# On demand QoS with a SDN Traffic Engineering Management (STEM) module

Cedric Morin*,†,‡, Géraldine Texier *†, Cao-Thanh Phan†
*IMT Atlantique/IRISA/Adopnet, France; first.lastname@imt-atlantique.fr
†BCOM, France; first.lastname@b-com.com
‡TDF, France; first.lastname@tdf.fr

***Abstract*—Software Defined Networking (SDN) allows new approaches to provide Quality of Service (QoS). In legacy networks, strict QoS guarantees often result in bandwidth over-provisioning. Then, QoS enforcement either consumes too many resources, or is not flexible enough. We present a solution to provide QoS based on the creation of on-demand MPLS tunnels with guaranteed bandwidths across an SDN network. We introduce an SDN Traffic Engineering Management (STEM) module that interacts with the northbound applications to satisfy their requests to forward QoS-guaranteed traffic flows. STEM delegates the path selection to a Path Computation Element (PCE), and the path enforcement to an SDN controller. We rely on a stateful PCE to record the attributed resources and estimate the remaining network capacity, avoiding overloading the network with monitoring traffic. Upon STEM requests, the SDN controller enforces the QoS policy in the data plane. User flows are aggregated into MPLS tunnels and packets are labeled with a priority depending on the flow effective bandwidth. We highlight the shortfalls of several material and software OpenFlow compatible switches and detail an implementation based on a pica8 switch to overcome them. The experimental results demonstrate that this solution efficiently enforces bandwidth sharing in SDN networks.**

***Index Terms*—SDN ; Traffic engineering ; QoS**

## I. Introduction

New 5G standards such as high peak data rate, high user experienced data rate or very low latency will impose a fine and dynamic control of QoS parameters (bandwidth, latency, loss or jitter). Legacy networks are facing a lot of difficulties trying to comply with such needs. Traditionally, protocols such as IntServ or DiffServ have been used to introduce a QoS management in the network. However those protocols are either fine grained but require complex implementation, or simple to deploy but coarse grained.

Over the past few years a new architecture emerged for communication networks: Software Defined Networking (SDN) [1]. To enable the softwarization of the network, the control plane and the forwarding plane are separated. The control plane is logically centralized in an element called controller, although it can be distributed, for scalability reasons for example. The controller has an holistic vision of the network topology and resources (bandwidth, link delay, CPU ...). With this global perception, the control plane is able to address QoS problems from end to end, without facing rule propagation delays and path computation latency experienced by distributed networks. Moreover, the controller can manage efficiently the resources of the network. Relationships between each component of the SDN architecture and traffic engineering services (TE) have raised a lot of interest in network research [2].

We explore the on the fly bandwidth reservation scenario raised by network service providers such as Telediffusion de France (TDF). Clients request a given amount of bandwidth for a restricted period of time to connect two locations. This situation may occur when a client wants to organize a punctual massive data transfer between two data centers, or when a client such as a stadium, a theater, or any actor of the events industry requires a temporary connection between its location and a data center. As implemented by TDF, clients must precise a Committed Information Rate (CIR), which will be their guaranteed amount of available bandwidth at any time. They also indicate a Peak Information Rate (PIR) that represents the maximum bandwidth they may reach (typically the size of the physical link), but which may not be guaranteed any time. They can also express latency constraints.

To meet these needs, this paper presents the SDN Traffic Engineering Management (STEM) module. In an SDN network, this module provides on the fly bandwidth allocation for users, without requiring operator's intervention. Network resources are managed by a Path Computation Element (PCE) located in the control plane. The PCE memorizes allocated resources, and does not have to poll the network equipment to gather information. To enforce bandwidth allocation policy, STEM uses a fixed number of queues and the meter tool. Packets' IP headers are not modified. In our experimentation we highlight the shortfalls of several material and software OpenFlow compatible switches, and detail an implementation based on a pica8 switch to overcome them and demonstrate the efficiency of our system. The rest of the paper is organized as follows. Section II presents the related works on QoS solutions in SDN networks. Section III details our solution based on MPLS. Its implementation and its validation through experimentation are related in section IV. Finally, section V draws a general conclusion.

