Performance Study of Dynamic QoS Management for OpenFlow-enabled SDN Switches

Raphael Durner, Andreas Blenk, Wolfgang Kellerer
Chair of Communication Networks
Department of Electrical and Computer Engineering
Technische Universität München
München, Germany
Email: {r.durner, andreas.blenk, wolfgang.kellerer}@tum.de

***Abstract*—Software-defined Networking (SDN) is a promising and powerful concept to introduce new dimensions of flexibility and adaptability in today's communication networks. In particular, the realization of Quality of Service (QoS) concepts becomes possible in a flexible and dynamic manner with SDN. Although concepts of QoS are well researched, they were not realized in communication networks due to high implementation complexity and realization costs. Using SDN to realize QoS mechanisms enables emerging concepts, such as application-aware resource management solutions. These emerging concepts demand latency or data rate guarantees for end-user applications, e.g., video streaming or gaming. However, the impact of dynamic QoS management on network traffic has not been studied in detail yet. This paper provides a first study of the impact on dynamic QoS mechanisms and their realizations for OpenFlow-enabled SDN switches. Although SDN and, in particular, OpenFlow as one dominant realization claim to provide a standardized interface to control network traffic, our measurement results show a noticeable diversity for different OpenFlow switches. In detail, our investigations reveal a severe impact on the performance of TCP-based network traffic among different switches. These observations of switch diversity may provide SDN application developers insights when realizing QoS concepts in an SDN-based network.**

I. Introduction

Today's communication networks are still suffering from their inflexibility and incapability to adapt to the demands of transmitted network traffic. However, as today's applications, such as high definition video streaming or gaming, have high and strict demands w.r.t. latency and data rate, a flexible and dynamic resource management is a fundamental basis of future network concepts. Furthermore, with the increasing amount of network traffic, communication network operators have to introduce new ways of flexible and dynamic resource management in order to provide the best possible performance for their networks.

With the emergence of Software-Defined Networking (SDN), a new flexible network operation and control is possible. Such a flexible operation and control demands the realization of well established Quality of Service (QoS) concepts. SDN promises to provide a powerful way to introduce QoS concepts in today's communication networks. Furthermore, OpenFlow as the best-known SDN standard so far defines a standard protocol for network control. Using SDN and OpenFlow for realizing new resource management, existing studies, such as [1], [2], [3], have shown that applications benefit from a fast and frequent change of network resource allocations, which we call dynamic QoS management. For instance, for progressive video streaming applications, [2] showed that changing the queue assignment of video flows based on the currently buffered playtime avoids stalling. Stalling has a high negative impact on perceived application quality of network users.

All prior research on dynamic QoS management in SDN-based networks, e.g., [1], [2], [3], has neglected the diversity of existing hardware, i.e., switch diversity. However, in existing work [4], [5], it was shown that switches from different vendors show different behaviors, e.g., for flow installation times or for different order of OpenFlow rule operations. Ignoring such switch diversities may lead to significant performance degradations in SDN networks. Accordingly, the diverse behavior has to be taken into account when designing SDN applications in general, and when designing SDN applications based on dynamic QoS management in particular.

In this paper, we study effects on network traffic when applying dynamic QoS management in OpenFlow-based SDN networks. We measure the impact on TCP network flows when changing the queue assignment of multiple flows at runtime. Our measurement study is done for different configurations of two fundamental quality of service concepts, namely priority queuing and bandwidth guaranteeing, which are deployed on three switches. The switches are the NEC PF5250, the P3290 from PICA8, and the software switch Open vSwitch (OVS) [6]. Furthermore, as the OVS switch is implemented in software, it provides detailed information on the used queuing disciplines and the used queue implementations, e.g., First-In-First-Out (FIFO) queues or Stochastic Fairness Queuing (SFQ) queues. Using the OVS behavior as reference allows us to draw conclusions about the used queue implementations of the hardware switches, which is only rarely publicly available.

The rest of this paper is organized as follows. Section II describes work related to our study. We give details about the measurement setup and execution in Section III. The results of our study are presented in Section IV. Finally, conclusions are drawn in Section V.

