논문 17-42-03-09 The Journal of Korean Institute of Communications and Information Sciences '17-03 Vol.42 No.03
https://doi.org/10.7840/kics.2017.42.3.612

SDN에서 엘리펀트 플로우의 실시간 분류, 시각화 및 QoS 제어

아팍 무하마드◆, 송 왕 철°

Real-Time Classification, Visualization, and QoS Control of Elephant Flows in SDN

Afaq Muhammad◆, Wang-Cheol Song°

요 약

오래 지속되는 플로우인 엘리펀트 플로우는 데이터 센터에서 많은 대역폭을 소비해서, 지연에 민감하고 짧은 시간 흐르는 플로우인 마이스 플로우의 흐름을 방해하게 된다. 이는 마이스 플로우에 대해 적지않은 지연을 발생시켜서, 결과적으로 그 네트워크에서 동작하고 있는 응용의 성능을 저하시키게 된다. 그러므로 데이터 센터 네트워크는 이를 분류하고 시각화 해내어 실시간으로 QoS 프로비저닝을 제공할 수 있어야 한다. 본 논문에서는 다음에 대해 논한다. (1) SDN에서 sflow를 이용한 엘리펀트 플로우의 실시간 검출 및 시각화를 위한 프레임워크 제시. 이는 시각화된 토폴로지에서 스위치를 더블클릭 하므로써 스위치를 통과하는 엘리펀트 플로우를 점검할 수 있게 한다. (2) SDN 제어기에 의해서 정의되고 관리되는 QoS를 보장하는 접근 및 OpenFlow에 의해 제공되는 규격. 본 논문에서는 SDN 네트워크내에서 rate-limiting (traffic-shaping) 분류 기법을 사용하는 것을 주로 논의한다.

Key Words : Elephant Flows, Classification, Detection, Visualization, Floodlight, sFlow, Avior, QoS, Rate-limiting, SDN, Mininet

ABSTRACT

Long-lived flowed termed as elephant flows in data center networks have a tendency to consume a lot of bandwidth, leaving delay-sensitive short-lived flows referred to as mice flows choked behind them. This results in non-trivial delays for mice flows, eventually degrading application performance running on the network. Therefore, a datacenter network should be able to classify, detect, and visualize elephant flows as well as provide QoS guarantees in real-time. In this paper we aim to focus on: 1) a proposed framework for real-time detection and visualization of elephant flows in SDN using sFlow. This allows to examine elephant flows traversing a switch by double-clicking the switch node in the topology visualization UI; 2) an approach to guarantee QoS that is defined and administered by a SDN controller and specifications offered by OpenFlow. In the scope of this paper, we will focus on the use of rate-limiting (traffic-shaping) classification technique within an SDN network.

※ This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2016R1D1A1B01016322).

◆ First Author : Jeju National University Department of Computer Engineering, afaq24@gmail.com, 학생회원
° Corresponding Author : Jeju National University Department of Computer Engineering, kingiron@gmail.com, 종신회원

논문번호 : KICS2016-12-378, Received December 11, 2016; Revised February 7, 2017; Accepted February 17, 2017

Ⅰ. Introduction

Most of the flows in data center networks tend to be short-lived mice flow, whereas the majority of packets belong to a few long-lived elephant flows. Mice flows are bursty, latency-sensitive applications like Voice over IP (VoIP) and search results, whereas elephant flows on the other hand are large transfers, such as backups, or back-end operations[1].

Network resources are utilized based on the requirements and limitations of different applications The tendency of elephant flows to fill buffers end-to-end causes significant delay to mice flows that actually share the same buffers with them, eventually degrading the overall network performance. Routing method like ECMP in data center networks may route elephant flows onto same links, but this yields to lower quality network utilization because other links will be under-utilized[2]. Hence, it is required to detect elephant flows, visualize them, and finally handle them using some QoS provisioning techniques.

Several techniques like Hedera[3], and Mahout[4] are currently available for the detection of elephant flows. Each technique has its own advantages and disadvantages. In this paper, we propose a framework based on sFlow[5] sampling technology that enables to detect elephant flows in SDN[6] systems by means of our developed SDN control application. The elephant flows can then be visualized in a separate pop-up window linked to sFlow-RT[7] traffic analysis tool. Since SDN systems use OpenFlow switches, the sFlow-RT receives a stream of sFlow measurements from the OpenFlow switch, rapidly detects elephant flows in real-time, and notifies the control application. Furthermore, the double-click functionality in our framework allows the administrator to visualize all flows including elephant flows that traverse a switch in the network topology.

The major challenge after the detection and visualization of elephant flows is to properly handle them. They may be either rescheduled, reshaped, reprioritized, or routed through high speed links by means of some QoS technique. While efforts in this area over the past include Overprovisioning, Type of Service (ToS), Differentiated Services (DiffServ), Integrated Services (IntServ), Resource Reservation Protocol (RSVP), Multiprotocol Label Switching (MPLS), and Traffic Shaping; we will only focus on the last one in this paper. Specifically, this will focus on how Traffic Shaping (rate limiting) QoS technique can be used to handle elephant flows properly within an SDN network.