## II. Related Works

SDN architecture has motivated a lot of new designs and solutions to perform traffic engineering and provide desired QoS to applications [3]. The QoS enforcement can be divided into three actions: a request reception (the network service indicates the QoS parameters required by the new flow), the path selection (the control layer determines the most suitable path for the flow) and the path enforcement (the switches or routers are configured to handle packets with respect to their priority and latency constraints).

The selection of a path satisfying QoS constraints forces the control plane to maintain an accurate representation of the network's available resources. In [4], the authors consider that QoS packets must not be dropped, and force network equipment to send a warning when such event occurs. This approach is efficient, but valid only for highly critical flows. In the literature, most of the solutions rely on network statistic gathering techniques to build, and periodically update, their knowledge of the current network traffic, and consumed resources [5] [6]. However, the constant fluctuation of the traffic load implies frequent updates of the statistics. First, this leads to an important augmentation of control traffic. Second, it generates a heavy work load for the controller and requires a lot of computational power. Such aspects are not taken into account, as most of the experiments in the literature consider only small topologies, with reduced number of flows. However, in [7] the authors point out that, in a production network, an excess of statistic polling could dramatically overload the control plane (both switch CPU and controller), and increase the response time of the switch. Third, the reaction time of the system does not guarantee that QoS contracts will be respected anytime, especially between two polls, and worth if the controller is overloaded).

Upon traffic changes (*e.g.,* the apparition of concurrent flows), QoS enforcement might result in flow rerouting to take into account the variation of the available resources, as detailed in [8]. While literature is very important about legacy networks, few solutions have been adapted to the SDN architecture. The proposed approaches are either reactive or proactive. Reactive approaches are based on measures to adapt to QoS shortage detection. [9] uses path diversity: several paths are pre-calculated for a given flow, so that it can quickly switch from one to another when QoS is no longer respected. This results either in over-provisioning or non guaranteed QoS. Proactive approaches can be implemented by resource reservation. Casellas *et al.* [10] use this technique to allocate optical wavelengths in optical networks, which cannot be done using statistics. This method can be extended to QoS parameters.

The most common tools to enforce QoS in the data plane are queuing disciplines. Queues allow to shape and prioritize traffic, to share the bandwidth and control the latency. Most of the solutions use pre-defined queues: they are pushed once and for all in the network equipment, and are not modified afterwards. This reduces the flexibility of the QoS offer, since the number of queues is limited. To solve this, some solutions dynamically create queues depending on the needs [11] [6]. Although it enables to control the QoS with a very fine grain, it includes two major drawbacks. First, in real switches the number of available queues is limited. For instance, the PICA8 P-3295[1] has only 8 priority queues per port , which is not enough to handle all the flows going through a port of a production network equipment. Second, the queue creation mechanism is associated to switch configuration, not to flow rules configuration. For this reason the timescales to create a flow rule and a queue are different, as specified for example in the OpenFlow (OF) documentation [12]. Adding queue creation to flow establishment will considerably slow down the process. To bring flexibility without having to create a lot of queues, [13] proposes a solution based on the meter tool. Meter is a switch element that can measure and control the rate of packets[2]. For each flow, one meter is placed in the entry point of the network. Meter counts the IP packets, and alters the Differentiated Service Code Point (DSCP) field in their IP header. The new DSCP can take 3 values: a priority value (if the flow respects its allocated bandwidth), a non-priority value (if the flow exceeds its allocated bandwidth), or a default value (if the flow did not require any specific bandwidth, *e.g.,* Best Effort flows). The main drawback is that the original DSCP value is lost during the transfer, which may impact the treatment of the packet after it exits the network.

We propose a solution to guarantee QoS anytime and provide bandwidth allocation across an SDN network involving the cooperation of the control plane with a PCE to identify suitable paths and reserve the resources in the data plane. To address scalability issues, we use a predefined number of queues, and we avoid the constant polling required by statistics gathering thanks to the memorization of allocated resources. Flexibility is provided by traffic classification using the meter tool, as in [13]. In order to preserve all the fields of the transported IP packet, the QoS information will be stored in a Multi Protocol Label Switched (MPLS) header, not in the IP header itself.