II. Related Work

Previous work on providing QoS guarantees using OpenFlow can be partitioned in three categories. First, studies deploying dynamic QoS in an SDN environment [1], [2], [7]. Second, studies on switch diversity [4], [5], [8], [9]. Third, research on network performance resulting from QoS with OpenFlow-enabled switches [10], [11].

The OpenFlow-assisted Quality of Experience (QoE) Fairness Framework (QFF) [1] provides user-level fairness for adaptive-video streaming (DASH). QFF guarantees QoS in the network in order to provide users suitable and more stable bandwidths. As DASH also utilizes long living TCP flows, we expect the appearance of the same effects as we show in this paper with the use of QFF. The study [1] only considers effects on the video application, while we measure effects in the network.

[3] introduces a QoS controller prototype that guarantees end-to-end QoS by routing the flows based on performance requirements. With the arrival of a new flow, the controller calculates the resource allocation and installs the necessary rules with QoS guarantees. Although the controller prototype is examined using one hardware and one software switch, the focus of the analysis lies on the QoS control framework and not on the effects that these guarantees cause when they are applied dynamically.

VMPatrol [7] employs a framework that limits the used portion of the available bandwidth when migrating Virtual Machines (VMs) in a cloud environment, for the sake of the productive traffic. The conclusion is that QoS mechanisms can decrease the adverse impact of VM migrations on other network flows. This is an important use case for dynamic QoS with OpenFlow, as many SDN researchers currently focus on virtualized environments.

The study [2] examines bandwidth management depending on the currently buffered playback time of YouTube progressive video streams. The study also measures and compares the impact of different queuing strategies. Different queuing strategies are found to lead to varying buffered playback times of the videos. In contrast to our study, [2] does not use OpenFlow and does not compare different switches. Additionally, [2] focuses on resource management, while we concentrate on detailed measurements of different dynamic QoS settings.

Tango [4] is a switch probing engine that aims at countering problems that result from switch diversity. In contrast to our approach, Tango analyzes the diversity regarding flow installation and manipulation timing in the control plane, while we focus on the effects that rule changing has on ongoing flows in the data plane. However, we believe that our observations can also be integrated in a framework such as Tango.

Recent studies [8], [5], [9] analyze the diversity of switches regarding the SDN control plane behavior. For instance, [5], [9] analyze rule installations in combination with barrier replies that should confirm the rule installation and show that some switches confirm the installation of rules prematurely. The barrier reply is sent out before the OF rules are actually active. Additionally, the relationship between the number of flow rules and data plane performance, as well as the impact of rule priorities is measured. [8] measures the processing capabilities of hardware and software switches for specific OF message rates.

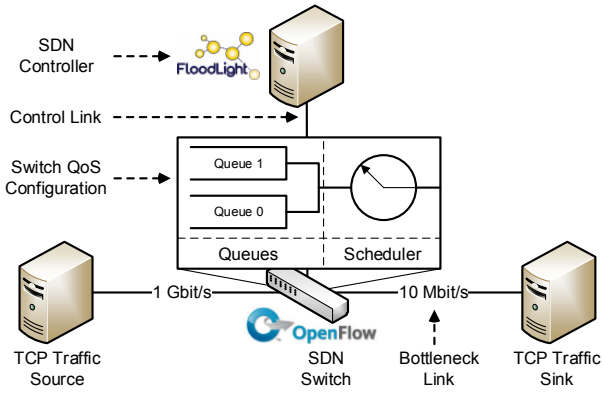

Fig. 1: OpenFlow-based testbed setup. Queue 0 (q_0), Queue 1 (q_1), and Scheduler are configured for each experiment accordingly.

[11] investigates data-rate guarantees equipped at the Pica8 P3290 switch. The influence of short living UDP flows, also called bursts, to a long living UDP flow is measured. The main result is that bandwidth guarantees of the P3290 switch are not fully enforced against such bursts. Therefore, isolation between the data flows can not be guaranteed. While this work focuses on UDP traffic, we investigate the behavior of TCP traffic and use a dynamic resource management instead of static QoS, which does not change over the course of the experiment.