The remainder of the paper is organized as follows. In Section II, the state of the art approaches for elephant flows detection and handling in data center networks are described. In Section III, the architecture of the proposed framework is presented. In Section IV, the results obtained from the real-time detection and visualization of elephant flows are explained. In Section V, the QoS module is presented, and in Section VI, the QoS handling of the detected elephant flows by means of this module is shown. In Section VII, the applicability of our proposed framework to large-scale SDN networks is shown. The paper is concluded in Section VIII.

Ⅱ. Overview of Elephant Flows Detection Systems

The mice and elephant flows may degrade network performance if not handled properly. Significant overhead can be yielded at both the data plane and control plane if a high number of such flows are aggregated at the end of SDN network devices[8].

Equal-cost multi-path (ECMP)[9] is a routing technique to load-balance traffic over multiple available paths using flow hashing methods. The ECMP-enabled network devices are configured with various possible forwarding routes for a given subnet. A packet having multiple potential paths is routed on the one that matches to a hash of packet header's selected fields modulo the number of routes, eventually distributing load to each subnet among multiple routes[8,9]. However, the main disadvantage of ECMP is that the collision of two more elephant flows on their hash may lead them on the same output port, eventually resulting in a

bottleneck situation. This may also overwhelm switch buffers affecting overall link and switch utilization[8].

Hedera[3] is a flow scheduling system to avoid the disadvantages of ECMP. It gets flow information from switches, determines routes for flows, and sends them to switches for re-routing the traffic accordingly. When it detects elephant flows, the edge-switch forwards it along one of its equal-cost routes. This route is utilized until the elephant flow reaches a pre-defined threshold rate. Hedera then dynamically computes a suitable route for it and installs that path on the switch[8]. However, in Hedera, elephant flows are detected by periodic polling that retrieves the per-flow statistics from each of the edge switch. Since, each edge-switch will be required to manage a huge amount of flows, which may not only be infeasible in the real OpenFlow switch implementation, but may also cause network congestion between polls.

Mahout[4] manages flow traffic by introducing timely periodic detection of elephant flows. It addresses limitations in Hedera by monitoring and detecting elephant flows at the end host through a shim layer in the Operating System instead of monitoring the edge-switches. However, Mahout is without an automated decision-making module that can continually determine a suitable threshold based on information from the network for elephant flow detection[10].

Our proposed approach based on sFlow technology consists of multiple features unlike the aforementioned approaches. It is able to classify, detect, handle (QoS), and visualize elephant flows in SDN. More precisely, the detected elephant flows are handled by the QoS module, which forwards them to rate-limited queues, eventually subjecting them traffic-shaping QoS technique.

Ⅲ. Framework for the Detection, Classification, and Visualization of Elephant Flows

In our proposed framework, each switch (sFlow agent) of a network topology is configured in such a manner that they forward sFlow samples to sFlow collector in real-time. The sFlow agent is a software process that runs as a part of the network management software within a device. It combines flow samples and interface counters into sFlow datagram that are sent across the network to an sFlow collector. Packet sampling is usually performed by the routing/switching ASICs, providing wire-speed performance. The state of the routing/forwarding table entries associated with each sampled packet is also recorded. 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 significant 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[11]. A sampling rate of 10 is used in the experiments, which means that out of every 10 packets captured by an sFlow agent, one will be sent to the sFlow collector.

sFlow-RT is a widely used sFlow collector and traffic analysis tool to process sFlow packets retrieved from the network. It sits in the control plane of SDN stack and offers real-time monitoring ability. It converts the retrieved datagram into flow summary statistics, or defined actionable metrics that are to be applied on the flows. A set of packets with an identical property is called a flow key which is observed periodically. The fields from a packet header like source and destination IP addresses, TCP/UDP port numbers usually specify the flow key. Flow names are represented as metrics, and programmatically accessible through RESTful Northbound APIs. OpenFlow controller and sFlow that are part of control plane software, use Northbound API to provide control functionality to SDN applications and summary statistics. Fig. 1 shows the functionality of several components used in our proposed framework. Avior is a GUI for OpenFlow networks management. It focuses on usability and versatility with various dynamic network statistics and useful management tools[12].