## III. Proposed solution

We aim at enforcing QoS policies with the following objectives. The northbound application specifies cost and bandwidth constraints. QoS must be guaranteed anytime. Network resources should be fully exploited, reducing wasted bandwidth to the minimum. The control plane

[1]http://www.pica8.com/documents/ pica8-datasheet-48x1gbe-p3290-p3295-v1.9.pdf

[2]https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf

must not be flooded by information. Queues are not dynamically created. Their number depends on the precision requested by the QoS policy, and is limited by the equipment's capabilities. To meet these objectives the solution must address resource management and packet handling. The full architecture, depicted in Figure 1, impacts mostly the control plane and the data plane, denoted as the infrastructure layer.

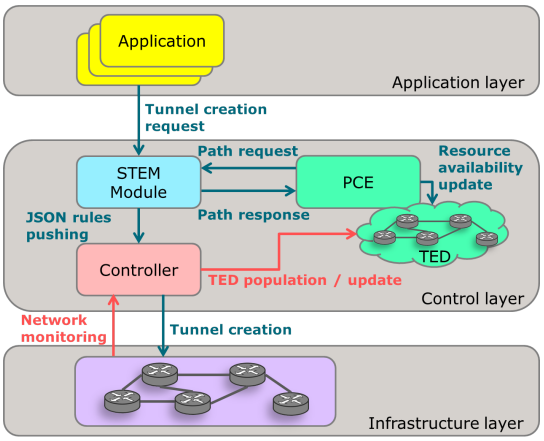



Figure 1: Architecture overview

In the application layer, a northbound application emits tunnel creation requests. These requests includes all the necessary information to identify a flow, typically the five-tuple: source IP address, destination IP address, source port, destination port, upper layer protocol. However, other information (*e.g.,* DSCP, Ethernet address or VLAN ids) can be used. The request also specifies the desired QoS parameters. Currently the supported parameters are bandwidth, hop count or cost, but bounds on latency or packet losses may be added afterwards.

The requests are handled by the control layer, composed of three modules: an SDN controller, a PCE and our STEM (SDN Traffic Engineering Management) module. STEM receives tunnel creation requests, interprets them, and send them to the PCE. The PCE calculates a suitable path across the network to satisfy the request, and returns it to the STEM module. STEM translates the path into flow rules transmitted to the controller. The SDN controller gathers the network topology information and enforces the flow rules in the data plane switches, two basic features provided by any major SDN controller implementation. These commands are received and interpreted by network's equipment (switches or routers).

### *A. The Path computation element (PCE)*

Widely studied and standardized in its stateless version [14], the PCE is used in production networks to calculate routes, possibly under multiple constraints. A PCE receives path computation requests from a Path Computation Client (PCC) through the Path Computation Element Protocol (PCEP). To be able to compute paths, the PCE must know the topology and the current load on each link of the network and populate its Traffic Engineering Database (TED) accordingly. In SDN, the best way to acquire the network topology is to interconnect the PCE and the controller. There are several ways to achieve this goal, depending on the architecture in which the PCE is inserted. The PCE and the controller interconnection can be performed in three ways, as detailed in [15]: either integrated or external to the controller, or the PCE is seen as an application. As shown in Figure 1 the PCE here is used as an application. The STEM module plays the role of PCC. This configuration has been chosen for various reasons:

- Simplicity: the controller does not have to implement PCEP.
- Encapsulation: since the controller does not implement new modules (*e.g.,* PCE or PCC) the application can use almost any controller out of the box, with only small adjustments at northbound interface level. For the same reason the PCE can be changed or updated very easily.
- Security: as a consequence of a better isolation, separating the different components of the control layer is a good practice to enhance system security [16].
- Maintainability: the separation of the possibly complex controller from STEM simplifies maintenance and reduces potential interference or update problems.

Instead of using flow statistic polling, the control layer relies on a stateful PCE [17] to secure flows' reserved resources. A stateless PCE populates its TED and then performs path computation based on it, without recording the resource changes. As a consequence, resources in use can be reallocated several times, thus QoS may no longer be ensured. On the contrary, a stateful PCE keeps track of the allocated resources by recording the computed routes and the associated QoS requirements. This mechanism guarantees that the same resource will not be allocated twice at control plane level (unless overbooking is explicitly allowed). Note that every flow is tracked. STEMinstalls rules only for accepted flows, any other traffic is dropped.