The study [10] examines the OpenFlow metering feature. The metering feature limits the data rate of a flow to a desired data rate. In this study, meters are applied to TCP flows on a bottleneck link. The authors observe that bursts of packets are dropped, which wastes resources and causes TCP to leave the congestion avoidance state and regress to the slow start state frequently. Although the setup is quite similar to our setup, the study is limited to measurements based on the network emulator Mininet and does not evaluate real hardware. Besides only static QoS setups are studied, while we analyze the dynamic behavior. Nevertheless, to our best knowledge, this study is most closely related to our analysis.

III. Measurement Setup and Realization

In this section, we describe the measurement setup and procedure and some background on queuing disciplines. We designed an experiment that examines dynamic TCP QoS support of OpenFlow. We probe and compare different queuing techniques on multiple switches.

A. Experimental Installation

As shown in Figure 1, two physical hosts act as TCP traffic source and TCP traffic sink. The hosts are connected via the

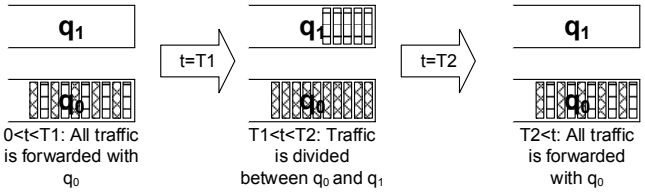

Fig. 2: Time sequence of the experiments

SDN switch. An additional host runs the SDN Floodlight controller, which is connected to the SDN switch. The round trip times (RTT) in the measurement network are $<$1 ms. The application *iperf* [12] is used in version 2.0.5 for the generation of the TCP traffic. The hosts run Ubuntu Linux 14.04 and Linux kernel version 3.13. The data flows are recorded with *tcpdump* [13] and analyzed afterwards. We use TCP CUBIC and disable the features segmentation offloading and TCP metrics save.

As QoS mechanisms are only useful if multiple flows interact with each other, we use two TCP flows that are routed via a single bottleneck link. This link's data rate is limited to 10 MBit/s, while the other link is unrestricted and has a maximum rate of 1 GBit/s. The bottleneck causes that packets are enqueued at the switch. The OpenFlow rules that are necessary for the experiments are set via Floodlight's REST-API [14].

The course of the experiment is presented in Figure 2, It is divided in three stages. At time $t = 0$, both TCP flows are started and forwarded over the same queue (q_0). As TCP provides fairness, they share the bandwidth approximately equally. At time $t = T1$, one flow is rerouted via a different queue (q_1). Depending on the settings of the queues, one flow has more bandwidth than the other in the second stage ($T1 < t < T2$). In the third stage, from time T2 onward, the flows are again forwarded together in q_0. Each of the stages has a duration of 30 s, which means that T1 is 30 s and T2 60 s after the start of the run.

B. Investigated Switches

In our studies, we investigate two hardware switches from different vendors, namely an NEC PF5240 and a PICA8 P3290, and the Open vSwitch (OVS) [6], which can run on commodity hardware.

The NEC switch supports multiple QoS techniques, while only Priority Queuing (PQ) and Weighted Fair Queuing (WFQ) are studied in this paper. WFQ is used to provide Bandwidth Guarantees (BG). Each queue has a length of 64 packets. The P3290 is a so called bare metal switch and supports different network operating systems. In our setup, the P3290 runs PicOS in version 2.1.5 in OVS-mode. In the OVS-mode, Open vSwitch running on top of PicOs provides the OpenFlow interface. Regarding the configuration of the queues, priorities and a predefined minimum and maximum bandwidth can be set. The OVS runs version 2.1.0 on the traffic sink. The advantage of OVS is that we can change the settings of the queues with the Linux application TC [15].

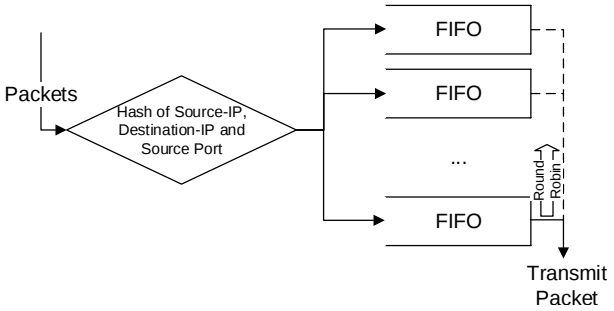

Fig. 3: Stochastic Fairness Queuing Algorithm

C. Background on Queuing Disciplines

In this section, we explain queuing techniques that are used by the studied switches.

