running iperf between hosts in xterm using the following command line:

while true; do iperf -c <HostIP> -u -t 40; done

where HostIP means the IP of the host to which traffic is sent, and 'u' shows that the traffic generated is UDP. TCP traffic can also be generated by making one host a TCP server and another a client.

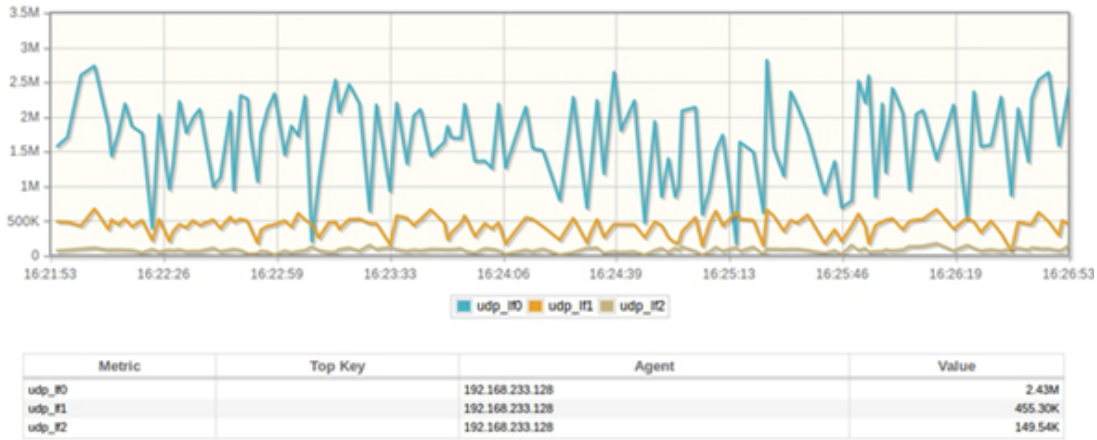
5.6 Visualizing and Analyzing Traffic on sFlow-RT Controller GUI

The following URL is used to visualize and analyze the detected and marked large flows:

http://server:8008/metric/ALL/udp_lf0,udp_lf1,udp_lf2,udp_lf3,udp_lf4,udp_lf5/html

where server is the host running sFlow-RT, and udp_lf is the name of the metric or UDP flow generated between different hosts.

Fig. 9 shows the traffic when only the control application for large flows detection is enabled, whereas the control application for marking of large flows is kept disabled. The flows that exceeds the predefined threshold value of 1Mbps is detected as large flow while the flow below this threshold value is recorded as small flow. Since, in the scope of this paper, our focus is to detect and mark large flows generated by the user, it can been seen from the network topology connected to our system model shown in Fig. 4 that each OVS is connected to a single host. The users can generate traffic from each host targeting other hosts connected in the topology. For the sake of simplicity, we generated UDP traffic by running iperf from different hosts. It is obvious from the figure that the flow named upd_lf0 shown in the light blue line exceeds the threshold value of 1Mbps and hence it trends the large unmarked flow seen in the network. However, the two flows named upd_lf1, and udp_lf2 shown in golden and moss green colors respectively are below the threshold



Metric	Top Key	Agent	Value
udp_lf0		192.168.233.128	2.43M
udp_lf1		192.168.233.128	455.30K
udp_lf2		192.168.233.128	149.54K

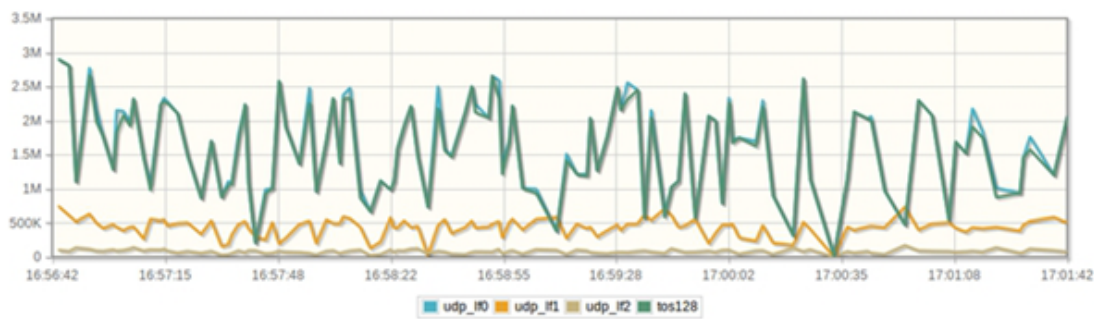
Fig. 9. Flows Detection.

value and hence recorded as small flows.

