Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking

[1]Airton Ishimori, [1]Fernando Farias, [1,2]Eduardo Cerqueira, [1]Antônio Abelém
[1]GERCOM Lab - Federal University of Pará (UFPA) - Brazil
[2]NRL - University of California, Los Angeles (UCLA) - USA
{airton, fernnf, cerqueira, abelem}@ufpa.br

***Abstract*—Packet scheduling is essential to properly support applications on Software-Defined Networking (SDN) model. However, on OpenFlow/SDN, QoS is only performed with bandwidth guarantees and by a well-known FIFO scheduling. Facing this limitation, this paper presents the QoSFlow proposal, which controls multiple packet schedulers of Linux kernel and improve the flexibility of QoS control. The paper assesses QoSFlow performance, by analysing response time of packet scheduler operations running on datapath level, maximum bandwidth capacity, hardware resource utilization rate, bandwidth isolation and QoE. Our outcomes show an increase more than 48% on PSNR value of QoE by using SFQ scheduling.**

***Index Terms*—QoSFlow, Packet Schedulers, QoS, OpenFlow**

I. Introduction

Computer networks have increased in an unprecedented scale, facing challenges to properly support applications such as multimedia services and mobile devices, besides efficient management and control system for networking have become more valuable inside of enterprise and academic backbones over recent decades. Many challenges, such as firewall and Quality of Service (QoS) management have become crucial on networks.

The researches for network innovation have opened a race for betterment on data and control plane to enable higher management flexibility. Among them, we highlight Software-Defined Networks (SDN) paradigm as the main model for next generation networks that has rapidly emerged with OpenFlow (OF) protocol.

The OF proposal, which follows the SDN principles for controlling network devices, allows a straightforward QoS support [6]. Both the so used OF 1.0 and the latest version 1.3, provide some QoS mechanism. OF 1.3 has been released with a rate limiting feature, which suplies OF 1.0 automation issues. On the one hand, this means that network owners are unaware of `dpctl` (OF's command-line utility). On the other hand, they are able to configure QoS of the entire network from an OF controller's application.

However, whichever is the version of the protocol, packets cross the network with the well-known First-In First-Out (FIFO) treatments. Because of that, OF switches might not meet the requirements for supporting QoS of some applications like multimedia streaming. Also, the Quality of Experience (QoE), which is the QoS assessed by users might be damaged due to packet handling of the network infrastructure. It is important to highlight that not only is the QoS related to traffic shapping capability, but also about the sending packet order, which is when some packets in a queue position can have higher priorities than the packets that are ahead of them. Packet schedulers besides FIFO may be found into the Linux's traffic control subsystem.

As we know, an OF software switch runs on top of Linux operating system. This means that vendor's switches based on Linux such as NetGear, Centec or Pantou, which is a wireless openflow switch, and Linux PCs are capable of obtaining advantages from a rich Linux traffic control. However, the OF specifications do not support to control packet schedulers. Hence, we have been facing potential challenges on OF networks.

This paper introduces QoSFlow, a QoS development strategy for OF enabled networks to overcome packet scheduling issues. The main goal of QoSFlow is to allow control of multiple packet schedulers. In another words, QoSFlow brings the traffic control of Linux to become part of OF networks. Our proposal extends the OF protocol 1.0 and the standard datapath based on it. This way, developers can deploy their own application to enable, for instance, a control of bandwidth on-demand with one or more packet schedulers on the network. Currently, QoSFlow provides control of the following packet schedulers: HTB (Hierarchical Token Bucket) [7], RED (Randonly Early Detection) [5], and SFQ (Stochastic Fairness Queuing) [9].

The remainder of the paper is structured as follows. In Section II the related work on QoS in OF networking is presented. In Section III the design and implementation of QoSFlow. In Section IV our experiments with QoSFlow, confirming its good performance to configure packet schedulers. We conclude this paper in Section V.

II. Related Work

Kim et al. [8] propose a network QoS control framework, allowing to program QoS parameters on network devices from high-level slice specification. However, the authors do not specify the extensions implemented to automate QoS control on OF networks.

Cinvalar et al. [3] have described an optimization model to improve packet routing. Such model considers delay and packet loss. The optimization model, through the linear programming, computes a QoS path for video traffics and a shortest path to best effort traffics.

