This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2755768, IEEE Access

A SDN-based QoS Guaranteed Technique for Cloud Applications

Fuliang Li, *Member, IEEE,* Jiannong Cao, *Fellow, IEEE,* Xingwei Wang, and Yinchu Sun

***Abstract*—Due to the centralized control, network-wide monitoring and flow-level scheduling of Software-Defined-Networking (SDN), it can be utilized to achieve Quality of Service (QoS) for cloud applications and services, such as voice over IP, video conference and online games, etc. However, most existing approaches stay at the QoS framework design and test level, while few works focus on studying the basic QoS techniques supported by SDN. In this paper, we enable SDN with QoS guaranteed abilities, which could provide end-to-end QoS routing for each cloud user service. First of all, we implement an application identification technique on SDN controller to determine required QoS levels for each application type. Then, we implement a queue scheduling technique on SDN switch. It queues the application flows into different queues and schedules the flows out of the queues with different priorities. At last, we evaluate the effectiveness of the proposed SDN-based QoS technique through both theoretical and experimental analysis. Theoretical analysis shows that our methods can provide differentiated services for the application flows mapped to different QoS levels. Experiment results show that when the output interface has sufficiently available bandwidth, the delay can be reduced by 28% on average. In addition, for the application flow with the highest priority, our methods can reduce 99.99% delay and increase 90.17% throughput on average when the output interface utilization approaches to the maximum bandwidth limitation.**

***Index Terms*—Software Defined Networking, cloud computing, QoS, applications.**

I. Introduction

SDN enhances network flexibility and scalability by separating control plane from data plane. It is progressively dominating the dynamic management for timely network trouble shooting and fine grained traffic scheduling in the data center network infrastructure [1, 2], which is the foundation for building today's cloud computing services. More and more multimedia applications are deployed on cloud. So cloud users can access the multimedia data from any geographical location. As the cloud providers, they should meet QoS requirements for each cloud user application.

The Internet uses the traditional best-effort service model to route traffic flows. It cannot provide differentiated services for cloud applications. Then, some enforcement service models are proposed, such as Integrated Services (IntServ) [3], Differentiated Service (Diffserv) [4] and Multi Protocol Label Switching (MPLS) [5]. However, these models are lack of traffic control, or are hard to configure, manage and troubleshoot. Compared with the traditional best-effort service model and the enforcement service models, SDN has the potential to provide a better QoS guarantee for cloud applications and services due to its centralized control, network-wide monitoring and flow-level scheduling [6-7, 23].

Existing studies have tried to use SDN to provide end-to-end QoS routing [8] or multipath routing [9] for multimedia applications. However, most existing approaches stay at the QoS framework design and test level, while few works focus on studying the basic QoS techniques supported by SDN. Different from previous works, we implement and verify a SDN-based QoS guaranteed technique for cloud applications. We combine application identification with queue scheduling to meet the application flows with different required QoS levels. Our methods supply a basic QoS technique for end-to-end QoS routing, as well as make a supplement to existing approaches. The main contributions of this paper are summarized as follows.

(1) We implement an application identification technique on SDN controller based on *C4.5 decision tree*. In addition to identifying application types, it also determines the required QoS level for each type of application and issues corresponding matching rules to SDN switches to meet the QoS requirements of different applications.

(2) We implement a queue scheduling technique to allow delay-sensitive data to be dequeued and sent first. We create multi-queues for each output interface of the switch, including the *Expedited Forwarding (EF)* queue with the highest priority, the *Assured Forwarding (AF)* queue with the medium priority and the *Best Effort (BE)* queue with the lowest priority. We then implement two algorithms to queue the packets into the queues of each output interface and schedule the packets out of the queues with different priorities.

(3) We evaluate the proposed SDN-based QoS technique through both theoretical and experimental analysis. Theoretical analysis shows that our methods can provide differentiated services for the application flows mapped to different QoS levels. And experimental results show that when the output interface has sufficiently available bandwidth, the delay can be reduced by 28% on average. In addition, for the application flow with highest priority, our methods can reduce 99.99% delay and increase 90.17% throughput on average when the output interface utilization approaches to the maximum bandwidth limitation.

According to *OpenFlow* switch specifications [34], it supports limited QoS futures by the *Hierarchical Token Based*

Fuliang Li is with the School of Computer Science and Engineering, Northeastern University, Shenyang, China, and the Department of Computing, Hong Kong Polytechnic University, Hong Kong, China (csfli@comp.polyu.edu.hk).

Jiannong Cao is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, China (jiannong.cao@polyu.edu.hk).

Wingwei Wang and Yinchu Sun are with the School of Computer Science and Engineering, Northeastern University, Shenyang, China (wangxw@mail.neu.edu.cn, qinqinmuji@163.com).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2755768, IEEE Access

(HTB) [35] queuing technique and the *Hierarchical Fair Sequence Curve (HFSC)* [36] queuing technique. Taking *HTB* as an example, it only allows to configure guaranteed minimum rate and limited maximum rate for the flows. And by default, each output interface has only one *First In Firs Out (FIFO)* queue. In this paper, we implement multi- *FIFO* queues for each output interface, and use *HTB* to achieve both rate-limiting and priority-scheduling.

The remainder of this paper is organized as follows. Related work is presented in Section II. Section III describes the SDN-based QoS technique. Section IV analyzes the effectiveness of the queue scheduling technique with the queuing theory. We conduct an experimental evaluation the proposed SDN-based QoS technique in Section IV. Section V concludes the whole paper and gives some future remarks.

II. Related Work

SDN has become a promising network technology and it has been deployed in data center networks [1, 2]. Benefiting from the centralized control, network-wide monitoring and flow-level scheduling, SDN provides the opportunity to achieve a better QoS guarantee for cloud applications and services.

Jeong et al. [10] extended the *Network Operating System* [11] for SDN with the QoS-aware ability of resource discover, routing computation, fault notification and restoration, etc. Wallner et al. [12] showed a basic idea to realize QoS through adding QoS modules and tools to the *Floodlight* controller [13]. Ishimori et al. [14] proposed the control of multiple packet schedulers to improve QoS for SDN. Bueno et al. [15] extended *OpenNaaS* [16] framework with SDN capacity to provides dynamic QoS control. Jarschel et al. [17] proposed an application-aware SDN approach to provide QoE for YouTube video streaming. They evaluated which types of application information can be exploited to enhance QoE. Bari et al. [18] proposed an autonomic SDN-based QoS policy enforcement framework by specifying QoS-based *Service Level Agreements*. Gorlatch et al. [19] used SDN to address the dynamic network demand and improve the QoS of real-time online interactive applications. Akella et al. [6] studied QoS-guaranteed bandwidth allocation for cloud users based on SDN. They introduced queuing techniques and considered the performance metrics of response time and the number of hops. Tomovic et al. [8] presented a new SDN control framework for QoS provisioning. The framework could provide required QoS level for multimedia applications automatically and flexibly. Seddiki et al. [20] proposed a SDN-based approach to achieve per-flow QoS for broadband access networks. Yan et al. [21] proposed an SDN-based multipath QoS solution, which could reduce delay, increase throughput and quickly reroute traffic from path failure. Sieber et al. [22] proposed a *Network Services Abstraction Layer* on top of the network control and management plane. They then introduced a unified data model for both SDN and legacy devices to achieve QoS for time-critical tasks. Dwarakanathan et al. [23] proposed a framework to meet the QoS requirements of cloud applications while providing high availability guarantees. Adami et al. [24] designed and developed a network control application for QoS provisioning on top of the *Floodlight* controller.