The PCE does not rely on a specific algorithm to perform its path calculation. The choice is determined by the number of additive QoS constraints requested. Our PCE uses SAMCRA algorithm [18] to compute path under multiple constraints, Dijkstra otherwise.

### *B. The SDN Traffic Engineering Management (STEM) module*

The STEM module exposes a northbound interface to the applications that want to reserve bandwidth. When a request is received, the module turns it into an appropriate PCEP request and submit it to the PCE. Once the path is computed, it creates the appropriate set of flow rules and send it to the controller. STEM module does not only create the direct IP path from source to destination: it also

handles the way back and the corresponding ARP tunnels, required for IP communication. Other automatic services can be added, depending on the needs.

The STEM module creates tunnels called Label Switched Paths (LSP) between two points in the network using MPLS, a protocol widely used to forward traffic in legacy networks and offering a traffic engineering functionality. Due to the reduced size of its header, an MPLS packet can be treated very fast in switches and routers. The MPLS header is formed by a stack of labels, each entry is composed by a label field (20 bits), a Traffic Control (TC) field (3 bits), a bottom-of-stack field (1 bit) and a time to live (8 bits). This header is placed between the level 2 and 3 headers (Ethernet and IP in this case). MPLS encapsulates the IP packets, which allows to transport them without alteration. In particular, the TC field can be used to store the packet priority, the DSCP field doesn't have to be modified.

STEM designs rules to enforce the QoS policy using priority queues and meters. First, we set the packet's priority (in the TC field) of each QoS flow. To do so, a meter is created in the network entry equipment to count the corresponding packets. When the bit rate exceeds the defined Guaranteed Bit Rate (GBR), meter triggers an action to increment packet's MPLS TC field, reducing it's priority. Best Effort packets have a medium priority. Then, we assign to each packet a label that identifies the flow they belong to. Label and TC settings only occur in the entry switch. Other switches simply match label and TC fields to select the output port and queue respectively. We fixed the number of priority values (High, medium and low), requiring only 3 queues in each port. However, additional priority levels might be added if more queues are available, for further latency and loss management. Note that, as opposed to queues, meters can be created on the fly at the same timescale as flow rules. Moreover, only one meter per flow is requested, not one per flow and per switch. However, a study of meter's impact on switches' performance needs to be performed.

## IV. Implementation and experimentation

### A. Implementation

We implemented a platform to evaluate our bandwidth management solution for SDN networks based on MPLS tunneling and the meter tool. The application layer is simply implemented by a REST client. The implementation of other layers requires the support of OpenFlow, MPLS, queues and meter. It turned out to be complicated, especially the meter tool.

*1) Control plane:* In the control layer, the SDN controller is an unmodified OpenDaylight (ODL) controller Lithium SR4 using OpenFlow 1.3 (OF13) to communicate with the data plane[3].

The PCE is based on Netphony PCE[4]. The original software has been modified to fully support resource reservation (stateful PCE). The PCEP protocol has been extended to request several different metrics such as cost or latency. The PCE is currently stateful passive: the resources are reserved upon a path validation and are only released by addressing an explicit request to the PCE. To improve network usage, the PCE should be stateful active: it may then propose modifications of already existing paths to optimize resource distribution.

*2) Data plane:* In the literature, the data plane usually integrates a network of OpenV Switches (OVS)[5]. Although these virtual switches are very efficient, they do not implement meter yet, which excludes them from our platform implementation. The OpenFlow SoftSwitch[6] (also named CPqD) implements meter and presents two advantages. First, it is implemented in Mininet, allowing to test the solution on different topologies. Second, it implements all the rules needed for our solution to encapsulate IP packets in an MPLS packets and to set the label and TC fields in the MPLS header. The main difficulty is that meter can only increment the IP DSCP field. The Figure 2a presents the solution used by STEM.

We can extend the rules to detect incoming MPLS packets entering the edge switches by adding an "in_port" match option in the first table. Our MPLS label would be stacked on top of the existing one(s), and popped at the end of the tunnel. The tables that contain the higher numbers of rules (the ones that have to match the 64 possible values of DSCP) are static: they are pushed once and for all when the switch is connected to the network, and do not have to be pushed for each flow afterwards. Unfortunately, the CPqD switch is not able to push the MPLS header and to set the MPLS fields value in the same rule. If it was possible (as in OVS), tables 4/7, 5/8 and 6/9 could be merged. These rules have been tested, the MPLS headers were successfully set. However, CPqD's queuing mechanism does not perform well, and it is impossible to carry experimentation on bandwidth reservation.