978-1-4799-2433-2/13 $31.00 © 2013 IEEE
DOI 10.1109/EWSDN.2013.20

OpenQoS [4] is an evolution of Cinvalar et al. work [3]. The proposal adds a service layer over an OF controller, which network owners are able to configure flow definitions by using a new prioritization strategy based on routing. This proposal performs per-flow routing with or without QoS criterias.

Sonkoly et al. [11] propose a QoS formulation to Ofelia Control Framework (OCF) and an overall need to use fine-grained QoS control on testbeds environments. The adaptation includes an extension in OCF Expedient, Opt-In Manager, FlowVisor, and OF datapath. The main goal is to achieve resource guarantees for experimenters.

Besides that, Cinvalar et al. [3] and Egilmez et al. [4] do not offer support for traffic shaping, but Kim et al. [8] use it as the main features to provide QoS. Finally, Sonkoly et al. [11] have no proposal of evaluation.

According to the related works above, the overall focus is to achieve network resource isolation on OF domain. Indeed, an isolation is extremely necessary for QoS improvements, but this paper proposes the control of multiple packet scheduling mechanisms on the network core to improve QoS. It is possible to define different scheduling alogrithms to each services and traffic shaping feature is achieved as well.

III. Design of QoSFlow

QoSFlow module adds extensions to the standard software switch (datapath) of OF 1.0. The reason why the specification 1.0 has been used is because when the QoSFlow project started, such specification was the latest stable version. Even though OF 1.3 has brought a new mechanism for rate limiting, but as well as the OF 1.0, we are just able to use FIFO instead of other packet schedulers to achive different treatments to the packets.

The OF datapath plus QoS modules form the QoSFlow datapath. This datapath is a user space implementation where queues are located in the kernel space. The QoS module opens a channel with the kernel through Netlink and Packet socket families to connect both user and kernel space. Thus, the packet schedulers can be instantiated to enable traffic shaping and enqueueing of flows. The components called Traffic Shaping, Packet Schedulers and Enqueueing that compose the QoS module of the QoSFlow datapath, and their relationships are illustrated in Fig. 1.

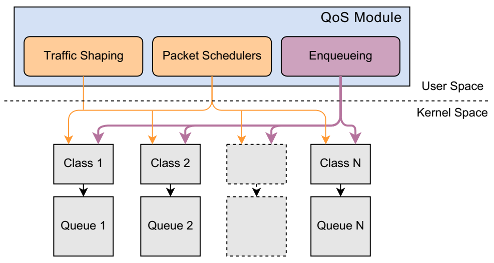

Fig. 1. QoS module which has been added to the standard OpenFlow datapath

- **Traffic Shaping and Packet Schedulers.** These components use Netlink socket family to manipulate `OFPT_QOS_QUEUEING_DISCIPLINE` message type, which is a new extension of the message to represent the QoS messages in OF protocol.

  Hence, the Traffic Shaping and Packet Schedulers components administer the QoS messages receipt from control plane by splitting the bandwidth size in queues and by attaching or detaching packet schedulers for these queues, respectively.

  To establish a connection with the kernel, these components open a Netlink socket channel and send a Netlink message through it. The Netlink message is the type of message that Linux kernel accepts for network resources management. In this way, the QoS messages are mapped to Netlink messages.
- **Enqueueing.** It is the component responsible to operate `OFPT_FLOW_MOD` messages of the OF protocol. This message modifies the state of flow table, where each entry contains header fields, counters and actions for matching packets or flow packets.

  The enqueueing mechanism maps flows to queues using the `skb->priority` of kernel's data structure called `sk_buff`. This configuration is done through the use of the `SO_PRIORITY` option of the Packet socket family. Since, user space cannot access such data structure directly. This is the limitation of the standard OF datapath, whose QoSFlow has also based on.

Currently, the QoSFlow implementation supports a maximum limit of 8 queues per switch ports. This is due to the Slicing mechanism [1]. However, unlike of Slicing mechanism, QoSFlow is able to handle multiple packet schedulers and does not call `tc` (Linux's tool) at datapath. Moreover, the number of queues is chosen and instantiated from control plane. So, this turns the datapath more flexible.