Existing works have proved the benefits of SDN to achieve QoS provisioning. Most studies stay at the systematic framework design and test level, while few works focus on queuing and scheduling techniques. In this paper, we implement and verify a SDN-based QoS guaranteed technique, which combines application identification with queue scheduling to achieve QoS guarantee for each cloud user application flows. Our work makes a supplement to existing studies, and provides a basic support for end-to-end QoS routing and multipath routing.

III. A SDN-based QoS Guaranteed Technique

We first briefly introduce the system framework of the proposed SDN-based QoS guaranteed technique. Then, we describe the application identification approach and the queue scheduling mechanisms.

A. System Design

As depicted in Fig. 1, the system mainly contains three modules: the application identification module, the queue management module and the queue scheduling module. We also redesign the control message management modules for both the switch and the controller.

1) Control Message Management: This module is responsible for sending, receiving and processing the control messages, mainly including the *packet_in* message, the *packet_out* message, the *flow_mod* message and the *queue_mod* message. If a packet can match a rule of the flow table, it will be forwarded at a line rate. Otherwise, the switch will generate a *pkt_in* message and send it to the controller. After the controller decides how to forward the packet, it will send a pair of control operation messages (*flow_mod* and *pkt_out*) to the switch: *flow_mod* message carries the forwarding rule that will be installed in the switch; *pkt_out* message instructs to directly forward the miss-match packet through a specified output interface of the switch. The *queue_mod* message is used to configure the queues on the output interface.

2) Application Identification: We implement the application identification technique in this module. it can identify application types according to the application features, and map different application types to different required QoS levels. This will instruct to configure QoS forwarding rules in the flow tables through the control message management module.

3) Queue Management: This module is in charge of configuring queues on the output interfaces and maintaining the queue configuration information. It sends queue configuration commands to a switch through a *queue_mod* message. The switch parses the *queue_mod* message and configures the queues on a specified output interface.

4) Queue Scheduling: This module queues the packets of different applications into different queues, and then schedules the packets out of the queues with different priorities. Each output interface can configure no more than eight queues with different required QoS levels. In our study, we create three queues for each output interface. The relation between an application and a queue is presented in the *action* filed of a

flow table item, marked as $enqueue = x : y$. It means packets of this application is queued into queue y of output interface x.

In addition to the primary modules, we also use the route computation, the topology management and the flow monitoring functions of the controller. They work together with the application identification module to calculate an output interface with a specific queue number for an application flow. The workflow of the proposed QoS guaranteed technique is described as follows.

Step 1-2: A flow contains many packets of $\{p_1, p_2, ..., p_n\}$ arriving at a switch. If p_1 matches a rule of the flow table, it will be directly forwarded through *step 8-9*. Otherwise, the switch needs to request the controller for forwarding decision through *step 3-4* for the miss-match packet.

Step 3-4: The switch generates a *pkt_in* message for p_1 and send it to the controller. The control message management module captures the header fields of p_1 included in the *pkt_in* message. Then, the header information is sent to the application identification module.

Step 5: According to the header information, the application identification module extracts required features and queries the trained classification algorithm to determine the application types and the required QoS levels. The results are sent to the control message management module.

Step 6: According to the classification and QoS mapping results, as well as the forwarding interface computed by the route computation function, the control message management module generates a pair of control operation messages (*flow_mod* and *pkt_out*) and sends them to the switch.

Step 7: The switch installs the forwarding rule in the switch according to the *flow_mod* message and directly queues p_1 into a queue of a specified output interface according to the *pkt_out* message.

Step 8-9: The switch queues the subsequently arrival packets of the application flow into the queue of the specified output interface. And at the same time, the queue scheduling module schedules the queued packets of the application out of the queue of the interfere.

B. Application Identification

In this section, we train the *C4.5 decision tree* to identify application types. Then we define the rules to map each application type to a specific required QoS level.

1) Application Types Identification: Features selection is the basis of application type identification. It starts at the flow setup phase. For TCP application flows, the features includes {*source port*, *destination port*, *MSS* (Maxitum Segment Size), *window size*}. It is well known that the *port* is closely related to application types. In addition, *MSS* and *window size* present great differences among the applications. For UDP application flows, the features includes {*source port*, *destination port*}.

According to the selected features, the *C4.5 decision tree* [25] is trained and implemented in the controller to identify the application types. Assuming the training dataset of S has k kinds of application types, the information entropy of S can be expressed as equation (1).

$$H(S) = -\sum_{i=1}^{k} p_i \log_2 p_i \tag{1}$$

Each element in $\{p_1, p_2, ..., p_k\}$ represents the probability of an application type appearing in the dataset of S. Entropy is used to address the information uncertainty. The smaller the entropy is, the lower uncertainty the information is. And low information uncertainty means the dataset of S is concentrated in some application types. If we divide S into *n* subsets according to the attributes of *X*. The expectation entropy of *X* to S (conditional entropy) is expressed as equation (2).

$$H(S|X) = \sum_{i=1}^{n} p(S_i) H(S_i) \tag{2}$$

As shown in equation (3), the information gain is the difference between the entropy and the expectation entropy.

$$Gain(S, X) = H(S) - H(S|X) \tag{3}$$

The *C4.5 decision tree* introduces the information gain ratio (*GainRatio*) based on the information gain. It calculates the *GainRatio* for each attribute and chooses the attribute with the maximum *GainRatio* as the split node. The *GainRatio* is expressed by equation (4).

$$GainRatio(S, X) = \frac{Gain(S, X)}{SplitInfo(S, X)} \tag{4}$$

Where, $SplitInfo(S, X)$ is the split information of X to S and expressed as equation (5).

$$SplitInfo = -\sum_{i=1}^{n} p(S_i) \log_2 p(S_i) \tag{5}$$

Algorithm 1 describes how to use the *C4.5 decision tree* algorithm to identify application types. Since the algorithm is well known, we omit the details of the algorithm description. In this paper, we adopt the *Moore* dataset [26] to train and verify the algorithm. *Moore* dataset is popularly applies to application identification. The dataset contains 370000 flows composed by 10 application types shown in Table I [39]. We use *k-folder Cross Validation* [27] to test the effectiveness of ***Algorithm 1***. We divide the *Moore* dataset into K subsets and consider $K-1$ subsets as the training data. The selected feature is the port number. Each subset of the K subsets will be regarded as the test set, so the training and test process will be conducted K times. As a result, we can get K classification models. We find that when K is set to 10, the average identification accuracy is 99%. Results show that the training *decision tree* can be used to identify the application types.

For the misclassification application flows, on one hand, we can still provide best effort services as the traditional way. On the other, during the subsequent transmission, more features (such as, *packet size*, *packet number* and *inter-packet gap*, etc.) can be collected to further determine the application types. If the result is different to that judged at the flow setup phase, corresponding QoS forwarding rules will be updated.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2755768, IEEE Access