1) Classless Queuing Disciplines: Classless queuing disciplines forward traffic neutrally. Neutrally has different meanings for packets and for flows, which lead to different implementations:

FIFO queues have a very simple structure. The item that was first put into the queue leaves it first. In the case of a full queue, incoming packets are dropped. FIFO is fair in the sense of individual packets, as every packet is processed equally.

SFQ [16], in contrast, is a queuing mechanism that tries to ensure fairness between several flows. The basic structure is shown in Figure 3. In order to provide fairness, incoming packets are enqueued to several queues based upon a hash function. Packets are dequeued using the round-robin scheme between the queues. This mechanism results in a high probability that flows are processed fairly, although the fairness is not deterministically guaranteed. Consequently, the mechanism is called Stochastic Fairness Queuing.

2) Classful Queuing Disciplines: In contrast to the previous disciplines, Classful Queuing Disciplines forward packets according to a scheduling algorithm. In case of PQ, queues have different priorities. The packets of the queues are dequeued according to their assigned priority, i.e., a queue is only served if no queue with higher priority has packets. WFQ guarantees a specific, configured minimum bandwidth for each queue. The remaining unused bandwidth is shared between the queues.

OVS uses the Hierarchical Token Bucket (HTB) discipline for BGs. While HTB affects the rate at which packets are sent from each queue, the type of the underlying queue affects the order of the sent packets. This underlying queuing discipline can be, for example, FIFO or SFQ.

IV. Measurement Results

The following figures show the results of 50 runs. Figure 4 shows the bandwidth of the flow over time and Figure 5 shows the sum of the duplicated packets for intervals with a length of 10 s around T1 and T2. We examined Priority Queuing (PQ) and Bandwidth Guarantees (BG).

A. Priority Queuing

Figures 4a)-d) show the results for PQ. In the first stage, both flows are in q_0 and share the bandwidth equally. q_1

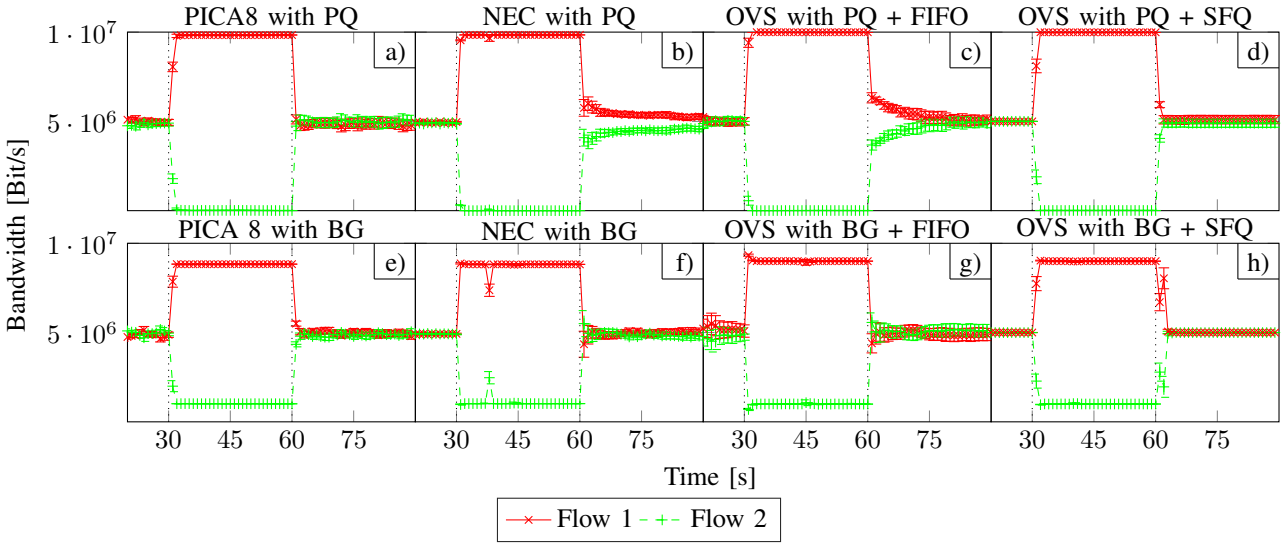

Fig. 4: Measured mean bandwidth over time using different switches and queuing settings. 95% confidence intervals for the mean values are added to all plots.