A. QoS Features

The QoSFlow project brings the rich traffic control subsystem of Linux kernel to OF environment and, at the same time, it enables more flexibility to the OF datapath. Below, there is a description of the QoS features in QoSFlow:

- **Shaping.** Delay packets to meet a desired rate to each queue. It enables to configure a minimum and maximum bandwidth capacity. This is performed by HTB packet scheduler.
- **Round-robin treatment.** Enqueued traffic is sent in turn. Each packet in a queue is sent in order to give every flow an equal chance to use the transmission channel. This performed by SFQ packet scheduler.
- **Congestion avoidance.** Smart enqueueing to avoid queue saturation. It simulates a physical congestion and mark enqueued packets to be dropped. This is performed by RED packet scheduler.

The QoS features of QoSFlow use the packet schedulers of Linux kernel, also known as queueing disciplines. There are two kinds of queueing disciplines supported, which brings to the OF environment a wildly and powerful system for QoS provisioning. They are known as classless and classfull queueing disciplines.

The classless queueing discipline has no configurable internal subdivisions in classes. Such type creates an order to transmit packets through an egress port. This mechanism selects the next packet that must be dequeued, for example, in FIFO scheduling approach the first packet that arrived must be sent firstly, thereafter the second one and so on.

On the other hand, a classfull queueing discipline contains several classes where there are two types of classes called root and leaf class. The former is the main class from subdivisions in classes can be done, while the latter is a terminal class, which has a queue associated to it where SFQ or RED scheduling (classless) can be attached.

The class-based queueing discipline needs to determine in which class a flow packet must be sent to be forwarded into queues. This is done by using the the flow table of the switches in a QoSFlow environment that separate traffics into queues.

Currently, QoSFlow controls the following packet schedulers: HTB, SFQ and RED where the HTB is a classfull, while SFQ and RED are classless queueing discipline. Thus, the current QoSFlow features come from these Linux kernel packet schedulers.

- **HTB.** Allows to split bandwidth size of the network. By default, the Linux kernel automatically attaches a FIFO packet scheduler to each bandwidth segment. It creates logical links wich is slower than physical link.
- **SFQ.** Belongs to fair queueing algorithms. The SFQ schedules the packets transmission based on information about IPv4/v6 source and destination address, and TCP/UDP source port to assign each flow to each hash bucket, on the enqueueing phase.
- **RED.** It drops packets in a queue gradually. It performs a tail drop like FIFO, but smartly. Such packet scheduler has a threshold value to mark packets to be discarded, after queue length becomes greater than threshold value.

We emphasize that FIFO can also be configured as well. Actually, this always happens automatically when a bandwidth slice is created, because Linux kernel use it as a default packet scheduling algorithm. However, QoSFlow is able to resize queue length by using Packet FIFO (PFIFO) and Byte FIFO (BFIFO) as FIFO scheduler. The difference between both of them is the manner for configuring queue length. In the first one, the queue length is set based on the number of packets, and in secondly, based on the number of bytes.

B. Implementation

1) Algorithm: The Algorithm 1 shows QoS message handling process, which represents the Traffic Shaping and Packet Schedulers components presented in Section III.

The algorithm is part of the QoS module. It requires pieces of information related to datapath, controller and QoS message with `OFPT_QOS_QUEUEING_DISCIPLINE` message type.

Algorithm 1 QoS Message Handling

```
Require: $Dpid, Sender, QosMsg$
Ensure: $Success$ or $Fault$

  $Body \leftarrow$ GetBody($QosMsg$)
  $Type \leftarrow$ CheckType($QosMsg$)

  switch $Type$
  case OFP_QOS_SCHED_HTB
      ProcessQosHtbMsg($Dpid$, $Body$)
  case OFP_QOS_SCHED_SFQ
      ProcessQosSfqMsg($Dpid$, $Body$)
  case OFP_QOS_SCHED_RED
      ProcessQosRedMsg($Dpid$, $Body$)
  case OFP_QOS_SCHED_PFIFO
      ProcessQosPfifoMsg($Dpid$, $Body$)
  case OFP_QOS_SCHED_BFIFO
      ProcessQosBfifoMsg($Dpid$, $Body$)
  other case
      SendErrorToController($Dpid$, $Sender$)
      return $Fault$
  end switch

  return $Success$
```

This message has a subtype as showed in each `case` statement of the algorithm.