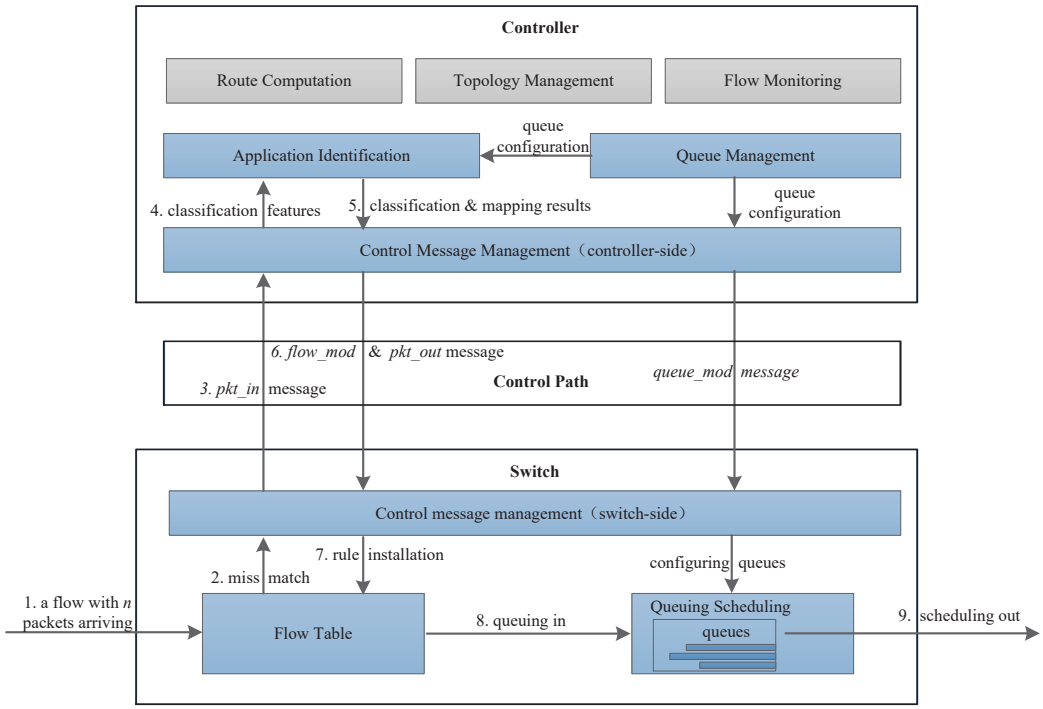

Fig. 1: System Framework of the SDN-based QoS Guaranteed Technique.

Algorithm 1 Application Types Identification with *C4.5 decision tree*

Input: training dataset of S, attribute set of A
1: $Tree \leftarrow \{\}$
2: **if** S belongs to the same class or A is Φ **then**
3: Terminate;
4: **end if**
5: **for** all attribute $X \in A$ **do**
6: compute $H(S|X)$, $SplitInfo(S, X)$ and $GainRatio(S, X)$
7: **end for**
8: X_{best}=$Maximum(GainRatio(S, X))$;
9: $Tree \leftarrow$ create a decision node with X_{best};
10: $S_{subset} \leftarrow$ split S into n different subsets based on X_{best};
11: **for** all S_{subset} **do**
12: $Tree_n \leftarrow C4.5$ (S_{subset}, $A - X_{best}$);
13: attach $Tree_n$ to the corresponding branch of the $Tree$;
14: **end for**
15: **return** $Tree$

TABLE I: Application Types and Corresponding Required QoS Levels

Application Types	QoS Levels
VOIP, GAME, SERVICES, CHAT	Expedited Forwarding
MULTIMEDIA, WEB, INTERACTIVE	Assured Forwarding
EMAIL, BULK, P2P	Best Effort

2) Application QoS-Levels Mapping: Different application types have different QoS requirements, which should be mapped to different QoS levels. Application flows mapped to the same QoS level are queued into the same queue. Through QoS levels mapping, the switch can provide differential services for the application flows. According to the specification of IEEE 802.1Q [28], we classify the QoS into three levels: *Expedited Forwarding (EF)*, *Assured Forwarding (AF)* and *Best Effort (BE) forwarding*. Considering the requirements of the applications in delay, jitter and bandwidth, we map the application types to the three required QoS levels. The mapping relations are shown in Table I. Real-time applications (VOIP, GAME, SERVICES and CHAT) are sensitive to delay and jitter. 2) Streaming application (MULTIMEDIA) focuses on unidirectional transmission and interactive applications (WEB and INTERACTIVE) are executed on the basis of the request-response model. Both of them are less sensitive to delay, but require bandwidth guarantee for availability. 4) Compared with the other applications, background applications have little demands on delay and bandwidth, e.g., Email, Bulk and P2P.

C. Queue Scheduling

Queue scheduling aims to queue the application flows into the queues with different QoS levels, and schedule the queued packets out of the output interfaces. In this paper, we design the queue scheduling algorithms based on *LLQ (Low Latency Queueing)* [29].

1) Queue Implementation: LLQ brings the ability to specify low latency behaviour for a traffic class. Each queue is equipped with a priority and application flows queued in the queue with higher priority will be scheduled out of the interface first. As shown in Fig. 2, we implement three queues at each output interface, including EF queue, AF queue and BE queue. *EF* queue (highest priority): packets queued in the EF queue are served with strict QoS guarantee. *AF* queue (middle priority): packets queued in the AF queue can get a certain degree of minimum bandwidth guarantee. *BE* queue (lowest priority): packets queued in the BE queue enjoy the best effort service without QoS guarantee.

Algorithm II describes how to queue the packets into the queues of an output interface. Assume that n packets of different applications arrive at a switch, and all the packets

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2755768, IEEE Access

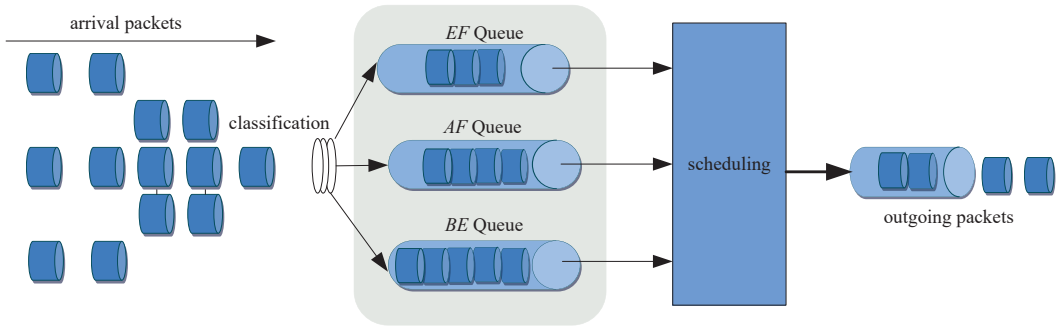

Fig. 2: Queuing Implementation at Each Output Interface.

are forwarded to the same output interface of the switch. According to the forwarding decisions, the packets belonging to different QoS levels are queued into different queues of the output interface. ***Algorithm III*** describes how to schedule the queued packets out of an output interface. If the EF queue is not empty, all the packets in the EF queue are scheduled out of the output interface (line 2∼4). Otherwise, the scheduler checks whether the AF queue is empty or not. If the AF queue is not empty, one packet in the AF queue is scheduled out of the output interface (line 5∼6). Then, the scheduler moves out from the current *while* loop and moves into the next *while* loop (line 7). If the AF queue is empty, the scheduler checks the BE queue. If the BE queue is not empty, one packet in the BE queue is scheduled out of the output interface (line 9).

Algorithm 2 Queue the Packets into the Queues of an Output Interface

Input: n packets arriving at the output interface of I
1: **for** each arrival packet p_i **do**
2: **if** p_i belongs to EF QoS level **then**
3: queue p_i to EF Queue;
4: **else**
5: **if** p_i belongs to AF QoS level **then**
6: queue p_i into AF Queue;
7: **else**
8: queue p_i into BE Queue;
9: **end if**
10: **end if**
11: **end for**

Algorithm 3 Schedule the Queued Packets out of an Output Interface

Input: EF, AF, BE
1: **while** TRUE **do**
2: **while** $isNotEmpty(EF)$ **do**
3: schedule out one packet in the EF Queue;
4: **end while**
5: **if** $isNotEmpty(AF)$ **then**
6: schedule out one packet in the AF Queue;
7: continue;
8: **else**
9: schedule out one packet in the BE Queue;
10: **end if**
11: **end while**

IV. Analysis based on Queuing Theory

To prove the effectiveness of the queue structure and scheduling algorithms, we conduct a theoretical analysis based on the queuing theory. The metrics of queue length, delay and throughput are used to evaluate the proposed queue scheduling technique.

