SDN control framework for QoS provisioning

Slavica Tomovic, Neeli Prasad, *Senior Member, IEEE*, Igor Radusinovic, *Member, IEEE*

***Abstract* — This paper presents new Software Defined Networking (SDN) control framework for Quality of Service (QoS) provisioning. The proposed SDN controller automatically and flexibly programs network devices to provide required QoS level for multimedia applications. Centralized control monitors state of the network resources and performs smart traffic management according to collected information. Beside QoS provisioning for priority flows, the proposed solution aims at minimizing degradation of best-effort traffic. The experimental results show significant performance improvement under high traffic load compared to traditional best-effort and IntServ service models.**

***Keywords* - OpenFlow, SDN, QoS provisioning.**

I. Introduction

PROVIDING QoS guarantees to various applications is an important objective of the next-generation networks. The Internet architecture is based on discrete sets of protocols designed primarily to provide reliable communication over best-effort service model. This was suitable for early Internet applications. Today, enormous popularity of various real-time applications imposes the need for service model with performance guarantees.

The increasing demand for the multimedia applications has triggered a lot of research work in the field of QoS provisioning. These efforts resulted in several proposed service models and mechanisms, including: Integrated Services (IntServ)/Resource Reservation Protocol (RSVP), Differentiated Services (Diffserv) and Multi Protocol Label Switching (MPLS). However, each of them is problematic in its own way and neither one has been widely deployed [1]. Intserv and Diffserv lack means for traffic control in IP networks, since traffic always takes the shortest paths determined by distributed internal and external routing protocols. MPLS provides a partial solution with its traffic engineering capability, however due to inflexibility of the underlying protocols MPLS is also thought to be hard to configure, manage and troubleshoot. For these reasons, QoS problem is often solved by assigning separate physical network, with dedicated hardware and communication protocols, to each class of traffic. Such approach not only requires large investments in infrastructure, but also imposes large operational expenditures, while on the other side the resources are underutilized most of the time.

Having in mind limitations of traditional Internet architecture, characterized by distributed control plane, we addressed problem of QoS provisioning by relying on SDN [2] principles. SDN is relatively new concept of network architecture that clearly decouples the network intelligence from the data (forwarding) plane. The SDN control plane is independent, centralized and programmable. With simple programmatic interface, network administrators can introduce new services and applications, while providing automatic isolation in accordance with their requirements and priorities. On the other hand, SDN protocols such as OpenFlow [3] allow traffic control on per-flow level, which is one of the prerequisites for end-to-end performance guarantees. For these reasons we believe that SDN technology holds a great promise to make QoS more agile.

The main contributions of the paper can be summarized as follows. We presented original design of SDN/OpenFlow control environment that provides bandwidth guarantees for priority flows in automated manner. Priority flows and their requirements only need to be specified, while new controller performs route calculation and resource reservation. To protect best-effort traffic, instead of standard shortest path routing, we proposed new algorithm that use information about resource utilization. We implemented the design and experimentally proved its benefits in comparison with best-effort shortest path routing - which is dominantly deployed on the Internet today, and IntServ - as another example of end-to-end QoS mechanism.

The rest of the paper is organized as follows. Section II gives detailed description of the proposed SDN control environment with reference to implementation challenges that remain to be addressed. Experimental results and analysis are presented in Section III. Conclusion remarks are given in Section IV.

II. QoS SDN Control System

The main design goal for proposed system was enabling service differentiation and efficient use of network resources by automated, fine-grained control. Traditional network architecture is unsuitable for such purpose due to its static and complex nature. This is consequence of distributed control plane, which makes network equipment hard to configure optimally and troubleshoot. Although existing networks can provide differentiated QoS levels for different applications, provisioning of those resources requires a lot of administrator intervention. Consequently, the network is not capable to dynamically adapt to changing application and user requirements. Thus, for example, administrator must configure a large number of

S. Tomovic is with the Research Centre for ICT, Faculty of Electrical Engineering, University of Montenegro, Dzordza Vasingtona bb, 81000 Podgorica, Montenegro (phone:382-69-468583; e-mail: slavicat@ac.me).

N. Prasad is with CTIF-USA, Princeton, New Jersey, USA (e-mail: np@es.aau.dk).

I. Radusinovic with the Research Centre for ICT, Faculty of Electrical Engineering, University of Montenegro, Dzordza Vasingtona bb, 81000 Podgorica, Montenegro (phone: 382-20-245873; e-mail: igorr@ac.me).

devices separately in order to adjust QoS parameters on a per-session or per-application basis.