Finally, we used a PICA8 switch P-3295 with Linux System Version 2.6.4 and OVS/OF Version 2.6.4. Pica8 is a traditional L2/L3 switch that implements OpenFlow protocol, to act as an OVS, and both meter and queues. However, OpenFlow is only implemented at software level, and hardly reflected at hardware level, where all packets are processed. In particular, the switch has only one physical ternary content-addressable memory (TCAM) and cannot implement more than one table. To overcome this limitation, the switch merges the rules to fit in the unique hardware table when multiple tables are used at the software level. Such operation is impossible if a rule updates flow parameters matched by another rule. In this case the resulting hardware rule is erratic, and most of the time results in dropping the packet. For the same reason, Pica8

[3]https://www.opendaylight.org/

[4]https://github.com/telefonicaid/netphony-pce

[5]http://openvswitch.org/

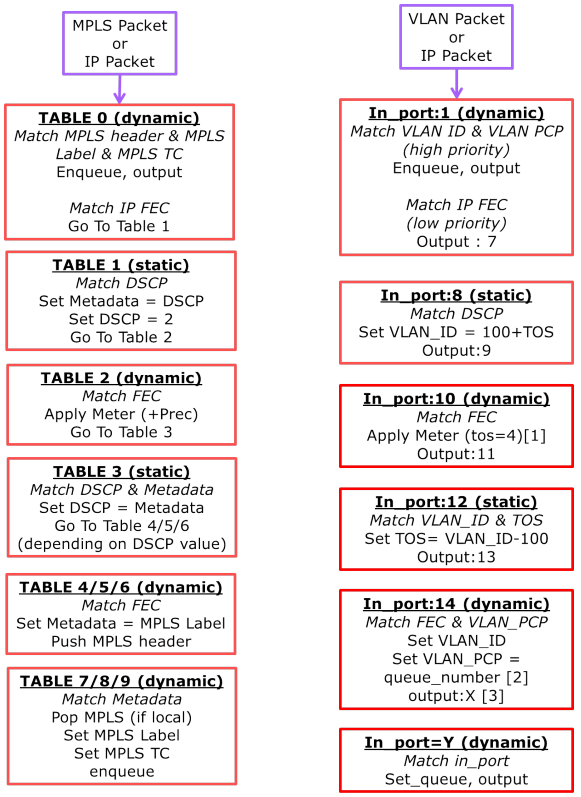
[6]https://github.com/CPqD/ofsoftswitch13

does not implement the bridge system: bridges declared at software level are ignored at hardware level. In [13], authors dealt with these issues by using multiple switches, each switch playing the role of one table. Moreover, P-3295 does not implements MPLS treatment at hardware level, but only at software level, which strongly impacts performances over 10Mb of traffic. Last, no tables implies no metadatas, which are necessary in our implementation, as shown in Figure 2a.

To overcome these limitations, we emulated multiple tables by encapsulating the IP packet. MPLS encapsulation cannot be used for this purpose because, when a packet is encapsulated in MPLS, it cannot be matched based on it's IP fields anymore. We envisioned to carry out the experiment using VLANs: the IP packet is encapsulated in a VLAN when entering the switch. The VLAN ID field represents the metadata, while the VLAN PCP field represents the table. Once the packet is treated, the VLAN PCP is incremented and the packet is resubmitted to its entry port to be treated again as a new packet. In addition to their "normal" matching options, all the rules will also match the PCP field. The advantage of this solution is that it uses only one port and provides an equivalent to the missing metadata mechanism. The drawback is that VLAN cannot be used by the incoming flows.

Due to the Pica8 limitations regarding resubmit action, the VLAN solution couldn't be used to replace all the tables: "table 3" couldn't match the new TOS resulting of the meter action. We tried to bypass this by using several ports to act as different tables. The input port acts as a table identifier. The main drawback of this solution is that multiple ports are used to perform a single set of actions. However, this solution works well on Pica8 and was used to carry out the experiment, in parallel with the VLAN encapsulation to emulate metadata retention. Eventually, MPLS was replaced by VLAN in the experiment due to MPLS lack of efficiency evoked above. This has two main drawbacks in real applications: the encapsulated traffic cannot be a VLAN itself, and the number of possible tunnels is reduced as the number of available VLAN IDs is lower than the number of possible MPLS labels.