A. Model Creation

A switch equips each output interface with three kinds of queues, i.e., *EF* queue, *AF* queue and *BE* queue. These queues provide differentiated services for various application flows. Assuming the arrival interval of application type i follows the exponential distribution with the parameter of λ_i (i=*EF*, *AF*, *BE*). The service time and the polling time also have the exponential distributions with the parameters of μ_i (i=*EF*, *AF*, *BE*) and θ. According to the *LLQ*-based scheduling algorithms, packets in the *EF* queue are always served first. Only when the *EF* queue is empty, the scheduler polls the *AF* queue and *BE* queue in turn. Assuming the total bandwidth is μ_{all}, thus, the bandwidth of *EF* queue is μ_{EF}= μ_{all}. If the minimum guaranteed bandwidth of *AF* queue is μ_{AF}, the available bandwidth of *BE* queue is μ_{BE}=μ_{all}-μ_{AF}.

Correspondingly, the service state is divided into three types, including serving the *EF* queue, serving the *AF* queue and serving the *BE* queue. We use $Q_{EF}(t)$, $Q_{AF}(t)$ and $Q_{BE}(t)$ to represent the number of packets in the three kinds of queues respectively at time t. As expressed by equation (6), $S(t)$ denotes the state at time t, i.e., the queue the scheduler is serving at time t.

$$S(t) = \begin{cases} 0, & serving\ the\ EF\ queue \\ 1, & serving\ the\ AF\ queue \\ 2, & serving\ the\ BE\ queue \end{cases} \tag{6}$$

The state space of $\{S(t), Q_{EF}(t), Q_{AF}(t), Q_{BE}(t)\}$ is depicted as equation (7). In equation (7), m_{EF}, m_{AF} and m_{BE} represent the maximum length of each type of queue respectively.

$$\begin{aligned} &\{(f,i,j,k)|0 \le i \le m_{EF}, 0 \le j \le m_{AF}, \\ &0 \le k \le m_{BE}, f \in \{0,1,2\}\} \end{aligned} \tag{7}$$

The state transition rate matrix of *EF* queue is shown in equation (8). In equation (8), each element of Q is a $(m_{EF}+1)(m_{AF}+1)(m_{BE}+1)$ order matrix. A_1, A_2 and B are the transition rate matrices for state $0 \to 1$, state $1 \to 2$ and state

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2755768, IEEE Access

$2 \rightarrow 1$ respectively. A_0 is the transition rate matrix for state $0 \rightarrow 0$.

$$Q = \begin{matrix} 0 \\ 1 \\ 2 \end{matrix} \begin{pmatrix} A_0 & B & \\ & A_1 & B \\ B & & A_2 \end{pmatrix} \tag{8}$$

The value of B is shown in equation (9). When the state transition $0 \rightarrow 0$ happens, the scheduler serves the *EF* queue. The value of A_0 is shown in equation (10).

$$B = diag(\theta ... \theta)_{(m_{EF}+1)(m_{AF}+1)(m_{BE}+1)} \tag{9}$$

$$A_0 = \begin{matrix} 0 \\ 1 \\ \vdots \\ m_{EF} \end{matrix} \begin{pmatrix} C_0 & E & & & \\ D & C_0 & E & & \\ & \ddots & \ddots & \ddots & \\ & & & D & C_0 \end{pmatrix} \tag{10}$$

Where, the values of E, D and C_0 are shown in equation (11), equation (12) and equation (13) respectively.

$$E = diag(\lambda_{EF} ... \lambda_{EF})_{(m_{AF}+1)(m_{BE}+1)} \tag{11}$$

$$D = diag(\mu_{EF} ... \mu_{EF})_{(m_{AF}+1)(m_{BE}+1)} \tag{12}$$

$$C_0 = \begin{matrix} 0 \\ 1 \\ \vdots \\ m_{AF} \end{matrix} \begin{pmatrix} F_0 & G & & \\ & F_0 & G & \\ & & \ddots & \ddots \\ & & & F_0 \end{pmatrix} \tag{13}$$

Where, F_0 and G are depicted in equation (14) and equation (15).

$$G = diag(\lambda_{AF} ... \lambda_{AF})_{(m_{BE}+1)(m_{BE}+1)} \tag{15}$$

When the state transition $0 \rightarrow 1$ happens, the scheduler goes to serve the *AF* queue from the *EF* queue. The value of A_1 is shown in equation (16).

$$A_1 = \begin{matrix} 0 \\ 1 \\ \vdots \\ m_{EF} \end{matrix} \begin{pmatrix} C_1 & E & & \\ & C_1 & E & \\ & & \ddots & \ddots \\ & & & C_1 \end{pmatrix} \tag{16}$$

Where, C_1 is shown in equation (17).

$$C_1 = \begin{matrix} 0 \\ 1 \\ \vdots \\ m_{AF} \end{matrix} \begin{pmatrix} F_1 & G & & \\ H & F_1 & G & \\ & \ddots & \ddots & \ddots \\ & & H & F_1 \end{pmatrix} \tag{17}$$

Where, the value of G is shown in equation (15). F_1 and H are depicted as equation (18) and equation (19).

$$H = diag(\mu_{AF} ... \mu_{AF})_{(m_{BE}+1)(m_{BE}+1)} \tag{19}$$

Similarly, when the state transition $0 \rightarrow 1$ happens, the scheduler goes to serve the *BE* queue from the *AF* queue. The value of A_2 is shown in equation (20).

$$A_2 = \begin{matrix} 0 \\ 1 \\ \vdots \\ m_{EF} \end{matrix} \begin{pmatrix} C_2 & E & & \\ & C_2 & E & \\ & & \ddots & \ddots \\ & & & C_2 \end{pmatrix} \tag{20}$$

Where, C_2 is shown in equation (21). The value of G is shown in equation (15) and the value of F_2 is depicted as equation (22).

$$C_2 = \begin{matrix} 0 \\ 1 \\ \vdots \\ m_{AF} \end{matrix} \begin{pmatrix} F_2 & G & & \\ & F_2 & G & \\ & & \ddots & \ddots \\ & & & F_2 \end{pmatrix} \tag{21}$$

B. Model Solving

According to the state transition matrices, we try to solve the queue system. Equation (23) shows the steady-state distribution of the queuing system.

$$\Pi \cdot Q = \vec{0} \tag{23}$$

The steady-state probabilities of each state are expressed by equation (24).

$$\begin{aligned} \Pi = \{\pi_{f,i,j,k} | 0 \leq i \leq m_{EF}, 0 \leq j \leq m_{AF}, \\ 0 \leq k \leq m_{BE}, f \in \{0,1,2\}\} \end{aligned} \tag{24}$$

Through calculating the steady-state probabilities, we can get the performance metrics of the queuing system, including 1) the average queue length denotes the number of the packets waiting to be served in the queue; 2) the throughput refers to the packet forwarding rate of the queue without packet loss; 3) the average waiting time starts from the packet being queued into the queue to being scheduled out of the queue. Under the steady state, the average queue length of the *EF* queue is shown in equation (25).

$$L_{EF} = \sum_{f=0}^{2} \sum_{i=0}^{m_{EF}} \sum_{j=0}^{m_{AF}} \sum_{k=0}^{m_{BE}} i \cdot \pi_{f,i,j,k} \tag{25}$$

The throughput of the *EF* queue and its average waiting time are shown in equation (26) and equation (27) respectively.