The message processing is done according to each subtype. The HTB scheduling has as goal to split the bandwidth into smaller segments. By default, all the switch ports come with HTB attached to each port, but with any leaf classes (or queues). Thereby, control plane only needs to send HTB messages to create queues with a decided bit rate capacity. Other scheduling mechanisms are classless that is used to associate into queues created previously by HTB.

However, problems might occur in the message manipulation. For that, the datapath sends an error message to notify control plane, through the `OFPT_ERROR_MSG` message type of the OF protocol.

The error message of OF data structure has a field called `type` and `code`. The `type` value indicates an error. The `code` value is interpreted based on the type. The new error type named `OFPET_SCHED_FAILED` and code named `OFPSFC_BAD_SCHED` was added to OF protocol. These are the only ways to notify packet scheduler operation issues to control plane in current QoS extension of the QoSFlow proposal.

2) QoS Message: The data structure of QoS message has a field named `header`, `sched_type` and `body`. The `header` is a field of OF header, the `sched_type` defines a packet scheduler type and the `body` holds the packet scheduler parameters. The `body` content and total length of the QoS message varies with `sched_type`.

The Algorithm 1 manipulates the QoS message received from control plane in according to each `sched_ type`, where there is an appropriate function to handle it. Such function maps the QoS message to a Netlink message. And, this message is sent to Linux kernel traffic control subsystem.

The information to be mapped from `body` to Netlink message depends on `sched_type`. The QoSFlow proposal parses it to each `sched_type`:

- `OFP_QOS_SCHED_HTB`. The parsing process uses the parameters `rate` and `ceil` to create a bandwidth segment. They hold a value to limit a minimum and maximum bit rate, respectively.
- `OFP_QOS_SCHED_SFQ`. The parsing method uses the parameters `perturb` and `quantum`. Each hash bucket with data is queried in a round-robin manner in dequeuing phase. So, both the parameters hold perturbation time to hash bucket and a number of bytes to be dequeued, respectively.
- `OFP_QOS_SCHED_RED`. The parsing process uses the parameters `threshold` and `limit`. The packets are marked to be dropped starting from a certain queue lenght. Both parameters are used to set such starting point and total queue lenght, respectively.
- `OFP_QOS_SCHED_PFIFO`. The parameter `limit` has a queue length that defines the maximum number of the packet to be enqueued.
- `OFP_QOS_SCHED_BFIFO`. The parameter `limit` has a queue length that defines the maximum number of bytes can be enqueued.

Therefore, the QoS message can hold HTB, SFQ or RED packet schedulers parameters as showed above and processed according to Algorithm 1, but the `header` field is checked before. Such message processing is only performed to the QoS message. So, the `OFPT_QOS_QUEUEING_DISCIPLINE` message type must be always stamped in the `header` field for QoS handling.

3) Netlink Message: Netlink [10] is one of the IPC (Inter-Process Communication) method that Linux provides to the user space. Netlink is a socket interface family enhanced from BSD operating system. It gives datagram-oriented messaging facility to send and retrieve kernel space information from the user space. However, unlike other socket family (e.g. INET socket family) it cannot traverse host boundaries, because, on the Netlink communication, the processes are identified locally by their Process Identification (PID).

In the QoSFlow proposal, the Netlink message is used to request kernel to add, delete or update the packet scheduler configuration. The traffic control subsystem manipulates such message after receipt it from the user space. Thus, the kernel instantiates a packet scheduler configuration.

The kernel looks for `flags` bit field of the Netlink message. If `NLM_F_REQUEST` is set, this means that the message contains a request from user space. Such request can add a new configuration related to packet scheduler or classes. The `flags` is always checked by the kernel.

In a case of `NLM_F_REQUEST` and `NLM_F_CREATE` are activated together in `flags`, this means that new configuration must be done in the kernel. But, this configuration depends on the `type` field of the Netlink message. Such field can hold `RTM_ADDTCLASS` or `RTM_ADDQDISC` value. The QoSFlow proposal uses the former to install new HTB leaf class (queue), and the latter, to attach the RED or SFQ packet scheduler on a queue.