In order to automate the configuration and management process, proposed system is based on SDN architecture. The network intelligence is shifted from the underlying hardware infrastructure to the centralized, programmable controller. This enables whole network to be treated as a single logical entity, thereby allowing flexibility in configuration and management, as well as optimization of network resources by dynamical, automated SDN routines. Communication between SDN controller and devices in the data plane is achieved via OpenFlow protocol, which identifies traffic flows on the basis of matching rules dynamically or statically determined by the controller.

Available open-source SDN controllers still lack QoS support. Therefore, we propose new QoS SDN/OpenFlow architecture as shown in Fig. 1. The key functional blocks involved are: resource monitoring, route calculation, call admission control and resource reservation. Each of them will be described in detail in the rest of the Section.

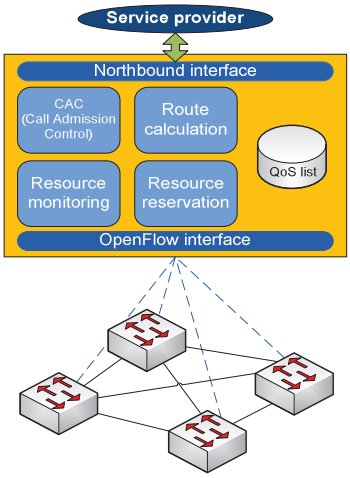

Fig. 1. The proposed QoS SDN architecture

A. Resource monitoring

Resource monitoring module is responsible for collecting and maintaining information about the current state of the network. Beside topology discovery functionality, which is already implemented on today's OpenFlow controllers, we augmented this module with the function for traffic statistics gathering. The controller is programmed to periodically send special OpenFlow request messages for per-flow and per-interface statistics. Counter values reported by the switches provide information about the number of bytes sent over each link and by the each flow. These raw data are used to derive useful information, such as the bandwidth of the flow and link load, which will be later put at disposal to the Route calculation module. Processing of statistics is very demanding since bandwidth calculation has to be recurred for a possible large number of flows and links. In order to relax controller from exhaustive computing as much as possible, only the ingress switch of the flow is queried for flow statistics. That is suitable because of accuracy, since collected statistics will not take packet loss within the network into account. Interval between consecutive measurements is set on 3s. This is a trade-off, because inappropriate values can negatively impact the accuracy of the obtained information. Also, in the case of Open vSwitch [4], we are using in our testbed, counter values are pushed from the kernel to the user-space at the intervals of 1s, so performing measurements that often only unnecessarily congests the controller.

B. Route calculation

Route calculation module determines the routes for different types of traffic. Several routing algorithms can be used in parallel in order to meet different requirements. As input arguments it takes the results of Resource monitoring module and a list of "flows" that require QoS guarantees. In this paper scenario with two classes of traffic was considered: priority traffic (with strict bandwidth requirements) and best-effort traffic.

QoS algorithm calculates optimal route as the shortest route having sufficient amount of bandwidth available. In this way, the delay is trying to be minimized. Another approach would be to choose the least loaded path to balance the network load. However, we have decided for first approach, since for most multimedia applications the delay is highly critical parameter. On the other hand, resource consumption is minimized as well. In order to calculate route, Dijkstra algorithm is applied on graph representing part of the network able to satisfy bandwidth requirement. During making decision, the QoS algorithm identifies available bandwidth as part of the bandwidth that is not already reserved for QoS flows. This can lead to significant performance degradation of best-effort traffic if it is routed across the shortest path. Considering that best-effort is still dominant traffic on the Internet, its routing metric was defined as function of estimated level of links congestion. Again, Dijkstra algorithm is used for route calculation, but weights of the edges are determined according to formula:

$$weight(i,j) = \frac{C(i,j)}{C(i,j) - \max(res(i,j), est(i,j))} \quad (1)$$

where $C(i,j)$ denotes capacity of the link with ingress port i and egress port j, and $res(i,j)$ and $est(i,j)$ reserved and overall occupied bandwidth on the link respectively. Value of $est(i,j)$ is actually output of resource monitoring module, and equals to the estimated link load in the last measurement cycle. In this way, algorithm avoids highly utilized links, even if traffic passing over them is best-effort. The link weight rapidly increases with the utilization, which enables better operation of the heavily loaded network. The maximum function is used in denominator to take in account possible inaccuracy of information obtained by measurement.