$$\tau_{EF} = \sum_{i=0}^{m_{EF}} \sum_{j=0}^{m_{AF}} \sum_{k=0}^{m_{BE}} \mu_{EF} \cdot \pi_{0,i,j,k} \tag{26}$$

$$W_{EF} = \frac{L_{EF}}{\tau_{EF}} \tag{27}$$

similarly, the performance values of the *AF* queue and *BE* queue are shown in equation (28) $\sim$ equation (33).

$$L_{AF} = \sum_{f=0}^{2} \sum_{i=0}^{m_{EF}} \sum_{j=0}^{m_{AF}} \sum_{k=0}^{m_{BE}} j \cdot \pi_{f,i,j,k} \tag{28}$$

$$\tau_{AF} = \sum_{i=0}^{m_{EF}} \sum_{j=0}^{m_{AF}} \sum_{k=0}^{m_{BE}} \mu_{AF} \cdot \pi_{1,i,j,k} \tag{29}$$

$$W_{AF} = \frac{L_{AF}}{\tau_{AF}} \tag{30}$$

$$L_{BE} = \sum_{f=0}^{2} \sum_{i=0}^{m_{EF}} \sum_{j=0}^{m_{AF}} \sum_{k=0}^{m_{BE}} k \cdot \pi_{f,i,j,k} \tag{31}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2755768, IEEE Access

$$F_0 = \begin{matrix} 0 \\ 1 \\ \vdots \\ m_{BE} \end{matrix} \begin{Bmatrix} -(\theta+\lambda_{EF}+\lambda_{AF}+\lambda_{BE}) & \lambda_{BE} & & \\ & -(\theta+\lambda_{EF}+\lambda_{AF}+\lambda_{BE}+\mu_{EF}) & \lambda_{BE} & \\ & \ddots & \ddots & \ddots \\ & & & -(\theta+\mu_{EF}) \end{Bmatrix} \tag{14}$$

$$F_1 = \begin{matrix} 0 \\ 1 \\ \vdots \\ m_{BE} \end{matrix} \begin{Bmatrix} -(\theta+\lambda_{EF}+\lambda_{AF}+\lambda_{BE}) & \lambda_{BE} & & \\ & -(\theta+\lambda_{EF}+\lambda_{AF}+\lambda_{BE}+\mu_{AF}) & \lambda_{BE} & \\ & \ddots & \ddots & \ddots \\ & & & -(\theta+\mu_{AF}) \end{Bmatrix} \tag{18}$$

$$F_2 = \begin{matrix} 0 \\ 1 \\ \vdots \\ m_{BE} \end{matrix} \begin{Bmatrix} -(\theta+\lambda_{EF}+\lambda_{AF}+\lambda_{BE}) & \lambda_{BE} & & \\ \mu_{BE} & -(\theta+\lambda_{EF}+\lambda_{AF}+\lambda_{BE}+\mu_{BE}) & \lambda_{BE} & \\ & \ddots & \ddots & \ddots \\ & & \mu_{BE} & -\theta \end{Bmatrix} \tag{22}$$

$$\tau_{BE} = \sum_{i=0}^{m_{EF}} \sum_{j=0}^{m_{AF}} \sum_{k=0}^{m_{BE}} \mu_{BE} \cdot \pi_{2,i,j,k} \tag{32}$$

$$W_{BE} = \frac{L_{BE}}{\tau_{BE}} \tag{33}$$

C. Numerical Analysis

This section conducts a numerical analysis on the performance metrics of the queue system. Assuming the total bandwidth $\mu_{all} = 0.1\,Gbps$, the average packet size $p = 1000\,Bytes$, and the parameters of $\lambda_{EF} = \lambda_{AF} = \lambda_{BE} = 0.0333\,Gbps$, $m_{EF} = m_{AF} = m_{BE} = 5$ and $\theta = 0.01\,s$. Then, we can get $\mu_{EF} = \mu_{all} = 0.1\,Gbps$. At the initial phase, the bandwidth of *AF* queue is $\mu_{AF} = 0.04\,Gbps$, and the bandwidth of BE queue is $\mu_{BE} = \mu_{all} - \mu_{AF} = 0.06\,Gbps$.

1) Average Queue Length : As shown in Fig. 3, with the increase of the bandwidth assigned to *AF* queue, the average length of *AF* queue decreases gradually. Correspondingly, the bandwidth assigned to the *BE* queue decreases gradually, because only the remaining bandwidth is available for the *BE* queue. Less assigned bandwidth means low packet forwarding efficiency, causing many queued packets waiting for being scheduled out of the *BE* queue. As a result, the average length of *BE* queue increases gradually. Since the *EF* queue has the highest priority to use the bandwidth, the average length of *EF* queue is always kept in a stable state.

2) Average Throughput: As shown in Fig. 4, the average throughput of *AF* queue is lower than that of *BE* queue at the beginning. With the increase of the bandwidth assigned to *AF* queue, the average throughput of *AF* queue presents a growth, and the average throughput of *BE* queue decreases gradually. The *EF* queue has the highest priority of bandwidth utilization, so it keeps the high throughput and is not affected by the other two queues.

3) Average Waiting Time: As shown in Fig. 5, the average queue waiting time of *AF* queue presents a decrease with the increase of the bandwidth assigned to *AF* queue. Conversely, the average queue waiting time of *BE* queue presents a obvious growth when more bandwidth can be utilized. Packets queued in the *EF* queue are scheduled out of the interface with the highest priority, so the average queue waiting time of *EF* queue keeps at a low level all the time.

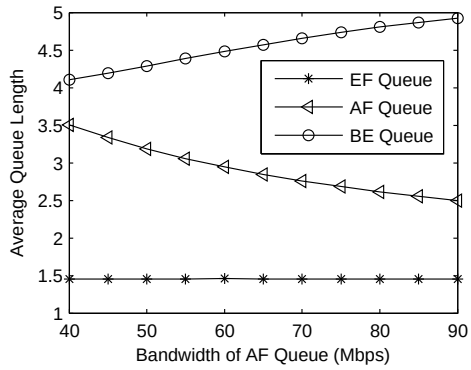

Fig. 3: Average Queue Length under Different Bandwidth of *AF* Queue

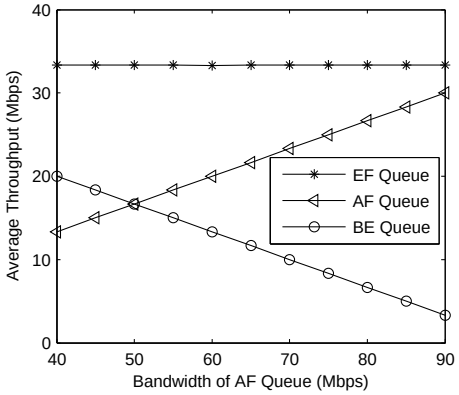

Fig. 4: Average Throughput under Different Bandwidth of *AF* Queue

In summary, the *LLQ*-based scheduling algorithms can

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2755768, IEEE Access

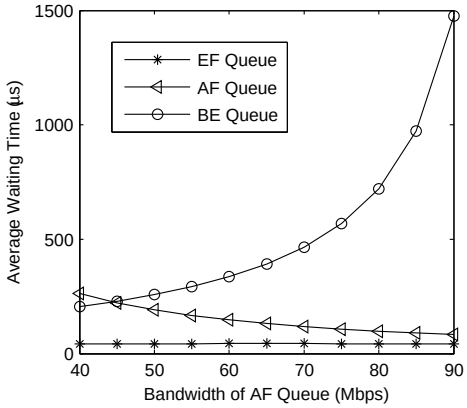

Fig. 5: Average Waiting Time under Different Bandwidth of *AF* Queue

provide strict QoS guarantee for the application flows queued in the *EF* queue. In addition to this, the algorithm can supply a certain degree of minimum bandwidth guarantee for the application flows queued in the *AF* queue. As a sacrifice, performance cannot be guaranteed for the packets queued in the *BE* queue. However packets queued in the *BE* are generic data flows, which are not sensitive to delay, jitter and bandwidth, etc. Theoretical analysis results prove the effectiveness of the *LLQ*-based scheduling algorithms.