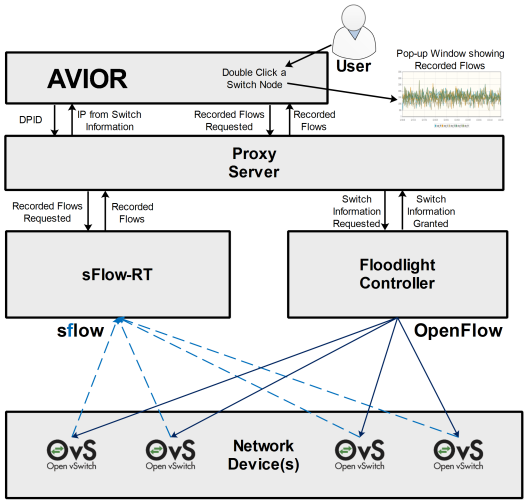


Fig. 1. Proposed Framework

Two essential JavaScript programs have been developed and named ElephantFlow-record, and proxy-server. Both these programs have been implemented using node.js which is optimized for very high performance I/O, and employs asynchronous programming model. In addition, the double-click feature have been added to Avior for visualizing the recorded flows in real-time when a switch node in the network topology is double clicked in Avior GUI. The traffic samples may be retrieved from a various devices like virtual switches, physical switches, and hosts. Every single interface of a device may be configured with sFlow for monitoring with very little overhead. The monitoring policy defines the sampling rate for each link.

The developed flow classification control application is generic, and records all types of TCP, UDP, and ICMP elephant flows. Fig. 2 shows the flow chart of 'ElephantFlowRecord' function defined in JavaScript ElephantFlow-record program. This function has a pre-defined threshold value for detecting the elephant flows, as well as method to push them to sFlow-RT. The flow keys allow recording of any flow greater than the pre-defined threshold value as elephant flow. The elephant flows are classified by taking into account the list of protocol numbers used in the Protocol field of the IPv4 header. In that list, the protocol number of for

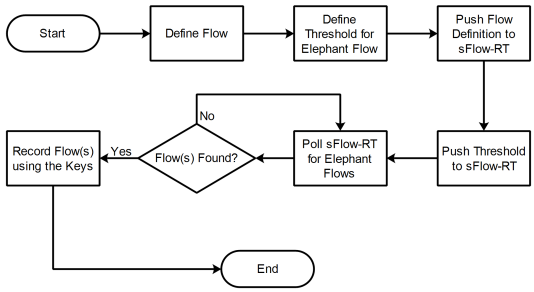


Fig. 2. Flow Chart for ElephantFlow-record Function

example TCP is 0x06 in Hexadecimal. So whenever the condition for Flow key is equal to 0x06 then the flow is classified and recorded as a TCP flow. Similarly, other flows like UDP and ICMP have Protocol numbers 0x11 (decimal 17) and 0x01 respectively. They are recorded accordingly whenever flow keys match their protocol numbers. In addition, our proposed framework enables users to specify any flow by means of a comma separated list of keys.

In Avior, the JavaScript topologyView has been also customized to show the topology-related information and the overall network topology in Avior GUI. It consists of the functions defined for rendering the legend and network topology using d3 (data driven documents) JavaScript library, showing labels, click events, and creating as well as displaying the network graph. It also gets flow information from sFlow-RT that can be viewed under each switch node of the network topology.

The developed 'double-click' function has been embedded in JavaScript topologyView, and it defines sFlow Agent (OVS) IP address, variables for WebSocket server, and sFlow-RT. It results in a new pop-up window which displays elephant flow(s) traversing a switch node that has been double-clicked in Avior GUI. The pseudocode for double-click function is shown in Fig. 3.

```
Get data associated with the node
  if node type is switch then
     Open web socket connection to ws-proxy-server
     Resolve switch DPID to IP
     Use IP to get list of elephant flows through the switch
     if there are any elephant flows then
        Show time series plot of flows in a pop-up window
     end if
  end if
```

Fig. 3. Pseudocode for Double-click Function

In order to overcome the restriction of browser's same origin policy that does not allow Avior to communicate directly with Floodlight OpenFlow controller[13] and sFlow-RT, web sockets are used which are not bound to this security policy. More precisely, Avior communicates with Floodlight OpenFlow controller and sFlow-RT by means of a proxy server. Avior has initially only the Data Path IDs (DPIDs) of the switches. It sends requests proxy server in order to get IPs for corresponding DPIDs. The proxy server forwards this request to Floodlight OpenFlow controller, retrieves IP address and other switch-related information, and sends a reply to Avior. After getting the IP of the switch, Avior sends a request to sFlow-RT to retrieve the names of the recorded elephant flows. On double clicking any switch node of the network topology shown in Avior GUI, a new pop-up window opens which displays all the flows traversing that particular switch node. Fig. 4 shows the pseudocode for JavaScript proxy-server program.

```
Listen to connection from Avior topology view
Accept the connection
Wait for requests on that connection
while On request Receipt do
    if it is request for elephant flows then
       Fetch flows from sFlow-RT using REST API
       Prepare a list of flows
       Add the list to JSON object
       Stringify the JSON object
       Send the string to Avior topology
    else
      if it is request for DPID to IP mapping
      Get switch info from the Floodlight controller using REST API
      Extract IP address to from the information
      Add IP address to JSON object
      Stringigy the JSON object
      Send the string to Avior topology view
    end if
end while
```

Fig. 4. Pseudocode for proxy-server Program

Ⅳ. Real-time Classification, Detection, and Visualization of Elephant Flows

In order to validate the experiments, the physical test-bed illustrated in Fig. 5 was realized. The proposed framework is deployed on top of a linear bus network topology. The network topology has four OVSes, where each OVS is connected to one more hosts. Fig. 6 shows the same network topology in Avior GUI, where the elephant flows traversing each switch node are visualized by simply double clicking that switch node.