Previously described definition of weight function still does not prevent degradation of existing best-effort traffic when new QoS flows arrive. For this reason, the controller has been programmed to reroute the best-effort traffic before congestion occurs. After the path for QoS flow is calculated, for each link on that path controller checks if certain utilization threshold (80 % in our case) is going to be exceeded. If that is the case, the controller calculates amount of bandwidth that should be released in order to bring the link usage below the threshold level. This information is used as input argument of rerouting algorithm. Basically, the algorithm is modification of one

presented in [5]. The main idea is to release required amount of bandwidth by rerouting as few as possible best-effort flows. Although this imposes a significant computational complexity, the modification of flow tables takes far more time than other calculations.

Firstly, the algorithm checks if the required bandwidth can be released by rerouting just a one flow. However, even such flow exists, it is going to be detoured only if there is some path in the network which the flow cannot push into congestion state. Note that before checking for possible migration, link occupancy information must be updated to not account for the load added by the examined flow. If attempt to release desired amount of bandwidth fails, the algorithm tries to achieve the same goal with multiple flows. However, examination again starts from the largest unchecked flow, in order to minimize rerouting.

C. *Resource reservation*

This module provides end-to-end bandwidth guarantees for priority flows by configuring output queues of network devices. *ENQUE* action, defined within OpenFlow protocol, allows packets to be mapped to specific output queue. However, although the controller can send such instructions, queue configuration itself is beyond OpenFlow specification. OF-Config protocol [6] promises to lay foundation on top of which various automated configurations will be possible in the networks running OpenFlow version 1.2 or above. In our OpenFlow 1.0 experimental environment we worked around this limitation by programming controller to remotely send configuration commands. Since our testbed is based on the software OpenFlow switches, these commands actually use built-in capability of Linux systems for traffic shaping. More precisely, the traffic is served with HTB (*Hierarchical Token Bucket* [7]) scheduling algorithm, which allows setting up guaranteed minimum, as well as limiting of maximum rate for the flow. Immediately after controller is launched, resource reservation module creates special buffer for best-effort traffic on each network interface. When the controller receives the packet belonging to one of QoS flows, it creates new buffer on all outgoing interfaces across the calculated route, and configures minimum and maximum rate according to the flow's requirements.

D. *Call Admission Control (CAC)*

CAC module is envisioned to reject QoS requests if there is no condition for fulfillment, and send feedback information to the client. However, northbound interface from the Fig. 2 is not defined yet, since focus was on a proof-of-concept. In current implementation, definitions of QoS flows are directly entered in a specific file whose content the controller periodically checks. Development of interface for communication with content provider and user is important direction of our future work.

III. EXPERIMENTAL RESULTS

The logic described in Section II was implemented on top of the POX controller [8]. Testbed environment used for the experimental evaluation is presented in Fig. 2. The forwarding plane consists of six Linux devices running OpenvSwitch software. They are interconnected in network with multiple disjoint paths for any client-server pair. Capacity of each link is 100Mb/s.

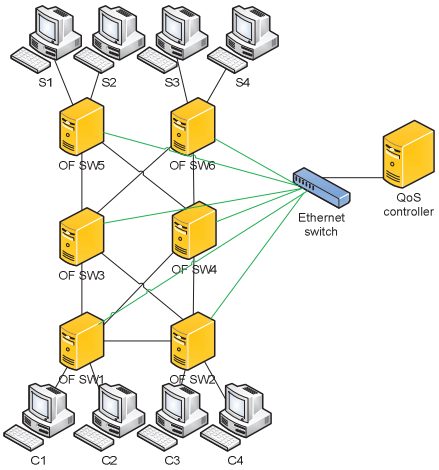

Fig. 2. Testbed environment (S-Server, C-Client)

In implemented OpenFlow setup two experiments were conducted in order to compare the proposed solution with traditional best-effort service model and IntServ-like QoS technique. Best-effort model was imitated by running *l2_multi* application, which comes with the PoX software and performs the shortest path routing. The special application was written to emulate IntServ [9] behavior. Basically, it just enriches the shortest path routing with resource reservation functionality. IntServ in combination with RSVP can provide end-to-end performance guarantees, but RSVP is not routing protocol. That means that resource reservation requests are sent over the path determined in traditional way, i.e. over the shortest one. Consequently, if that path is congested, QoS request can be rejected even some other path to the same destination has sufficient bandwidth available.

