Survey of Concepts for QoS improvements via SDN

Atanas Mirchev
Supervisor: Lukas Schwaighofer, Daniel Raumer
Seminar Future Internet SS2015
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: ga75lar@mytum.de

ABSTRACT

Considering the current state of the Internet, the available approaches to providing Quality of Service (QoS) for single services or specific tenants are limited and inflexible. Providers need a better solution that is scalable and that allows fine-grained tuning of network traffic. Recently, different SDN enabled QoS frameworks have emerged, offering a lot of possibilities for network reconfiguration and high level definition of policies. This paper gives a detailed summary of available QoS mechanisms that rely on SDN and compares their strengths and weaknesses. In the spirit of the survey, novel ideas for the future of QoS are discussed. SDN brings a lot of freedom and monitoring possibilities as a QoS domain, but the Internet has yet to adapt to this innovative concept: at the time of writing most of the existing solutions are prototypes.

Keywords

Software Defined Networking, Quality of Service, QoS via SDN, OpenFlow, OpenQoS, FlowQoS, QoS methods, SDN Approaches to QoS

1. INTRODUCTION

Today's networking consists of discrete sets of protocols that specify how hosts in different networks can be connected reliably. However, protocols tend to be defined in isolation and are designed to solve a specific problem. This results in high network complexity, which is a major limitation with regard to overall networking and providing Quality of Service (QoS). Due to this fact, today's networks are static. Typically all of the control decisions (e.g. how packets should be forwarded) are taken at the separate forwarding devices. This proves to be an obstacle considering the dynamic nature of QoS: IT administrators must configure each vendor's equipment separately, adjusting parameters (such as bandwidth) to meet the predefined rules and policies. This approach cannot dynamically adapt to the constantly changing application and user demands.

To tackle this problem, the concept of Software Defined Networking [1] has emerged. It focuses on decoupling the control plane from the forwarding plane (*Figure 1*), therefore leaving the existing routers and switches as simple forwarding devices. The control logic is instead centralized and deployed on a server (commodity hardware), called an SDN controller, which allows for easier network management and monitoring, while improving extensibility and scalability possibilities. The **Northbound API** (c.f. *Figure 1*) of an SDN controller is the public interface that other higher level applications can access. It conceals the actual implementation and is used to dynamically define abstract rules which are then enforced on the network by the controller. This also includes control policies and setting up traffic priorities that can be used for QoS. The **Southbound API** (c.f. *Figure 1*) is the interface between the controller and the forwarding plane. The standard protocol for communication over this API is called OpenFlow. It enables the SDN controllers to dynamically configure all forwarding devices and allows for more sophisticated traffic management. The configuration is done by defining rules and actions (called "flow table entries") in a switch's flow table. Those can then periodically or on demand be changed by the controller, according to the chosen policy. The protocol also specifies that new unmatched packets should be redirected through the controller (so that it can identify them and assign new rules to the forwarding elements).

"OpenFlow Configuration and Management Protocol" (OF-Config) and the "Open vSwitch Database Management Protocol" (OVSDB) act as specific extensions to OpenFlow and help the controller to configure the forwarding devices according to the defined rules.

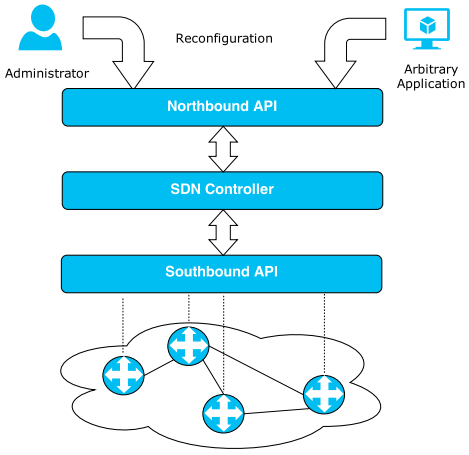

Figure 1: Simplified representation of the SDN architecture

In the context of this paper, an important part of networking is the overall performance of a connection, called Quality of Service. QoS comprises requirements on all major aspects of

doi: 10.2313/NET-2015-09-1_05

data transmission, such as response time, jitter, interrupts, etc. In order to provide QoS, application flows[1] need to be differentiated as they compete for bandwidth. Then network resources have to be allocated to ensure the precedence of the higher-priority traffic. That allows for the appropriate network resource distribution. This process often requires knowledge of the current network state, so that the right decisions with regard to packet forwarding can be made. That is why network monitoring is an important aspect of QoS. Ultimately, the goal is to also ensure network convergence, meaning that the network resources should be utilized as evenly as possible.