Three different scenarios have been investigated, i.e. detection and visualization of elephant flows classified as UDP, TCP, and ICMP. The threshold value of 100KBps (800Kbps) has been pre-defined in ElephantFlow-record control application. Any flow surpassing this threshold value is recorded is elephant flow. Each of the scenarios is explained in detail in the following sub-sections.

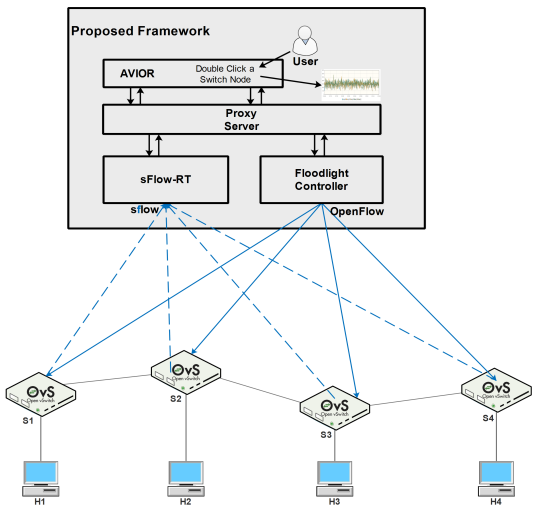


Fig. 5. Linear Bus Network Topology with Proposed Framework

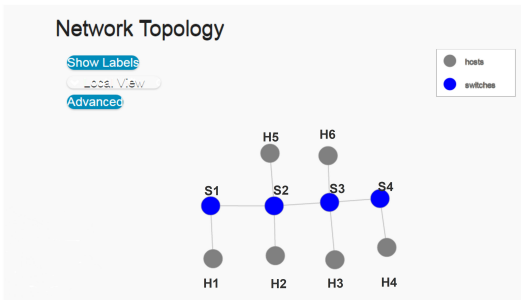
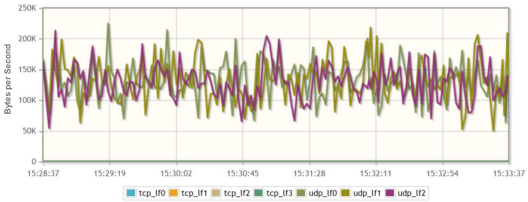


Fig. 6. Linear Bus Network Topology in Avior GUI

4.1 Detection and Visualization of Flows Classified as UDP

The emphasis is to visualize the elephant flows generated by users. Fig. 5 shows that each switch in the network topology is connected to at least one host. The users on hosts H4, H2, and H1 generate traffic destined to hosts H3, H4, and H3 respectively. Fig. 7 shows all three detected elephant



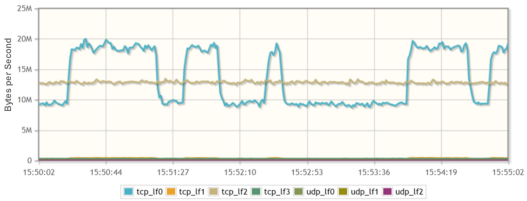
Metric	Top Key	Agent	Value(Bps)
udp_lf0	17,192.168.0.101,192.168.0.103	117.17.102.169	144.23K
udp_lf1	17,192.168.0.104,192.168.0.103	117.17.102.169	118.49K
udp_lf2	17,192.168.0.102,192.168.0.104	117.17.102.169	135.97K

Fig. 7. UDP Flows Traversing S3

flows traversing switch node S3 when it is double clicked in Avior GUI.

4.2 Detection and Visualization of Flows Classified as TCP

Hosts H3 and H4 are configured to act as TCP servers in this scenario, whereas hosts H2 and H1 as TCP clients. The clients H1 and H2 generate TCP traffic destined to TCP servers H3 and H4 respectively. This results in traffic traversing switch nodes S1, S2, and S3. The elephant TCP flows are visualized as shown in Fig. 8 when the switch node S2 is double clicked in Avior GUI.