In our first experiment 8 UDP flows (Table 1) were generated via iperf [10]. Throughput results are compared in Fig. 3. In IntServ and best-effort scenarios flows experienced worse performance then network could afford. This is consequence of using always the same shortest path tree for routing. IntServ managed to accept only four QoS request (flows 1, 3, 5, 7 from the Table 1) because shortest path trees of different sources overlapped. However, we can see that throughput of those flows also occasionally experienced significant degradation. This happened because all the flows were routed over the switch 4, which was struggling from time to time due to its software limitations. On the other side, the new application exploited path diversity and provided the required QoS level for each flow.

TABLE 1: CHARACTERISTICS OF FLOWS IN EXPERIMENT 1

Flow	*Src.-Dst.*	*Prot.*	*Rate*	*Type*
1.	C3-S1	UDP	35Mb/s	QoS
2.	C4-S2	UDP	60Mb/s	QoS
3.	C1-S1	UDP	40Mb/s	QoS
4.	C2-S4	UDP	70Mb/s	QoS
5.	C3-S3	UDP	40Mb/s	QoS
6.	C4-S4	UDP	10Mb/s	QoS
7.	C1-S3	UDP	45Mb/s	QoS
8.	C2-S2	UDP	15Mb/s	QoS

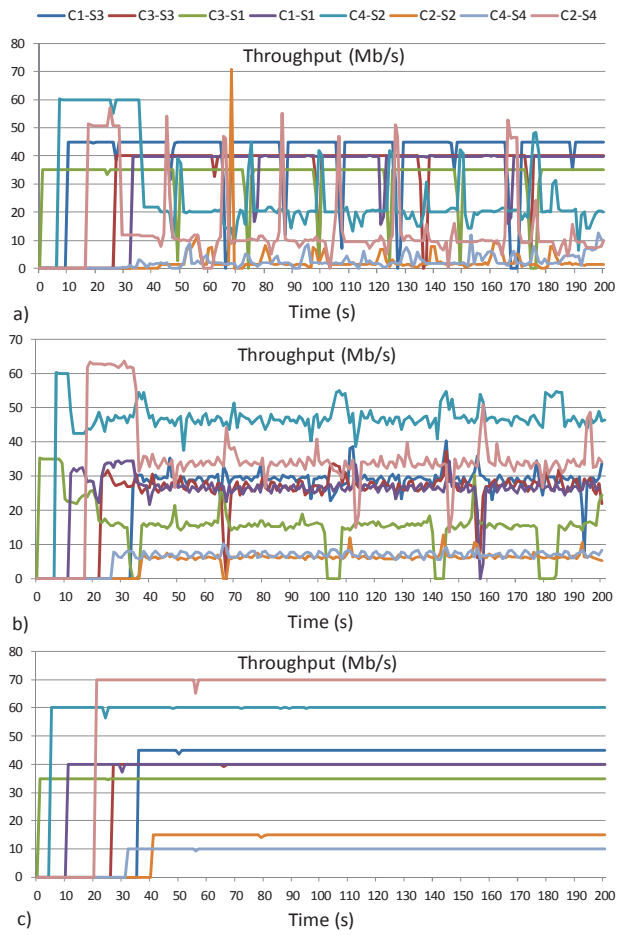

Fig. 3. Achieved throughputs in experiment 1: a) best-effort, b) IntServ, c) proposed QoS-aware app.

In second experiment we examined an influence of the proposed QoS solution and IntServ model on best-effort traffic performance. At the beginning, 6 best-effort flows were generated one by one in order defined by Table 2. After that, 45Mb/s priority flow was initiated. In IntServ scenario, all the flows were routed over the same route, which resulted in significant performance degradation of best-effort traffic after introducing the priority flow (Fig. 4 (a)). Our solution distributed best-effort load equally over OF1-OF3-OF5 and OF1-OF4-OF5 routes (51Mb/s on each). This was achieved by using routing mechanism described in Section II. Later, one of the loaded paths was selected for priority flow. However, the controller promptly identified that congestion threshold (80Mbit/s) is going to be exceeded and run rerouting algorithm. According to described logic, 20 Mb/s flow was moved to alternative path. Achieved improvements are clearly shown in Fig. 4.