We insist on the fact that we used multiple ports to emulate tables, and VLAN tags instead of MPLS labels, because of hardware and software limitations. The resulting solution is not as efficient as the theoretical solution but it highlights that meter can be used to successfully share the bandwidth among multiple flows.




(a) CPqD (b) Pica

*[1] misbehaviour : the meter does not increment precedency but re-write the DSCP field*
*[2] bug : queues re-write the vlan_pcp. It must be set equal to queue number*
*[3] bug : If traffic is aggregated Pica8 queue misbehaves. Each flow is directed to a different port, then re-aggregated before entering the queue*

Figure 2: OpenFlow rules

## B. Experimentation

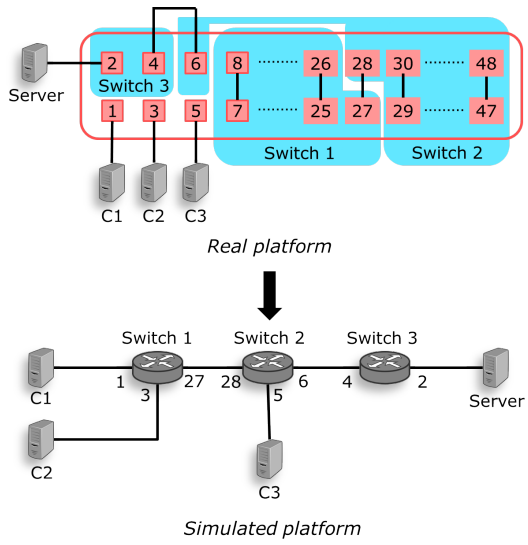



Figure 3: Test platform

We implemented a platform with one Pica8 P-3295 switch, three clients (C1, C2 and C3) and one server. From one physical switch we created the equivalent of a 3 switch topology, as shown in Figure 3. The corresponding rules are presented Figure 2b. In order to assess the efficiency of our bandwidth management mechanism, we designed the following scenario. We generated traffic with the Iperf tool according to the following pattern: between time t=0s to 60s, C1 generates 100Mb of Best Effort traffic; between

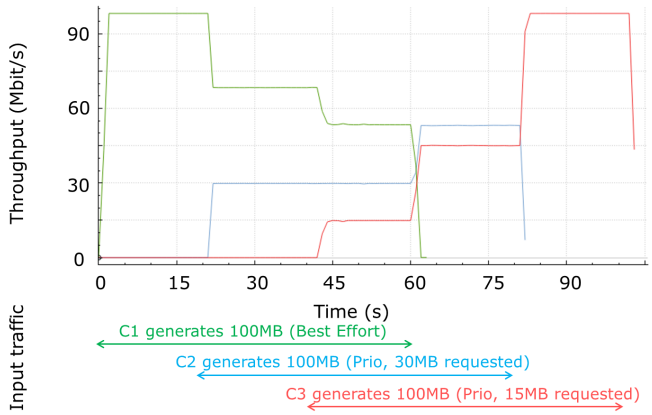



Figure 4: Experimental results

t=20s and 80s, C2 generates 100Mb of priority 2 traffic with a requested bandwidth of 30Mb; between t=40s and t=100s, C3 generates 100Mb of priority 1 traffic, with a requested bandwidth of 15Mb. The ARP messages corresponding to these traffics are carried in separated tunnels, with priority 3 (the highest).

Results are presented in Figure 4 and demonstrate the efficiency of the STEM module. We can observe that at the beginning, only the Best Effort traffic is present and occupies the entire link bandwidth. But, upon the C2 traffic arrival, 30Mb of priority traffic are transmitted, while the remaining 70Mb are blocked. This is normal since they have lower priority than the best effort. The same situation occurs when C3 starts emitting. At t=60, when C1's traffic stops, the exceeding traffics of C2 and C3 compete for the extra bandwidth, with equal priority, which results in fair division of the resource. In the end, only C3 remains, and uses the whole link capacity. The flows behave accordingly to QoS policy. In particular, the link is always used at full capacity, even if the bandwidth is never fully reserved: no capacity is lost.