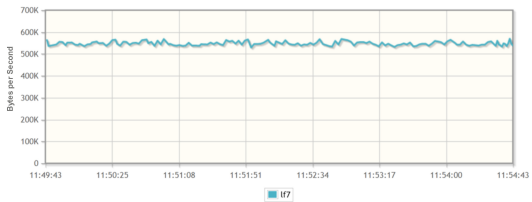


Metric	Top Key	Agent	Value (Bps)
tcp_lf0	6,192.168.0.101,192.168.0.103	117.17.102.109	19.20M
tcp_lf1	6,192.168.0.103,192.168.0.101	117.17.102.109	379.54K
tcp_lf2	6,192.168.0.102,192.168.0.104	117.17.102.109	12.83M
tcp_lf3	6,192.168.0.104,192.168.0.102	117.17.102.109	305.07K

Fig. 8. TCP Flows Traversing S2

4.3 Detection and Visualization of Flows Classified as ICMP

Usually access to a specific network resource is denied as a result of a DoS attack. More precisely, that particular network resource is overwhelmed when it is targeted by a flood of illegitimate connections. Some specific weaknesses in an IP protocol are exploited by each type of DoS attack[14]. In this scenario, an elephant flow is generated by means of flood ping from host H1 destined to host H3. The detection and visualization



Metric	Top Key	Agent	Value (Bps)
lf7	1,192.168.0.101,192.168.0.103	117.17.102.109	537.88K

Fig. 9. ICMP Flood Elephant Flows

of the generated elephant ICMP flow is shown in Fig. 9.

4.4 Analysis of the Detected Elephant Flows

Table 1 explains the details of the detected and visualized UDP, TCP, and ICMP elephant flows shown in Fig. 7, Fig. 8, and Fig. 9 respectively. The Metric column of the table shows the name of a particular detected and visualized elephant flow, the Top Key column represents the classification protocol number, and the IP addresses of the source and destination hosts of that flow, the Agent column shows the sFlow-configured OVS switch node of the network topology which forwards the elephant flow, the Value column gives the flow rate in bytes per second for each respective detected elephant flow, and the Threshold Value column has the reference threshold value above which any flow detected is recorded as an elephant flow.

For instance, it is obvious from the protocol number 17 in the third column of Table 1 that udp_lf0 has been classified as a UDP flow. As mentioned in subsection 1 of Section IV, this flow is due to the traffic generated from host H1 (192.168.0.101) destined to host H3 (192.168.101.103). It is visualized when the switch node S3 (117.17.102.169) is double-clicked in Avior GUI. Finally this flow has been detected as an elephant flow because its flow rate value is greater than the pre-defined threshold value of 100KBps. Similarly, each elephant flow shown in Table 1 is classified, detected, and visualized in the same manner. As mentioned above that the protocol number for TCP and ICMP are 6 and 1 respectively.

Table 1. Detected Elephant Flows with reference to a Threshold Value

Flow Classification	Metric	Top Key	Agent	Value(Bps)	Pre-defined Threshold Value
UDP elephant flows	udp_lf0	17, 192.168.0.101, 192.168.0.103	117.17.102.169	144.23K	100KBps
	udp_lf1	17, 192.168.0.104, 192.168.0.103	117.17.102.169	118.49K	
	udp_lf2	17, 192.168.0.102, 192.168.0.104	117.17.102.169	135.97K	
TCP elephant flows	tcp_lf0	6, 192.168.0.101, 192.168.0.103	117.17.102.109	19.20M	
	tcp_lf1	6, 192.168.0.103, 192.168.0.101	117.17.102.109	379.54K	
	tcp_lf2	6, 192.168.0.102, 192.168.0.104	117.17.102.109	12.83M	
	tcp_lf3	6, 192.168.0.104, 192.168.0.102	117.17.102.109	305.07K	
ICMP elephant flows	lf7	1, 192.168.0.101, 192.168.0.103	117.17.102.109	573.88K	

V. A Software-Defined QoS Module

One of the requirements for end-to end performance guarantees is provided by SDN OpenFlow protocol which allows traffic control on per-flow basis. Common queuing QoS technique within OVS is utilized to access a software-defined QoS module[15,16]. n order to achieve a software-defined approach to this, rate-limiting paths have been defined through a given network of N switches. More precisely, a module named QoS has been implemented in Floodlight OpenFlow controller that allows to insert and delete flows, and also handle QoS policies.

The architecture for Floodlight OpenFlow controller with an added QoS module to it is shown in Fig. 10. Its northbound REST API is exploited by the QoS module that uses REST applications to manage QoS and apply QoS paths through the network. As obvious from the name, QoSManager REST application manages QoS from the command line, whereas QoSPath REST application uses Floodlight Circuit Pusher to implement QoS circuit through the network. When the QoS state is pushed down the network by means of QoSPath application, all packets that match certain fields into the rate-limited queue are enqueued. All packets are matched against OpenFlow 12-tuple of fields, i.e. IP and MAC addresses, Ethernet type, protocol, TCP/UDP ports, and so on. Hence, a flow with the matched destination and source IP addresses is forwarded for further actions to the queue that is pre-defined in the policy. An example of pseudocode for defining the policies is shown in Fig. 11.