If only `NLM_F_REQUEST` is set in `flags` and `type` holds `RTM_ADDTCLASS` or `RTM_ADDQDISC` value, it means an update on a leaf class or a packet scheduler must be done, respectively. However, if the `type` field holds `RTM_DELTCLASS` or `RTM_DELQDISC` value than a leaf class or a packet scheduler must be deleted from the Linux kernel, respectively.

IV. Performance Evaluation

The QoSFlow datapath performance evaluation is over a commercial switch called TP-Link 1043ND, which its processor uses instruction set of MIPS (Microprocessor without Interlocked Pipeline Stages) architecture and a non-OF vendor's firmware. However, the original firmware must be replaced to a Linux operating system to allow such commercial switch model to support OF. For this propose, OpenWrt BackFire has been installed according to Pantou project's [2] description. Thus, the QoSFlow datapath executes on top of OpenWrt.

We evaluated the response time of QoS low-level operations, maximum throughput and resource usage of the switch, where the datapath resides. On evaluations, a confidence level of 95% are applied in averages. The outcomes in Fig. 2 and 4, a network topology with a single switch is used, but for results showed in Fig. 3 a linear topology with one, two and three switches are used. For the outcomes in Fig. 5 a linear topology with three switches are used. Lastly, for results showed in the Fig. 6 a linear topology with three switches are used.

A. Response Time

The total of 40 repetitions is done to calculate the response time for QoS operations under device. These operations are performed to each function after `case` statement as presented by Algorithm 1 in Section III. To obtain response time values, a timer is inserted to get time value before (`t1`) and after (`t2`) calling a function of QoS operation. Thereafter, the difference time between `t2` and `t1` is calculated. Thus, such difference time is the spent time (or setup time) when a QoS operation is done. This time involves the QoS message parsing time, the transmission time to kernel space and configuration time into the kernel.

According to Fig. 2, aproximately, HTB spent 10.24 ms, PFIFO and BFIFO spent 0.54 ms, SFQ spent 0.80 ms and RED spent 4.40 ms, respectively, for add, remove and update operations. The distinct values in response time are because of the number of bytes transmitted from user space to kernel space after QoS message parsing and the complexity of each packet scheduler implementation.

The HTB response time refers to HTB leaf class (queue) configuration time. The response time for SFQ, RED, PFIFO and BFIFO refers to the time spent to configure the packet scheduler on an existing queue. These times do not consider the queue creation time spent by HTB.

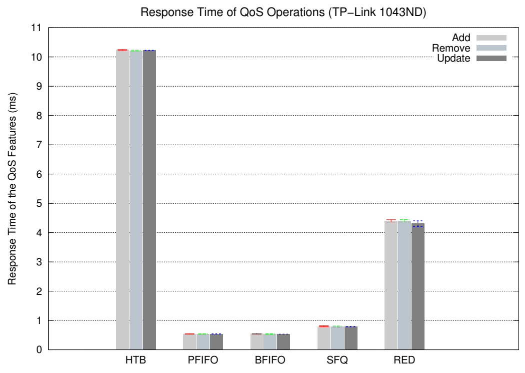

Fig. 2. Response time to execute operations over Linux packet schedulers

All of the response times do not take into consideration the network latency. The tests only evaluate the switches' internal operations costs, where they have smooth samples variation. This can be induced by checking the short confidence intervals.

B. Switch Capacity

The following tests are related to switch capacity as Fig. 3 shows. Such tests evaluate the standard OF switch and QoSFlow switch maximum throughput in linear topologies. Two physical end hosts are used; one to produce TCP traffic and other to receive it over 120 seconds. This traffic is sent by using `Iperf`[1] traffic generator tool.