Fig. 10 shows the traffic when control applications for both large flow detection and marking are enabled. As soon as the flow named udp_lf0 shown in the light blue line exceeds the threshold value, sFlow-RT immediately detects the large flow and makes a call to Floodlight's Static Flow Pusher API to create a rule that matches the IP source and destination addresses of the large flow with action to set the IP Type of Service bits to 1000000 (decimal 128). The flow named tos128 shown in the green line trends the large flow marked, while the other two small flows have been left unmarked. By assigning such different classes of service to large and small flows enables QoS tool like priority queuing (out of the scope of this paper) to give a different priority to large and small flows. Without marking, large flows "Elephants" impact the latency of small flow "Mice".

6. LARGE FLOWS MITIGATION

Distributed Denial of Service (DDoS) attack is one of the major use cases of large flow. The attacker uses a command and control network to instruct large numbers of systems to send traffic to a designated target with the objective of overwhelming the target infrastructure and denying



Metric	Top Key	Agent	Value
udp_lf0		192.168.233.128	2.06M
udp_lf1		192.168.233.128	503.34K
udp_lf2		192.168.233.128	77.86K
tos128		192.168.233.128	2.06M

Fig. 10. Large Flow Marking.

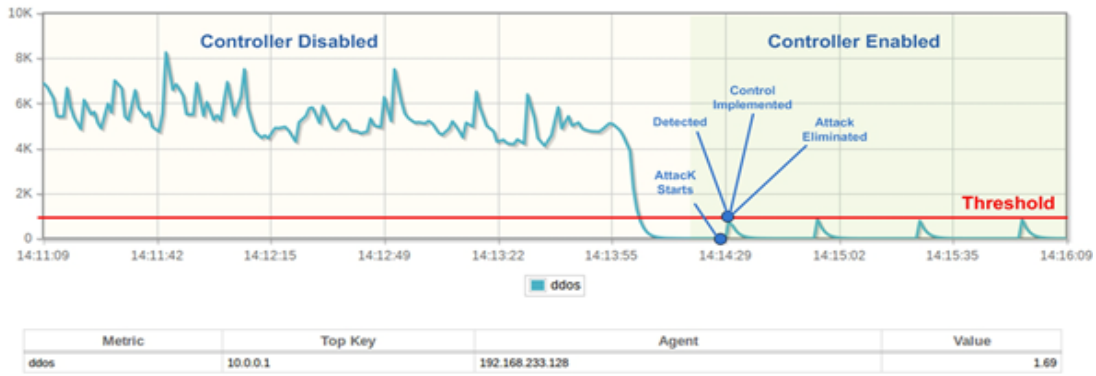


Fig. 11. Large Flow Mitigation.

access to legitimate users [11], [12].

Fig. 11 shows a typical DDoS attack, consisting of traffic levels over 7,000 pakcets per second. When the controller is disabled, the attack traffic sustains over 7000 packets per second until the attacker stops sending. When the controller is enabled, traffic is stopped the instant it hits the 1,000 packets per second threshold defined in the application. The control is removed 20 seconds later and re-triggers if the attacker is still sending traffic.

In our DDoS mitigation system, the sFlow measurements from all the virtual switches are forwarded to the sFlow-RT analytics engine which offers real-time notification of attacks and targets to the DDoS mitigation application. The DDoS mitigation application instructs the controller which communicates with the switches to mitigate the DDoS traffic.