## V. Conclusion

We presented a full solution to provide guaranteed bandwidth to flows across an SDN network that respects the integrity of the transported packets. At the control plane level, we introduced the module STEM to manage any northbound application's QoS-guaranteed path requests. It delegates traffic engineering decisions to a PCE and the path enforcement to an SDN controller. The combined use of a stateful PCE and the meter tool allows network clients to reserve flexible amounts of bandwidth, guaranteed anytime, without constantly polling the network to gather flow statistics. We exposed how difficult it is to find off-the-shelf equipment able to support the needed flow rules. This highlights the necessary improvement of hardware and software OpenFlow-capable devices. Although meter is a known tool in OpenFlow, it is not widely supported yet. In future works, we would like to change the stateful passive PCE into a stateful active one to keep the network monitoring light and optimize resource usage. Furthermore, we plan to extend the considered QoS metrics (currently limited to bandwidth and cost) with latency and loss considerations.

## References


[1] "Software-Defined Networking (SDN) Definition - Open Networking Foundation." [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition

[2] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of Software-Defined Networking to Traffic Engineering," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2016.

[3] M. Karakus and A. Durresi, "Quality of Service (QoS) in Software Defined Networking (SDN): A survey," *Journal of Network and Computer Applications*, vol. 80, pp. 200–218, 2017.

[4] 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)*. IEEE, 2010, pp. 351–356.

[5] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and Scalable QoS Control for Network Convergence." *INM/WREN*, vol. 10, no. 1, 2010.

[6] S. Tomovic, N. Prasad, and I. Radusinovic, "SDN control framework for QoS provisioning," in *Telecommunications Forum Telfor (FOR), 2014 22nd*. IEEE, 2014, pp. 111–114.

[7] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.

[8] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "PolicyCop: An autonomic QoS policy enforcement framework for software defined networks," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*. IEEE, 2013, pp. 1–7.

[9] Y. Jinyao, Z. Hailong, S. Qianjun, L. Bo, and G. Xiao, "HiQoS: An SDN-based multipath QoS solution," *China Communications*, vol. 12, no. 5, pp. 123–133, 2015.

[10] R. Casellas, R. Muñoz, R. Martínez, R. Vilalta, L. Liu, T. Tsuritani, I. Morita, V. López, O. González de Dios, and J. P. Fernández-Palacios, "SDN Orchestration of OpenFlow and GMPLS Flexi-Grid Networks With a Stateful Hierarchical PCE [Invited]," *Journal of Optical Communications and Networking*, vol. 7, no. 1, p. A106, Jan. 2015.

[11] K. Govindarajan, K. C. Meng, H. Ong, W. M. Tat, S. Sivanand, and L. S. Leong, "Realizing the quality of service (QoS) in software-defined networking (SDN) based cloud infrastructure," in *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*. IEEE, 2014, pp. 505–510.

[12] "OF-CONFIG 1.2 OpenFlow Management and Configuration Protocol," 2014.

[13] H. Krishna, N. L. van Adrichem, and F. A. Kuipers, "Providing bandwidth guarantees with OpenFlow," in *Communications and Vehicular Technologies (SCVT), 2016 Symposium on*. IEEE, 2016, pp. 1–6.

[14] J. L. Le Roux, "Path computation element (PCE) communication protocol (PCEP)," 2009.

[15] R. Casellas, R. Martínez, R. Muñoz, R. Vilalta, L. Liu, T. Tsuritani, and I. Morita, "Control and management of flexi-grid optical networks with an integrated stateful path computation element and OpenFlow controller [invited]," *Journal of Optical Communications and Networking*, vol. 5, no. 10, pp. A57–A65, 2013.

[16] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A Survey of Security in Software Defined Networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[17] E. Crabbe, I. Minei, J. Medved, and R. Varga, "PCEP Extensions for Stateful PCE," Internet Engineering Task Force, Internet-Draft draft-ietf-pce-stateful-pce-19, May 2017, work in Progress.

[18] P. Van Mieghem and F. Kuipers, "SAMCRA - Concepts of Exact QoS Routing Algorithms," *IEEE/ACM Transactions on networking*, vol. 12, no. 5, pp. 851–863, May 2004.