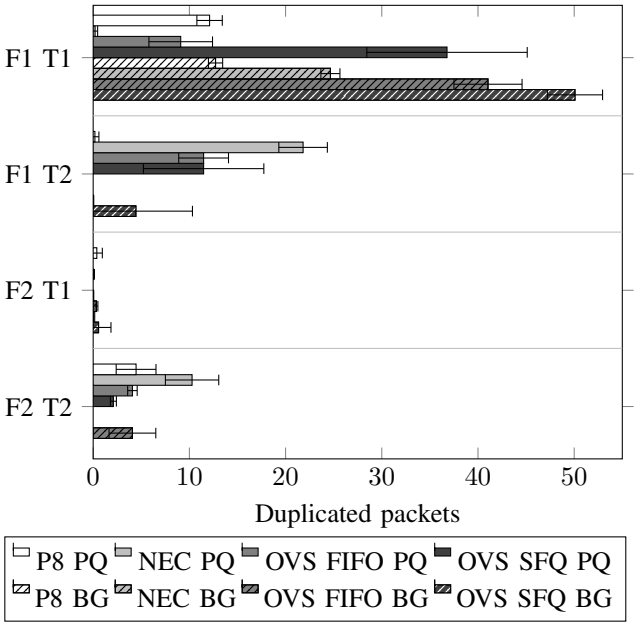

Fig. 5: Mean number of duplicated packets of Flow 1 (F1) and Flow 2 (F2) in an interval of 10 s around T1 and T2. 95% confidence intervals are included. Measurement at the sink of the flows.

is set to have a higher priority than q_0, therefore Flow 2 is depleted very fast after T1. After T2, Flow 1 is again together in q_0 with Flow 2. For the PICA8 switch, we can observe that Flow 2 recovers pretty fast. This can be explained as follows: In the first stage, Flow 1 via q_1 prevents the submission of packets from Flow 2 via q_0. In the meantime, as the source of Flow 2 still tries to reestablish the connection, these packets are enqueued at q_0 but can not be forwarded and are not dropped. After T2 both Flows are forwarded via q_0 and thus these packets of Flow 2 are flushed to the sink. The sink acknowledges these packets, which causes the congestion window and, thus, the bandwidth of Flow 2 to increase quickly.

The number of duplicated packets at the sink confirms the observation. Packets of Flow 1 are duplicated at both switchover points (T1, T2). At time T1 the packets in q_1 pass packets that are enqueued in q_0 and cause the sink to request the slower packets again, which results in duplicated packets. This means that short after T1 packets from both queues are sent, in contrast to the rest of the second stage. At time T2 packets from q_0 are submitted again, at this time some retransmission packets from Flow 2 are enqueued that were sent during the second stage and appear at T2 as duplicates.

The NEC Switch shows a different behavior in the third stage. When Flow 1 is rerouted to q_0 at T2, the average bandwidth of Flow 2 grows fast first, until it again reduces. The time until fairness between the two competing flows is regained is longer than 30 seconds. The number and distribution of duplicated packets are similar to the results with the PICA8 switch, although fewer packets are duplicated at T1. On the other hand, duplicated packets of Flow 1 are measured at T2. A detailed analysis shows that these packets were stuck in the queue since T1. The very steep ascent of Flow 1 at T1 causes this behavior.

For comparison, we have examined OVS with two different classless queuing disciplines, namely SFQ and FIFO, which handle the transmission of packets out of q_0 and q_1. The FIFO queues are configured with a length of 64 packets. The SFQ queues have a length of 128 packets, which is fixed at compile time. Until T2 the differences between FIFO and SFQ are marginal. After T2, fairness is established very fast and accurate, if SFQ is used. This is the expected behavior as SFQ tries to ensure fairness between all flows. However, the

results for FIFO are quite different: The time until the two flows have again approximately the same bandwidth is in the range of 20 to 30 seconds. We can observe a higher number of duplicated packets when OVS is used at both switchover points. Especially SFQ causes a high number of duplicated packets. This can be justified with the bigger size of the queues.