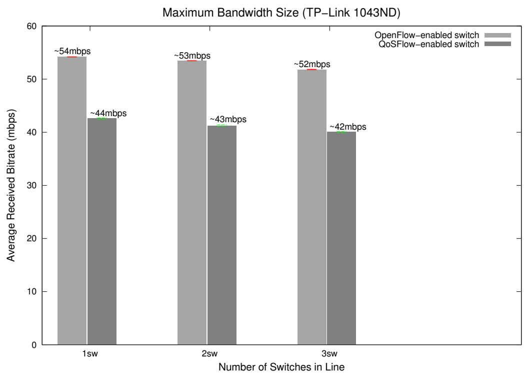

Fig. 3. OpenFlow switch and QoSFlow switch maximum throughput in linear topology

The standard OF switch reaches about 54 Mb/s, 53 Mb/s and 52 Mb/s and the QoSFlow switch reaches about 44 Mb/s, 43 Mb/s and 42 Mb/s, as the maximum network throughput through an Ethernet port, where one queue with 100 Mb/s capacity is configured to QoSFlow switch. In both cases, a linear decreasing appear with the number of switch grows in a linear topology. However, the QoSFlow switch has a lower capacity then standard OF switch. The difference is about 10 Mb/s. Because of QoSFlow uses virtual queues to keep packets enqueued into them while OF has no virtual queue and sends packet flows directly to the physical interface card.

C. Number of Queues Impact

As next test, the number of queues impact is evaluated. Again, the TCP traffic during 120 seconds and two physical end hosts is used. Furthermore, a Linux tool named `top`[2] has also been used to evaluate memory and CPU usage. The Fig. 4 shows these outcomes.

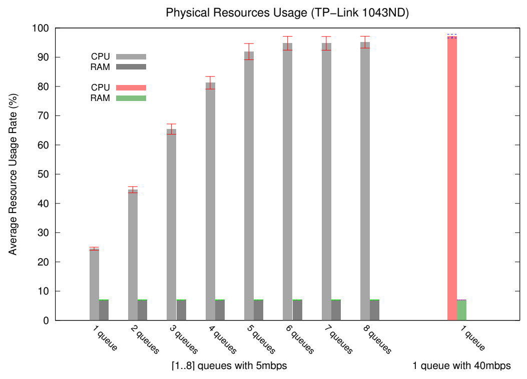

Fig. 4. Primary memory and CPU overhead

For such test, a single topology with QoSFlow switch is considered. The CPU usage rate when there is a single queue with a capacity of 5 Mb/s and a single queue with a capacity of 40 Mb/s are different. In a first case, the CPU usage is aproximately 24.5%, and in a second one, it is about 97.2%. Similarly, the overhead about 97.2% is obtained when there are 8 queues, which each one has the maximum capacity of $5Mb/s$. Thus, the CPU usage cost is not related to a number of queues, but there is a relationship between the maximum capacity of each queue and its utilization. However, the memory usage has no variation. Its overhead is constant at 7%.

D. Bandwidth Isolation

The outcomes in Fig. 5 shows an evaluation of bandwidth isolation for TCP and UDP flows during 120 seconds. This has as an objective to introduce the ability of QoSFlow to separate flow traffics. For UDP, a limit of 6 Mb/s is established while for TCP, is configured to 10 Mb/s. As we can see, both TCP and UDP respects each other, that is, the traffics do not interfere each other.

This bandwidth isolation feature can be used to control bandwidth on-demand as well as the packet scheduling algorithms. As Fig. 2 shows, the configuration of packet schedulers has good performance that is essential for on-demand services with delay restrictions. Moreover, a guaranteed-resource network slice to each experimenters is also possible.

[1] http://sourceforge.net/projects/iperf/

[2] http://linux.die.net/man/1/top

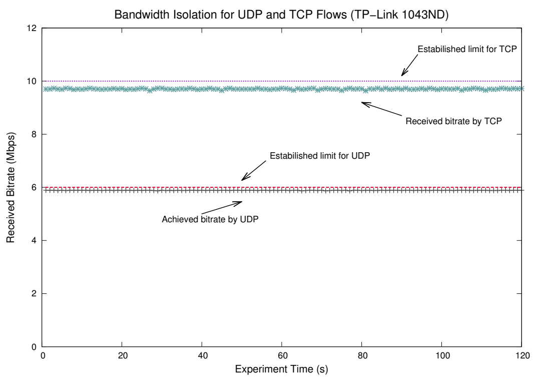

Fig. 5. Bandwidth isolation

E. QoE Evaluation

The last test, is a use case of QoSFlow. The objective metric of QoE named Peak Signal-to-Noise Ratio (PNSR) of two video flows is evaluated. Such metric is evaluated from tools of Evalvid[3] framework as well as the two video flows that were generated in the tests. The Fig. 6 shows the results of the last test. In such test, one queue with capacity of $10\ Mb/s$ is used.