TABLE 2: CHARACTERISTICS OF FLOWS IN EXPERIMENT 2

Flow	*Src.- Dst.*	*Transp. Prot.*	*Rate*	*Type*
1.	C1-S1	UDP	30Mb/s	Best-effort
2.	C2-S2	UDP	30Mb/s	Best-effort
3.	C1-S1	UDP	20Mb/s	Best-effort
4.	C2-S2	UDP	20Mb/s	Best-effort
5.	C1-S1	UDP	1Mb/s	Best-effort
6.	C2-S2	UDP	1Mb/s	Best-effort
7.	C1-S1	UDP	45Mb/s	QoS

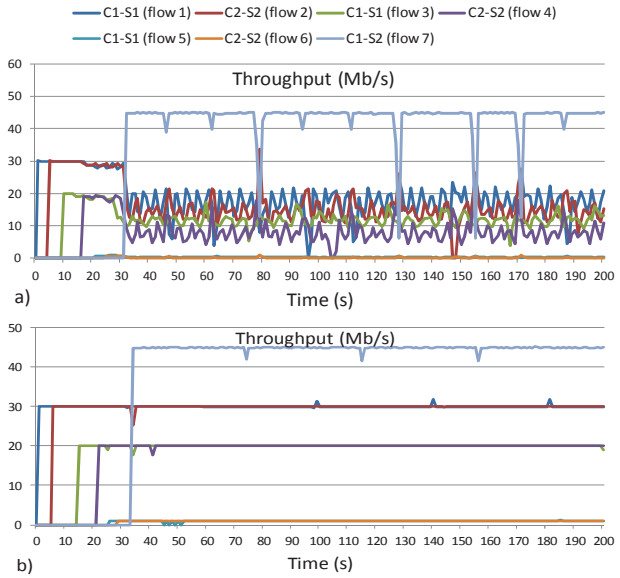

Fig. 4. Achieved throughputs in experiment 2: a) IntServ, b) proposed QoS-aware app.

IV. CONCLUSION

In this paper we presented original design of SDN/OpenFlow control environment that provides bandwidth guarantees for priority flows in automated manner. In order to protect best-effort traffic, instead of standard shortest path routing, we proposed new algorithm that use information about resource utilization. It is shown that proposed solution outperforms best-effort shortest path routing and IntServ. Our future research will involve development of Call Admission Control module with user interface, performance analysis of other QoS routing algorithms and finding solution for guaranteeing delay.

ACKNOWLEDGMENT

This work is supported by the Ministry of Science of Montenegro under grant 01-451/2012 (FIRMONT) and EU FP7 project Fore-Mont (Grant Agreement No. 315970 FP7-REGPOT-CT-2013) http://www.foremont.ac.me.

REFERENCES

[1] H. Eglimez, S. Civanlar, A.M. Tekalp, "An optimization framework for QoS-enabled adaptive video streaming over OpenFlow networks," *IEEE Transaction on multimedia*, vol.15, no.3, pp.710-715, Apr. 2013

[2] Open Networking Foundation, "Software Defined Networking: the new norm for networks," Web. White Paper, Retrieved Apr. 2014.

[3] OpenFlow switch specification v1.0.0, Retrieved Apr. 2014, Available at: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf.

[4] OpenvSwitch. [Online]. Available at: htttp://openvswitch.org

[5] Tim Henrinckx, "Dynamic and performance driven control for OpenFlow networks", Master thesis, Faculty of Engineering and Architecture, Ghent Univ., June 2013.

[6] Open Networking Foundation, "OpenFlow Management and Configuration Protocol 1.2", Retrieved Sept. 2014, Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf

[7] J. L. Valenzuela, A. Monleon, I. San Esteban, M. Portoles, O. Sallent, "A Hierarchical token bucket algorithm to enhance QoS in 802.11: Proposal, Implementation and Evaluation," VTC2004-Fall, vol. 4, pp. 2659-2662, 26-29 Sept. 2004.

[8] PoX. [Online]. Available at: http://noxrepo.org/pox/about-pox/

[9] *Integrated services in the Internet architecture: An overview*, IETF, RFC 1633, June 1994.

[10] Iperf - The TCP/UDP bandwidth measurement tool. [Online]. Available at: http://iperf.sourceforge.net