From the results we can see that only OVS with SFQ behaves as one would naively expect. In a network with different switches, the discovered diversities will degrade the performance of QoS-oriented SDN applications, if they are not prepared accordingly. Duplicated packets need to be taken into account when deploying and using QoS mechanisms, as they cause a waste of bandwidth.

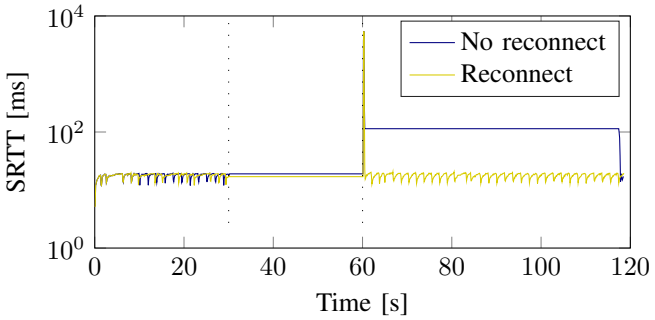

Fig. 6: Smoothed round trip time of Flow 1 from two different measurements using priority queuing on the NEC Switch

1) PQ and TCP's Retransmission Behavior: The results show that Priority Queuing with TCP flows can lead to the depletion of the flow in the lower priority queue. As TCP always interprets packet loss as congestion, the complete depletion causes the source to throttle its sending rate to almost zero. When a TCP flow is depleted, only some retransmissions are sent in an exponentially increasing interval starting with 1 s according to the retransmission timeout (RTO) timer definition in RFC6298 [17]. If we choose $T2 - T1 = 30\,\text{s}$ Flow 2 is blocked for 30 seconds and retransmission are sent 1 s, 3 s, 7 s and 15 s after T1. This results in an RTO of $2^4 = 16\,\text{s}$ at the end of the blocked section. When the transmission restarts at $t = 60\,\text{s}$, RTO is not reset immediately. Instead RTO is computed out of round trip time (RTT) measurements at the source, which are averaged out and result in the so called Smoothed Round Trip Time (SRTT). SRTT of Flow 2 is plotted for two different runs in Figure 6. Until T1, SRTT moves around about 20. Between T1 and T2, no packets are transmitted, therefore RTT measurement is not possible and SRTT is constant. At time T2, the packets that were stuck in q_0 are flushed to the sink, which acknowledges these packets. This leads to some very high RTT measurements at the source and, thus, a very large SRTT. When no further packets are lost, SRTT gets small again pretty fast as more and more RTT measurements after T2 are successful. On the other hand, if the reconnect after T2 fails, due to packet loss, this results in a subsequent connection interruption. The TCP stack of the source waits RTO until it sends another reconnect. As SRTT is still high, RTO is also big and it can take many seconds until the TCP flow restarts again. In the pictured measurement it took 60 seconds until the connection was finally reestablished.

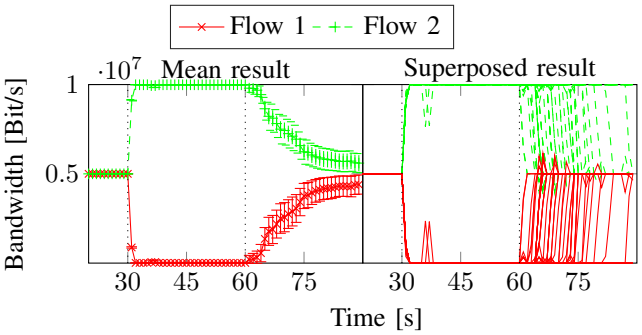

Fig. 7: Bandwidth of 50 runs using OVS employing inverted PQ and SFQ

Altogether this can lead to an interruption of the TCP connection. In the worst case, the application running the TCP connection does not restart the connection on its own and a user intervention is necessary. This means that in case of PQ starvation of flows should be avoided or at least kept short. An active notification to the source host from the switch at T2 could avoid the problem.