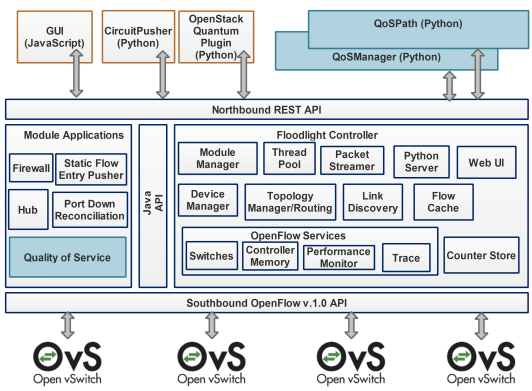


Fig. 10. QoS Module in Floodlight Architecture

```
add policy
 {Name:"Name of the policy",
  Protocol:"6",
  Ethernet-type:"0x0800",
  IP source:"IP address of the source host",
  Other optional fields in the OpenFlow 12-tuple,
  Switch:"Switch on which the policy is added",
  Queue:"queue on which the policy is added"}
```

Fig. 11. Pseudocode for Defining Policies

VI. Handling (QoS Control) of the Detected Elephant Flows

The aim of this section is to validate the QoS module embedded in Floolight OpenFlow controller, as discussed in Section V. To this purpose, the same linear bus network topology has been deployed in Mininet. Mininet uses Linux containers and Open vSwitch to allow realistic virtual networks of hosts and switches to be constructed using a virtual

machine[17]. In order to cope with Mininet Gigabit links, the pre-defined threshold value for the detection of elephant flows has been increased to 600KBps (4.8Mbps). Any flow surpassing this value is recorded as an elephant flow.

For each OVS in the network topology, three different queues have been set-up. Each port of an OVS has been assigned the maximum available bandwidth of 10Gbps, eventually allotting each queue its own bandwidth according to the QoS policy. The first queue, q1, has been assigned all the available bandwidth of 10Gbps. The second queue q2, and the third queue q3 have been rate-limited to 1Gbps and 2Mbps respectively. Fig. 12 depicts an incoming flow at OVS, the enqueuing process, and an outgoing flow. All three queues have been also illustrated in the figure, each with its own respective queue number.

Host H2, allotted an IP address 10.0.0.2 has been configured to act as a TCP server, whereas host H1 having an IP address 10.0.0.1 as a client. Iperf[18] has been generated from client destined to server via queue q1 first. It is obvious from Fig. 13 that the bandwidth limited to around 1Gbps has been used by queue q1 for traffic between client and server. This traffic has been visualized by our proposed framework as show in Fig. 14. Since it greatly

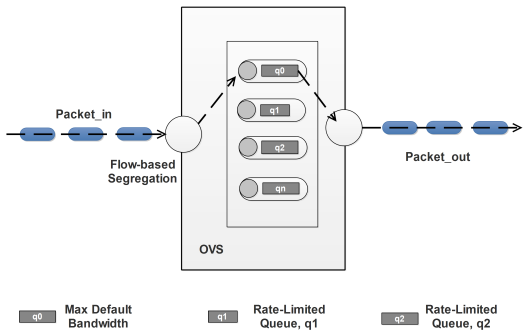


Fig. 12. Queue-based Classification in OVS

```
------------------------------------------------------------
[  4] local 10.0.0.1 port 48675 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-40.0 sec  4.42 GBytes   948 Mbits/sec
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

Fig. 13. Bandwidth Allocated to Queue q1

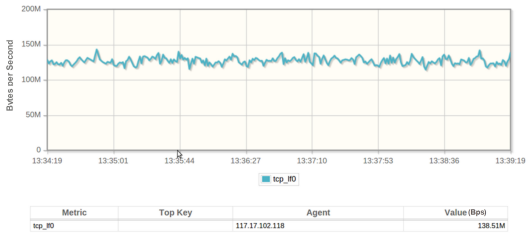


Fig. 14. Elephant Flow Visualized in q1

surpasses the pre-defined threshold value, hence it is recorded as an elephant flow.

The Iperf traffic between the same client and server has been also generated through queue q2, which limits it to around 2Mbps because of the bandwidth restriction for this queue as obvious from Fig. 15. The flow visualized by our proposed framework is below the pre-defined threshold value as shown in Fig. 16, which means that any elephant flow routed through queue q2 is not allowed to surpass the pre-defined threshold value. Thus all elephant flows subjected to this rate-limited queue are handled (controlled) in this manner.

Table 2 explains the handling of elephant flow before and after the application of QoS as shown in Fig. 14 and Fig. 16 respectively. The tcp_lf0 flow, before the application of QoS traverses the OVS switch node (117.17.102.118) with the flow rate of more than 100MB/second which greatly exceeds the threshold value of 600KB/second, hence it is detected as an elephant flow. The same elephant