V. Experimental Evaluation

In this section, we first describe the experimental environment. Then, we evaluate the effectiveness of the proposed SDN-based QoS technique.

A. Experiment Description

Fig. 6 shows our experiments setup. *Open vSwitch*(OVS) [30] is an open source *OpenFlow* virtual switch. *Floodlight* [31] is an open source SDN controller. We run *OVS* and *Floodlight* on two commodity PCs respectively. Table I shows the configurations of the experimental devices. $Host_1$ and $Host_2$ connect to OVS with 100Mbps interfaces. We run *pktgen* [32] on $Host_1$ to generate traffic at the rates of 5Mbps - 100Mbps with the Ethernet frame size of 1000 Bytes. We run *tcpdump* [33] to listen on the interfaces that are connected to the hosts and the controller respectively.

In this experiment, $Host_1$ sends three kinds of flows to $Host_2$, including the voice flow (delay-sensitive), the video flow (bandwidth-hungry) and the generic data flow. These three application flows are sent out in cross sequences. We implement the *C4.5*-based application identification technique in the *Floodlight* and the *LLQ*-based queue scheduling technique in the *Open vSwitch*. The three types of flows will be mapped to the *EF*, *AF* and *BE* QoS levels and correspondingly, be queued into the *EF* queue, *AF* queue and *BE* queue respectively. By default, each output interface is configured with a *FIFO* queue. We conduct a comparison study between the default queue technique and the *LLQ*-based queue technique implemented in this study.

TABLE II: Configurations of the Experimental Devices

Device Name	CPU	Cores	RAM	NIC
$Host_1$	3.3GHZ	4	4GB	1×100Mbps
$Host_2$	3.3GHZ	4	4GB	1×100Mbps
Open vSwitch	3.3GHZ	4	4GB	3×100Mbps
Floodlight	3.3GHZ	4	4GB	1×100Mbps

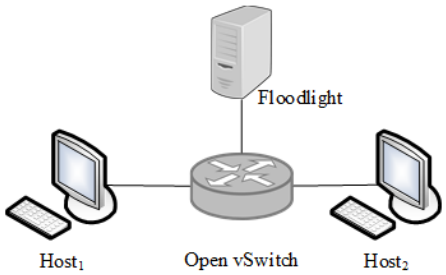

Fig. 6: Topography of the Experimental Platform.

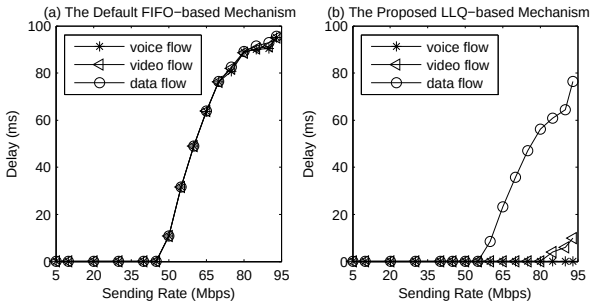

Fig. 7: Delay Variations across Different Sending Rates: (a) The Default *FIFO* Mechanism. (b) The Proposed *LLQ*-based Mechanism

B. Effectiveness Evaluation

Fig. 7(a) shows the delay variations of the flows using the default *FIFO* queue. With the increase of the sending rate, the three types of flows compete for the bandwidth intensively. When the total sending rate reaches to the maximum bandwidth of the switch interface, the delays of the three types of flows increase quickly. Since the default *FIFO* queue mechanism does not distinguish the differences of the flows, the delays of the three types of flows present similar patterns. Neither the voice flow nor the video flow is guaranteed with a tolerant delay. The *LLQ*-based queue mechanism treats the flows differently. As shown in Fig. 7(b), the voice flow is guaranteed with a small delay. It is not affected by the sending rate, because it is mapped to the *EF* QoS level, which has the highest priority to use the bandwidth. The video flow is mapped to the *AF* QoS level and guaranteed with the minimum bandwidth of 30*Mbps*, so the delay of this flow is less affected. However, when the sending rate exceeds 80*Mbps*, the delay of the video flow starts to be influenced slightly. The generic data flow is not guaranteed and the delay presents a similar pattern with the default *FIFO* queue mechanism.

We then conduct further experimental analysis about the proposed technique under different bandwidth conditions. 1) No-congestion: the total bandwidth of the interface is sufficient

for the requirements of the three flows. 2) Congestion: the bandwidth requirements of the three flows approach to the total bandwidth limitation of the interface, i.e. 100*Mbps*. Metrics of delay, jitter and throughput are used to evaluate the performance of the three types of application flows.

1) Delay: a) No-Congestion. As shown in Fig. 8(a) $\sim$ Fig. 8(c), when the interface is not congested, both the default and proposed mechanisms can meet the QoS requirements of the application flows. Using the default *FIFO* mechanism, the average delays of the three types of flows are $6.31\mu s$, $6.18\mu s$ and $6.24\mu s$ respectively. While using the *LLQ*-based mechanism, the average delays of the three types of flows are 4.49 μs, $4.43\mu s$ and $4.46\mu s$ respectively. The delays of the three types of flows are reduced by 28.8%, 28.3% and 28.5% on average. For the default mechanism, the sending rate is small, the flows do not need to compete for the bandwidth. Packets queued into the queue can be scheduled out quickly without any queue waiting time. So the default mechanism can also meet the QoS requirements of these application flows. For the *LLQ*-based mechanism, the bandwidth assigned to each queue is larger than the sending rate of the application flows, so the bandwidth is enough to meet the QoS requirements. However, the proposed mechanism creates three queues for each output interface. Multi-queues mechanism reduces the complexity of packet processing, so the *LLQ*-based (multi-queues) mechanism performs a little better than the default *FIFO* (single-queue) mechanism.

b) Congestion. As shown in Fig. 9(a) $\sim$ Fig. 9(c), the proposed *LLQ*-based mechanism obviously outperforms the default *FIFO* mechanism when the interface is congested. Using the default mechanism, more and more packets are congested in the queue over the time. As a result, the average delays of the three types of flows present fast growth. While using the proposed mechanism, there are there types of queues for each interface. Each queue is marked with a QoS level, and serves the application flows that are mapped to this QoS level. The voice flow has the highest priority to use the bandwidth, so the delay is small and keeps stable. The video flow is provided with a certain degree of minimum bandwidth guarantee, so the delay increases slowly. The generic data flow is served in the best effort way, so the delay increases quickly. However, its growth rate is smaller than that of the default mechanism. Under the congestion condition, the average delays of the three types of flows are reduced by 99.996%, 90.66% and 27.84% on average respectively. In view of delay, the voice flow is strictly guaranteed and the video flow can also be guaranteed to a great extent.

2) Jitter: a) No-Congestion. As shown in Fig. 10(a) $\sim$ Fig. 10(c), when the interface is not congested, the jitters of the three types of flows are small. However, the proposed mechanism performs a little better than the default mechanism. For the default mechanism, the average jitters of the three types of flows are 3.02 μs, 3.93 μs and 2.20 μs respectively. While for the proposed mechanism, the average jitters of the three types of flows are 1.02 μs, 1.12 μs and 1.09 μs respectively. The jitters can be reduced by 66.17%, 61.61% and 50.41% on average respectively. The reason is that the proposed mechanism adopts multi-queues to serve different application flows, which can avoid the inference when all the application flows are queued into a single queue.

b) Congestion. As shown in Fig. 11(a) $\sim$ Fig. 11(c), when the interface is congested, the proposed mechanism presents obvious advantages in view of jitter. Compared with the default mechanism, the jitters of the three types of flows can be reduced by 99.85%, 91.18% and 34.13% on average respectively.