2) Inverted Priority Queuing: This behavior has a major influence on the result with "inverted" priorities. "Inverted" priorities means that first both flows are routed over the queue with the higher priority, then one flow is routed over the lower priority queue until both flows are again together on the higher priority queue. We performed experiments with inverted PQ for all switches, as the results are all comparable with some minor differences, we only show the results for OVS with SFQ in Figure 7. The mean result gives the impression that the time until fairness is reestablished is very long. But in this case the superposed single results give a better insight. These plots show that if Flow 1 is reconnected, fairness is restored pretty fast. But the time when Flow 1 is reconnected is very divergent. Between T1 and T2 no packets can be transmitted from Flow 1, the source tries to reestablish the connection with retransmissions. In contrast to the previous experiments, these packets are still retained also after T2. Therefore the only way that Flow 1 reconnects is a retransmission packet after T2, although, as Flow 1 was disconnected fairly long, the frequency of retransmissions is very low. The link is still well loaded by Flow 2, therefore the probability is high, that one or more of these packets are dropped, which can stop the TCP connection again for many seconds. As this is a non-deterministic process each experiment is different. Summarizing this it must be noted that the queuing approach regarding PQ makes a major difference.

B. Queuing Disciplines with Guaranteed Bandwidth

In the following, we investigate queues with performance guarantees. The basic setup stays the same, although now, the bandwidth in the second stage ($T1 < t < T2$) is to be shared

according to a 1:9 ratio. The sum of the guaranteed rate equals the link speed. This results in 1 Mbit/s guarantee for q_0 and 9 Mbit/s for q_1. The NEC switch supports WFQ, while the other switches can employ bandwidth guarantees for their queues.

Using the PICA8 switch, after T1 the bandwidth of Flow 2 decreases very fast to the configured 1 Mbit/s. After T2, fairness is reestablished quickly. The number and distribution of duplicated packets is comparable to the PQ experiment.

In the second stage the bandwidth guarantees are violated for a short instance using the NEC product. The drop is visible for approximately 6% of the measurements always 8 seconds after T1. This regularity indicates some switch fault. As no packets of Flow 2 are accumulated during stage 2, fairness is established more steadily compared to the PQ case. Regarding the duplicated packets, one can see the same behavior as for the PICA8 switch, although more packets are duplicated in total.

The bandwidth results for the experiments with OVS and SFQ are close to ideal: at T1 the bandwidth of Flow 1 increases in less than two seconds to the predefined level and at T2 fairness is reestablished very fast. OVS and FIFO shows a different run of the bandwidth. Especially interesting is the behavior at T2: the bandwidth of Flow 2 overshoots Flow 1 and afterwards both flows oscillate around the fair distribution. The number of duplicated packets is much higher, when compared to the hardware switches.

Summarizing the results, we can see that the guarantees, as they do not cause the depletion of one flow, do not harm the overall connectivity of the network. Vendor dependencies for this setting are less distinctive compared to priority queuing, although with bandwidth guarantees none of the flows can use the full bandwidth of the link, which might be desired for some applications.

V. Conclusion

Software-Defined Networking (SDN) enables an easy and flexible realization of existing dynamic Quality of Service (QoS) mechanisms in today's communication networks. Although SDN and, in particular, OpenFlow claims to provide a standardized interface, the existing diversity of OpenFlow-enabled switches leads to varying behavior for the same QoS mechanisms. Existing work on realizing QoS mechanisms and on OpenFlow switch diversity has neglected the impact of dynamic QoS mechanisms on network traffic.

Accordingly, in this paper, we initially study switch diversity while deploying dynamic QoS mechanisms for TCP flows in an OpenFlow-based network. Our measurement results for two fundamentally different QoS techniques, namely priority queuing and bandwidth guaranteeing, show severe performance variations for two hardware and one software switch. Using priority queuing, the different switches lead to different, even unfair, bandwidth shares between TCP flows. Furthermore, the use of priority queuing can interrupt TCP connections, which can result in indeterministic flow behavior. Besides, our results for the software switch show that different queue implementations, i.e., FIFO queues or SFQ queues, also impact the network performance. In case of bandwidth guarantees, one hardware switch violates the configured bandwidth guarantees. For all setups, deploying dynamic QoS mechanisms leads to duplicated TCP packets, i.e., a waste of network resources.