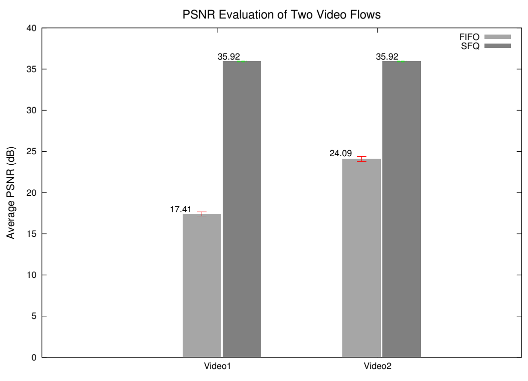

Fig. 6. PSNR value of two video flows

There are resource reservation in both cases, but the PSNR value is better using SFQ than FIFO scheduling. The difference between them is about 48.57% and 68.57% for video 1 and 2, respectively. This is because SFQ assigns each flow to a hash bucket on the enqueueing phase and it uses round-robin treatment to send packet flows.

V. Conclusion And Future Work

QoSFlow is a proposal to bring Linux packet scheduler control in an OF networks by extending standard datapath and the protocol. It offers QoS message to abstract queue configuration complexity. The evaluations showed that it has low response time and the use of SFQ brings improvements on QoE. So, it is essential to bring others packet schedulers to OF networks.

In addition, we believe that QoSFlow might be applied on a backbone of experimentation due to the isolation feature. Also, the ability to control multiple types of packet scheduling, per-flow forwarding with different scheduling mechanisms might be used on OF networks for QoS and QoE improvements.

As a work ahead, a QoS policy-based management framework is going to be developed to program a network with necessary QoS parameters. Such approach gives more flexibility and automation by using a QoS policy definition language, and we have been considering adding support to other packet schedulers as well.

Moreover, we have already started the implementation of QoSFlow into OF 1.3 and testbeds on a network topology formed by Centec, NetGear or other vendor's switch types, whose processing capacity is higher, will be done. Such equipment should be acquired as soon.

VI. Aknowledgments

We would like to thank FIBRE Project, FAPESPA, CNPq, and UFPA - Brazil.

References

[1] http://www.openflow.org/wk/index.php/Slicing, 2011.
[2] http://www.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT, 2011.
[3] S. Civanlar, M. Parlakisik, A.M. Tekalp, B. Gorkemli, B. Kaytaz, and E. Onem. A qos-enabled openflow environment for scalable video streaming. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. IEEE, 2010.
[4] Hilmi E Egilmez, S Tahsin Dane, K Tolga Bagci, and A Murat Tekalp. Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pages 1–8. IEEE, 2012.
[5] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, 1993.
[6] Open Networking Foudation. Openflow specification. 2012.
[7] Bert Hubert, Thomas Graf, Gregory Maxwell, Remco Van Mook, Martijn Van Oosterhout, Paul B. Schroeder, Jasper Spaans, and Pedro Larroy. *Linux Advanced Routing & Traffic Control HOWTO*. Linux Advanced Routing & Traffic Control, http://lartc.org/, April 2004.
[8] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.J. Lee, and P. Yalagandula. Automated and scalable qos control for network convergence. *Proc. INM/WREN*, 2010.
[9] Paul E McKenney. Stochastic fairness queueing. In *INFOCOM'90. Ninth Annual Joint Conference of the IEEE Computer and Communication Societies.'The Multiple Facets of Integration'. Proceedings., IEEE*, pages 733–740. IEEE, 1990.
[10] Pablo Neira-Ayuso, Rafael M Gasca, and Laurent Lefevre. Communicating between the kernel and user-space in linux using netlink sockets. *Software: Practice and Experience*, 40(9):797–810, 2010.
[11] B. Sonkoly, A. Gulyas, F. Nemeth, J. Czentye, K. Kurucz, B. Novak, and G. Vaszkun. On qos support to ofelia and openflow. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 109 –113, oct. 2012.

[3] http://www2.tkn.tu-berlin.de/research/evalvid/EvalVid/docevalvid.html