3) Throughput: a) No-Congestion. As shown in Fig. 12(a) $\sim$ Fig. 12(c), when the interface is not congested, the throughputs of the two mechanisms present similar patterns. This is because the interface can provide sufficient bandwidth for each kind of application flow. Packets queued into the queues can be scheduled out quickly by both of the default mechanism and the proposed mechanism.

b) Congestion. As shown in Fig. 13(a) $\sim$ Fig. 13(c), when the interface is congested, the proposed mechanism can achieve greater throughput than the default mechanism. Compared with the default mechanism, the throughputs of the three types of flows can be increased by 90.17%, 76.06% and 18.5% on average respectively.

VI. Conclusions and Future Remarks

In this paper, we propose and evaluate a SDN-based QoS technique for cloud applications. 1) We first design the architecture of the SDN-based QoS technique, which combines application identification with queue scheduling. 2) Then, we implement an application identification method in the SDN controller. It can identify application types and map each type of application to a required QoS level. 3) Thirdly, we implement a queue scheduling method in the switch. We also prove the effectiveness of the queue scheduling technique by a theoretical analysis. 4) At last, we evaluate the proposed SDN-based QoS technique through an in-depth experimental analysis.

In the future, more queues will be created at each output interface, and the proposed SDN-based QoS technique will be used to provide end-to-end QoS routing and multipath routing. In addition, more features will be extracted to classify the application flows at finer granularity. We will further evaluate the efficacy of proposed technique through commodity SDN switches with Gigabit Ethernet, and try to extend the use of the proposed QoS technique in the area of mobile computing [37, 38].

Acknowledgements

The authors would like to thank Dr. Tian Pan for his kind help and constructive comments. This work is supported by RGC General Research Fund (GRF) with RGC No. PolyU 152244/15E; the National Natural Science Foundation of China under Key Grant No. 61332004 and Grant Nos. 61602105 and 61572123; China Postdoctoral Science Foundation under Grant No. 2016M601323; the Fundamental Research Funds for the Central Universities Project under Grant No. N150403007; CERNET Innovation Project under Grant No. NGII20160126.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2755768, IEEE Access

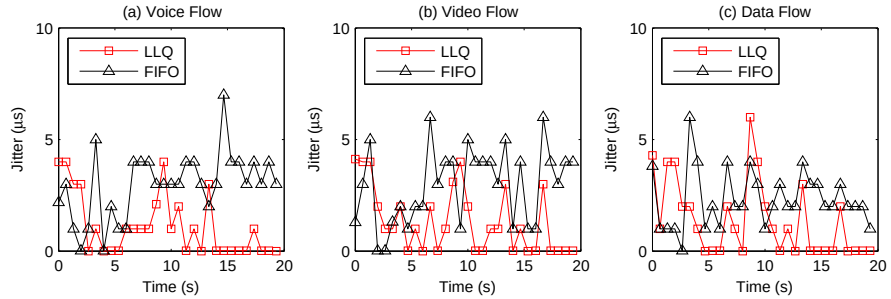

Fig. 8: No-congestion: Delay Variations of Different Application Flows: (a) voice flow; (b) video flow; (c) generic data flow.

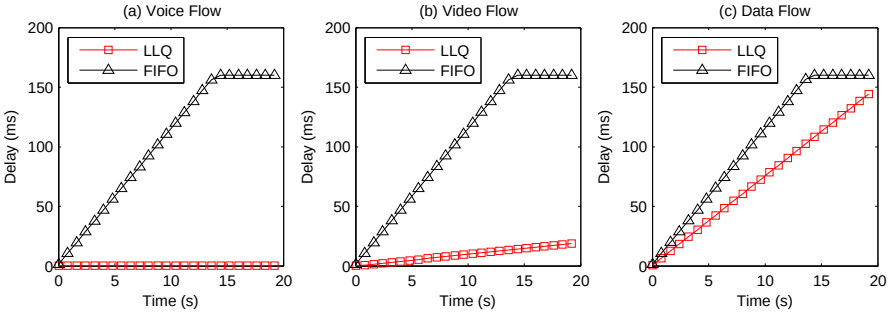

Fig. 9: Congestion: Delay Variations of Different Application Flows: (a) voice flow; (b) video flow; (c) generic data flow.

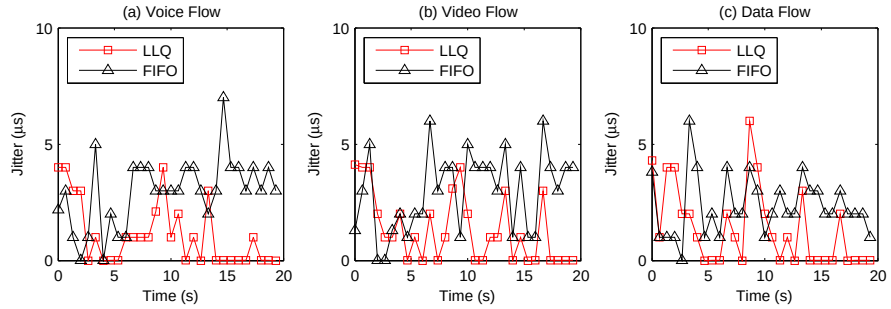

Fig. 10: No-congestion: Jitter Variations of Different Application Flows: (a) voice flow; (b) video flow; (c) generic data flow.

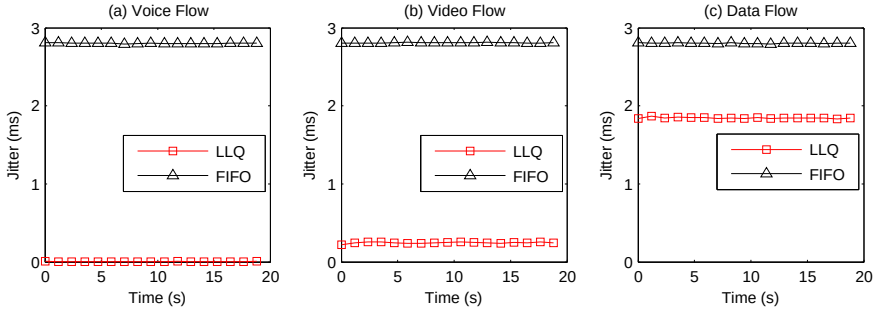

Fig. 11: Congestion: Jitter Variations of Different Application Flows: (a) voice flow; (b) video flow; (c) generic data flow.

REFERENCES

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In Proceedings of the 7th USENIX Symposium on Networked System Design and Implementation (NSDI), 2010: 19-19.

[2] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: Fine grained traffic engineering for data centers. In Proceedings of the 7th ACM COnference on emerging Networking EXperiments and Technologies (CoNEXT), 2011: 8.

[3] R. Braden, D. Clark and S. Shenker. Integrated Services in the Internet Architecture: an Overview. IETF RFC1633, (1994) June, pp. 2-3.

[4] S. Blake, D. Black, M. Carlson, et al. An Architecture for Differentiated Service. IETF FC2475, (1998) December, pp. 2-9.

[5] E. Rosenetal. Multiprotocol Label Switching Architecture. IETF R-FC3031, January, 2001: 2-3.