Acknowledgment

This work was partially funded by the Federal Ministry of Education and Research Germany (BMBF) under grant number 16KIS0260. The authors alone are responsible for the content of the paper.

References

[1] P. Georgopoulos, Y. Elkhatib, M. Broadbent *et al.*, "Towards network-wide QoE fairness using OpenFlow-assisted adaptive video streaming," in *Proc. of the 2013 ACM SIGCOMM Workshop on Future Human-Centric Multimedia Networking (FhMN 2013)*, Hong Kong, China, 2013, pp. 15–20.

[2] T. Zinner, M. Jarschel, A. Blenk *et al.*, "Dynamic application-aware resource management using software-defined networking: implementation prospects and challenges," in *Proc. of the 2014 IEEE Network Operations and Management Symposium (NOMS '14)*, Krakow, Poland, 2014, pp. 1–6.

[3] W. Kim, P. Sharma, J. Lee *et al.*, "Automated and scalable QoS control for network convergence," in *Proc. of the 2010 Internet Network Management Conference on Research on Enterprise Networking (INM/WREN '10)*, San Jose, Canada, 2010.

[4] A. Lazaris, D. Tahara, X. Huang *et al.*, "Tango: simplifying SDN control with automatic switch property inference, abstraction, and optimization," in *Proc. of the 10th ACM International on Conference on emerging Networking Experiments and Technologies (CoNEXT)*, Sydney, Australia, 2014, pp. 199–212.

[5] M. Kuzniar, P. Peresini, and D. Kostic, "What you need to know about SDN control and data planes," EPFL, Lausanne, Switzerland, Tech. Rep. EPFL-REPORT-199497, 2014.

[6] "Open vSwitch - an open virtual switch," 2014. [Online]. Available: http://openvswitch.org/

[7] V. Mann, A. Vishnoi, A. Iyer *et al.*, "VMPatrol: dynamic and automated QoS for virtual machine migrations," in *Proc. of the 8th International Conference on Network and Service Management (CNSM)*, Las Vegas, USA, 2012, pp. 174–178.

[8] Z. Bozakov and A. Rizk, "Taming SDN controllers in heterogeneous hardware environments," in *Proc. of Second European Workshop on Software Defined Networks (EWSDN)*, Berlin, Germany, 2013, pp. 50 – 55.

[9] M. Kuzniar, P. Peresini, and D. Kostic, "What you need to know about sdn flow tables," in *Passive and Active Measurement*, ser. Lecture Notes in Computer Science, J. Mirkovic and Y. Liu, Eds. Springer International Publishing, 2015, vol. 8995, pp. 347–359.

[10] P. M. Mohan, D. M. Divakaran, and M. Gurusamy, "Performance study of TCP flows with QoS-supported OpenFlow in data center networks," in *Proc. of the 19th IEEE International Conference on Networks (ICON)*, Singapore, Singapore, 2013, pp. 1–6.

[11] A. Nguyen-Ngoc, S. Lange, S. Gebert *et al.*, "Investigating isolation between virtual networks in case of congestion for a Pronto 3290 switch," in *Proc. of the Workshop on Software-Defined Networking and Network Function Virtualization for Flexible Network Management (SDNFlex 2015)*, Cottbus, Germany, 2015.

[12] A. Tirumala, F. Qin, J. Dugan *et al.*, "Iperf: the tcp/udp bandwidth measurement tool," 2008. [Online]. Available: http://sourceforge.net/projects/iperf/

[13] V. Jacobsen, C. Leres, and S. McCanne, "Tcpdump/libpcap," 2012. [Online]. Available: http://www.tcpdump.org

[14] "Floodlight - open SDN controller." [Online]. Available: http://www.projectfloodlight.org/floodlight/

[15] "tc - show / manipulate traffic control settings." [Online]. Available: http://www.lartc.org/manpages/tc.txt

[16] P. McKenney, "Stochastic fairness queueing," in *Proc. of the Ninth Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM '90)*, San Francisco, USA, 1990, pp. 733–740.

[17] V. Paxson, M. Allman, J. Chu *et al.*, "Computing TCPs retransmission timer," 2011. [Online]. Available: http://www.rfc-editor.org/info/rfc6298