```
------------------------------------------------------------
[  4] local 10.0.0.1 port 47336 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-42.8 sec  11.6 MBytes  2.28 Mbits/sec
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

Fig. 15. Bandwidth Allocated to Queue q2

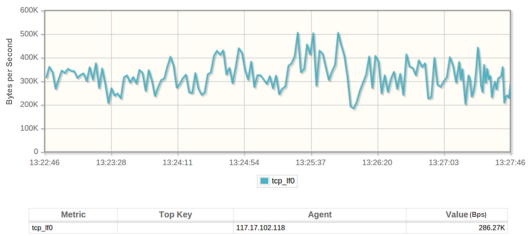


Fig. 16. Elephant Flow Rate-limited in q2

Table 2. QoS Handling of Detected Elephant Flows

Metric	Agent	Value(Bps)	Threshold Value
tcp_lf0 (before QoS)	117.17.102.118	138.51M	600KBps
tcp_lf0 (rate-limited)	117.17.102.118	286.27K	

flow is controlled and has a flow rate of 286.27KB/second when forwarded through the QoS rate-limited queue.

6.1 Impact of Elephant Flows on Network Performance

Distributed Denial of Service (DDoS) is a major use case of elephant flows. The intruder uses a control and command network to command various systems to send traffic to a specific target with the aim of overwhelming the target infrastructure and preventing the legitimate users from accessing the services offered by the server[19,20].

A typical DDoS attack consists of traffic levels that exceed a specific threshold value. As shown in previous sections, any flow above this threshold value is detected as an elephant flow. If this detected elephant flow is not handled properly (rate-limited), the attack traffic sustains above the threshold value until the intruder stops sending. On the other hand, the system can be prevented from performance degradation if the detected elephant flow is directed to the rate-limited queue to mitigate the DDoS traffic.

Ⅶ. Applicability of Proposed Framework to Large-scale SDN Networks

In this section, we show the validity of our proposed framework to large-scale SDN networks. So far, the experiments have been performed on a small-scale testbed, but in actual scenario, the datacetners have a huge number of hosts and network devices. So in order to emulate an actual large-scale SDN-based network scenario, we created a tree type network topology which consists of one thousand hosts has been created in Mininet. The OVSes in this topology are represented by the root

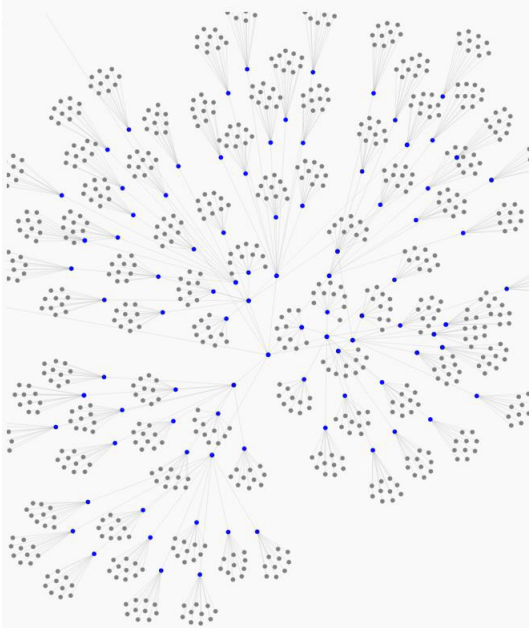


Fig. 17. Topology for Large-scale SDN Network

nodes, whereas hosts by the branch nodes as shown in Fig. 17.

The generic flow classification capability of our proposed framework has been already validated in the previous sections, i.e., any elephant flow classified as TCP, UDP, or ICMP can be detected and visualized. For the purpose of validation our proposed framework to large-scale SDN networks, UDP traffic has been generated by four random hosts destined to some random hosts in the large-sale network topology of Fig. 17. The threshold value has been set to 100KBps. It is obvious from Fig. 18 that all four flows have been detected and classified as UDP elephant flows

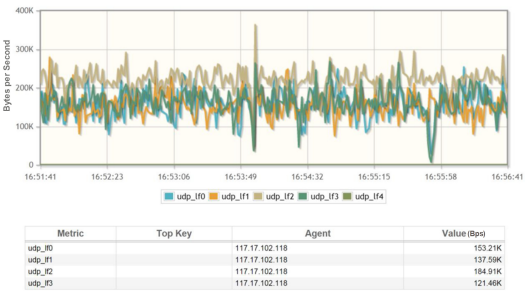


Fig. 18. Detection and Visualization of Elephant Flows in Large-scale SDN

because their flow rate values exceed the pre-defined threshold value of 100 KBps. As a result of which, all four UDP elephant flows have been visualized when the sFlow agent (117.17.102.118) is double-clicked in the network topology GUI.