[6] A. V. Akella, K. Xiong. Quality of service (QoS)-guaranteed network resource allocation via software defined networking (SDN). In Proceedings of the IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC), 2014: 7-13.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2017.2755768, IEEE Access

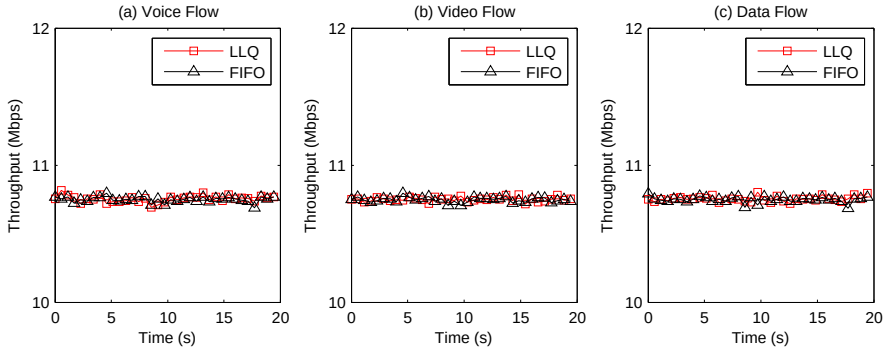

Fig. 12: No-congestion: Throughput Variations of Different Application Flows: (a) voice flow; (b) video flow; (c) generic data flow.

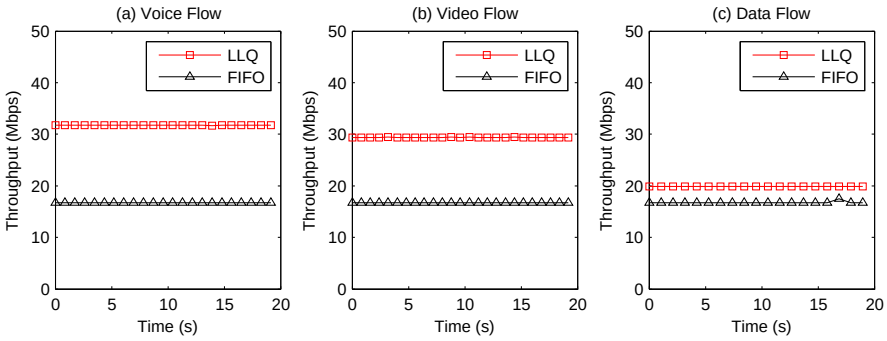

Fig. 13: Congestion: Throughput of Different Application Flows: (a) voice flow; (b) video flow; (c) generic data flow.

[7] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, G. Wang. Meridian: an SDN platform for cloud network services. IEEE Communications Magazine, 51(2), 2013: 120-127.

[8] S. Tomovic, N.Prasad, I. Radusinovic. SDN control framework for QoS provisioning. In Telecommunications Forum Telfor (TELFOR), 2014: 111-114).

[9] J. Yan, H. Zhang, Q. Shuai, B. Liu, X. Guo. HiQoS: An SDN-based multipath QoS solution. China Communications, 12(5), 2015: 123-133.

[10] K. Jeong, J. Kim, Y. T. Kim. QoS-aware network operating system for software defined networking with generalized OpenFlows. In 2012 IEEE Network Operations and Management Symposium, 2012: 1167-1174.

[11] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker. NOX: towards an operating system for networks. ACM SIGCOMM Computer Communication Review, 2008, 38(3): 105-110

[12] R. Wallner, R. Cannistra. An SDN approach: quality of service using big switch's floodlight open-source controller. Proceedings of the Asia-Pacific Advanced Network, 2013, 35: 14-19.

[13] Floodlight Home Page, http://docs.projectFloodlight.org

[14] A. Ishimori, F. Farias, E. Cerqueira, A. Abelém. Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking. In 2013 Second European Workshop on Software Defined Networks, 2013: 81-86.

[15] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer, J. A. García-Espín. An opennaas based sdn framework for dynamic qos control. In 2013 IEEE SDN for Future Networks and Services (SDN4FNS), 2013: 1-7.

[16] OpenNaaS, http://opennaas.org/

[17] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, P. Tran-Gia. Sdn-based application-aware networking on the example of youtube video streaming. In 2013 Second European Workshop on Software Defined Networks, 2013: 87-92.

[18] M. F. Bari, S. R.Chowdhury, R. Ahmed, R. Boutaba. PolicyCop: an autonomic QoS policy enforcement framework for software defined networks. In 2013 IEEE SDN for Future Networks and Services (SDN4FNS), 2013: 1-7.

[19] S. Gorlatch, T. Humernbrum, F. Glinka. Improving QoS in real-time internet applications: from best-effort to Software-Defined Networks. In 2014 International Conference on Computing, Networking and Communications (ICNC), 2014: 189-193.

[20] M. S. Seddiki, M.Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, Y. Q. Song. FlowQoS: QoS for the rest of us. In Proceedings of the third workshop on Hot topics in software defined networking, 2014: 207-208.

[21] J. Yan, H. Zhang, Q. Shuai, B. Liu, X. Guo. HiQoS: An SDN-based multipath QoS solution. China Communications, 12(5), 2015: 123-133.

[22] C. Sieber, A. Blenk, D. Hock, M. Scheib, T. Höhn, S. Köhler, W. Kellerer. Network configuration with quality of service abstractions for SDN and legacy networks. In 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015: 1135-1136.

[23] S. Dwarakanathan, L. Bass, L. Zhu. Cloud Application HA using SDN to ensure QoS. In IEEE 2015 8th International Conference on Cloud Computing, 2015: 1003-1007.

[24] D. Adami, L. Donatini, S. Giordano, M. Pagano. A network control application enabling Software-Defined Quality of Service. In 2015 IEEE International Conference on Communications (ICC), 2015: 6074-6079.

[25] S. Ruggieri. Efficient C4. 5 [classification algorithm]. IEEE transactions on knowledge and data engineering, 2002, 14(2): 438-444.

[26] A. W. Moore. Discrete content-based classification a data set. Technical Report, Intel Research, 2005.

[27] P. Refaeilzadeh, L. Tang, H. Liu. Cross-validation. In Encyclopedia of database systems. Springer US, 2009: 532-538.

[28] IEEE 802.1Q-2014. IEEE Standard for Local and metropolitan area networks-Bridges and Bridged Networks. https://standards.ieee.org/findstds/standard/802.1Q-2014.html.

[29] Low Latency Queueing - LLQ. Class-Based Weighted Fair Queueing. http://www.cisco.com/c/en/us/td/docs/ios/12_0s/feature/guide/fsllq26.html.

[30] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado. The Design and Implementation of Open vSwitch. In NSDI, 2015.

[31] Floodlight Home Page, http://docs.projectFloodlight.org

[32] R. Olsson. Pktgen the Linux Packet Generator. In Ottawa Linux Symposium, 2005.

[33] Tcpdump. http://www.tcpdump.org/

[34] OpenFlow Switch Specification. https://www.opennetworking.org

[35] M. Devera. HTB Linux queuing discipline manual - userguide. http://luxik.cdi.cz/ devik/qos/htb/manual/userg.htm.

[36] K. Rechert, P. McHardy, M. Brown. HFSC Scheduling with Linux. http://linuxip.net/articles/hfsc.en.

[37] H. Meng, Y. Zhu, R. Deng. Optimal Computing Resource Management Based on Utility Maximization in Mobile Crowdsourcing. Wireless Communications and Mobile Computing.

[38] R. Deng, R. Lu, C. Lai, T. H. Luan, H. Liang. Optimal Workload Allocation in Fog-Cloud Computing Towards Balanced Delay and Power Consumption. IEEE Internet of Things Journal, 3(6): 1171-1181, Dec. 2016.

[39] A. W. Moore, D. Zuev. Internet traffic classification using bayesian analysis techniques. In ACM SIGMETRICS Performance Evaluation Review, 33(1), 2005: 50-60.