7. CONCLUSION

Software Defined Networking (SDN) is an emerging architecture that allows network administrators to automatically and dynamically manage and control a large number of network devices, services, topology, traffic paths, and packet handling (Quality of Service) policies using high-level language and APIs. Since the utilization rate of the bandwidth has become much more significant, even a short delay in the detection of large flows could result in big loss of the overall performance. In this paper, we exploited QoS in SDN and detected and marked large flows by assigning them different TOS bits as compared to other flows in the network. Once detected and marked, the flows can be re-scheduled quickly which results in an optimize usage of an overall SDN network. In addition to that, we showed how sFlow can be used to rapidly detect DDoS attacks and drive controls to mitigate their effect. Our detection, marking, and controlling approach makes the packet handling in SDN very efficient.

REFERENCES

[1] Blog by Martin Casado, http://network-heresy.com/2013/11/01/of -mice-and-elephants/ (Accessed December 15, 2014)

[2] R. Zhou, "Datacenter Network Large Flow Detection and Scheduling from the Edge," Reading & Research Project, 2014.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Bahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," *Proceeding of Networked Systems Design and Implementation Symposium,* Vol. 10, pp. 19-19. 2010

[4] P. Peter, S. Panchen, and N. McKee, *InMon Corporation's sFlow: A Method for Monitor-*

ing Traffic in Switched and Routed Networks, RFC 3176, 2001.

[5] The Project Floodlight, http://www.project Floodlight.org/documentation/ (Accessed December 08, 2014).

[6] sFlow-RT, http://www.inmon.com (Accessed December 10, 2014).

[7] Stephen Baucke, Kyle Mestery, Anees Shaikh, Chris Wright, "OpenDaylight: An Open Source SDN for your OpenStack Cloud," *An OpenStack Summit, Hong Kong.* 2013.

[8] P. Goransson and C. Black, *Software Defined Networks: A Comprehensive Approach*, 1st Edition, 2014. Elsevier.

[9] Traffic Monitoring using sFlow, http://www.sflow.org (November 28, 2014).

[10] Controlling Large Flows with OpenFlow, http://blog.sflow.com/2013_05_01_archive.html (December 18, 2014).

[11] DDoS, http://blog.sflow.com/2013/03/ddos.html (December 20, 2014).

[12] Muhammad Nugraha et al., "Utilizing OpenFlow and sFlow to Detect and Mitigate SYN Flooding Attack", *Journal of Korea Multimedia Society,* Vol. 17, No. 8, August 2014 (pp. 988-994).

Muhammad Afaq

He received BS degree in Electrical Eng. from University of Eng. and Technology, Peshawar, Pakistan, and MS degree in Electrical Eng. with emphasis on Telecom from Blekinge Institute of Technology, Sweden in 2007 and 2010 respectively. Currently, he is pursuing his PhD degree as a KGSP (Korean Government Scholarship Program) scholar at Jeju National University. He has worked as a Research Associate in the Faculty of Comp. Sci. and Eng. at GIK Institute of Eng. Sceinces and Technology, Pakistan. His research interests are software defined networking, wireless networks, protocols etc.

Shafqat Rehman

He received masters degree MS (Computer Engineering) from Jeju National University, South Korea in 2008. He was awarded Korean Government IT scholarship for Master's studies. He did his PhD (Networks and Distributed Systems) from INRIA (French National Institute for Research in Computer Science and Control), SophiaAntipolis,France in 2012. He is currently working as an Assistant Professor at Air University, Islamabad, Pakistan. His research interests mainly include experimentation platforms, distributed systems, cloud computing, software defined networking, data science, mobile computing, security, Internet architecture, wireless networks, protocols etc.

Wang-Cheol Song

He received B.S. degree in Food Engineering and Electronics from Yonsei University, Seoul, Korea in 1986 and 1989, respectively. And M.S. and PhD in Electronics studies from Yonsei University, Seoul, Korea, in 1991 and 1995, respectively. Since 1996 he has been working at Jeju National University. His research interests include VANETs and MANETs, Software Defined Networks, network security, and network management.