Ⅷ. Conclusions

Nowadays, due to the increase in the number of users in datacenter network, the consumption rate of the bandwidth has become quite significant. With this increase in the requirement of bandwidth, even a short delay in the detection and handling of elephant flows could result in the overall performance degradation in the datacenter network. In order to cope with this, we presented a framework to classify, detect, and visualize elephant flows in real-time. We also added the double-click feature by means of which elephant flows traversing a switch node in a network topology can be visualized by simply double clicking that switch node in Avior GUI. In addition, we handled the detected elephant flows by presenting a QoS provisioning approach that is defined and managed by an SDN OpenFlow controller. We validated this approach by using rate-limiting QoS classification technique within an SDN network. Finally, we showed the applicability of our proposed framework to a large-scale SDN network.

In the future, we aim to extend this work by measuring the elephant flow detection error and the detection delay based on the sFlow sampling rate defined when configuring sFlow on a network device.

References

[1] J. S. Marron, Felix Hernandez-Campos, and F. D. Smith, “Mice and elephants visualization of internet traffic,” in *Compstat*, pp. 47-54, Physica-Verlag HD, 2002.

[2] J. Liu, J. Li, G. Shou, Y. Hu, Z. Guo, and W. Dai, “SDN based load balancing mechanism for elephant flow in data center networks,” in *IEEE 2014 Int. Symp. Wirel. Pers. Multimedia Commun.* (WPMC), pp. 486-490, 2014.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” in *NSDI*, vol. 10, pp. 19-19, San Jose, California, Apr. 2010.

[4] A. R. Curtis, W. Kim, and P. Yalagandula, “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection,” in *Proc. IEEE INFOCOM*, pp. 1629-1637, 2011.

[5] P. Phaal, S. Panchen, and N. McKee, *InMon corporation’s sFlow: A method for monitoring traffic in switched and routed networks*, 2001, Retrieved Nov. 15, 2016, from https://tools.ietf.org/html/rfc3176

[6] A. Islam, et al., “Robust software-defined scheme for image sensor network,” *J. KICS*, vol. 41, no. 2, pp. 215-221, Feb. 2016.

[7] inMon sFlow-RT, Retrieved Nov. 13, 2016, from http://www.inmon.com/products/sFlow-RT.php

[8] Ian F. Akyildiz, et al., “A roadmap for traffic engineering in SDN-OpenFlow networks,” *Computer Networks*, vol. 71, pp. 1-30, 2014.

[9] M. Chiesa, G. Kindler, and M. Schapira, “Traffic engineering with equal-cost-multipath: An algorithmic perspective,” in *IEEE INFOCOM 2014-IEEE Conf. Computer Commun.*, pp. 1590-1598, 2014.

[10] S. Liu, H. Xu, and Z. Cai, “Low latency datacenter networking: A short survey,” arXiv preprint arXiv:1312.3455, 2013.

[11] S. U. Rehman, et al., “Network-wide traffic visibility in OF@TEIN SDN testbed using sFlow,” *IEEE APNOMS*, pp. 1-6, Sept. 2014.

[12] C. Marist, *What is Avior?* (2012), Retrieved Nov. 13, 2016, from http://openflow.marist.edu/avior.html

[13] *Floodlight Is an Open SDN Controller* (2013), Retrieved Nov. 22, 2016, from http://www.projectfloodlight.org

[14] P. Ferguson and D. Senie, *Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing*,

1997, Retrieved Dec. 02, 2016 from https://www.ietf.org/rfc/rfc2827.txt

[15] S. Tomovic, N. Prasad, and I. Radusinovic, "SDN control framework for QoS provisioning," *IEEE TELFOR*, pp. 111-114, 2014.

[16] R. Wallner and R. Cannistra, "An SDN approach: Quality of service using big switchs floodlight open-source controller," in *Proc. Asia-Pacific Advanced Network*, vol. 35, pp. 14-19, Jun. 2013.

[17] Mininet (2013, Mar), *An Instant Virtual Network on your Laptop (or other PC),* Retrieved Oct. 25, 2016, from http://www.mininet.org

[18] Iperf: *The TCP/UDP Bandwidth Management Tool*, Retrieved Nov. 18, 2016, from https://iperf.fr/

[19] G. Bang, et al., "A protection method using destination address packet sampling for SYN flooding attack in SDN," *J. Korea Multimedia Soc*., vol. 18, no. 1, pp. 35-41, Jan. 2015.

[20] J. W. Seo and S. J. Lee, "A study on the detection of DDoS attack using the IP Spoofing," *J. Korea Inst. Inf. Secur. and Cryptol.*, vol. 25, no. 1, pp. 147-153, Feb. 2015.

아팍 무하마드 (Afaq Muhammad)

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. Sciences and Technology, Pakistan. His research interests are software defined networking, network function virtualization, wireless networks, and protocols.

송 왕 철 (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.