Without consideration of the available SDN methods, QoS relies on end-to-end agreements between hosts and Service-level Agreements (SLAs) between provider and user. While this approach is robust, it is best suited for best-effort services and doesn't allow for fine grained traffic engineering. Multimedia streaming and VoIP flows, on the other hand, require timely delivery over robustness and must be handled separately. The current "hop-by-hop decision" architecture of the Internet is sometimes difficult to monitor, mainly because of the many different vendor-specific firmwares at use [6]. There is no standardized way for specifying high level traffic control policies and restrictions with regard to the depth of traffic differentiation exist.

With the introduction of the SDN controller, a decoupled control plane is established. Besides translating the requirements of the application layer, it provides applications with an abstract view of the network. The network state is obtained for example via sampling of packets passing through the controller and can consist of different data, such as statistics or events. Using this information, control policies and SLAs can be specified by an administrator at a higher abstraction level and even dynamically adjusted. These mechanisms have proven to be beneficial for QoS. [2] [4] [8].

In the rest of the paper different techniques and possibilities for providing QoS via SDN are discussed. Chapter 2 presents viable approaches and methods in this area. Chapter 3 categorizes existing proposed solutions, while comparing their weak and strong points. In chapter 4, actual figures and results from the different frameworks are summarized and the overall viability of SDN as a QoS domain is assessed. In the last chapter, novel approaches and ideas for the future are proposed and concluding remarks are given.

2. APPROACHES TO PROVIDING QOS

There are two main categories of QoS techniques that need to be taken into consideration: **approaches proposed before the introduction of SDN** and **SDN enabled technologies**.

When considering **traditional QoS strategies**, without the involvement of SDN, two main types have been standardized. *Integrated services* [10] (short: IntServ) is a fine-grained, per flow traffic control architecture. The idea behind it is that every network element (router, switch) has to individually reserve resources for **each** flow. This is hard to accomplish in the current Internet [4]. First, routers have limited computational resources, that prohibits the classification of all possible application flows in the device itself. Second, this approach is not scalable, as every rooter on a flow's route needs to support Integrated Services and store all possible states and informations for the different flows. This is generally difficult to achieve (vendor lock-ins, limited memory resources). Therefore, this method is only applicable to small scale networks.

The second approach, *Differentiated services* [11] (short: DiffServ), is a coarse-grained traffic control architecture, relying on the 8-bit DS field (in place of the outdated TOS field) in the IP header. This field supports up to 64 different classes of traffic [15]. DiffServ routers then decide on per-hop basis how to forward packets based on their class. While this technique is applicable to bigger networks (since only a constant number of 64 classes need to be differentiated), it is static (because of the predefined number of classes) and lacks the ability to fine-tune the QoS of separate flows. For example, a use case with eight tenants having eight application types of traffic each easily reaches the limit of the DS field. This is not an unusual scenario for a single Autonomous System (AS). Furthermore, the specification of policies and the classification of traffic in DiffServ is done at the boundaries of DiffServ domains (Autonomous Systems) [4]. Because of this, there are no end-to-end guarantees across domains for the user, as the DS-classes can be interpreted differently in each one. To solve this, RFC 2638 [12] introduces a Bandwidth Broker for each domain, that has some knowledge on the policies in use. To ensure end-to-end QoS, bandwidth brokers need to communicate with each other across domains so that policies are interpreted properly. This is similar to the SDN enabled approaches, as it extracts the logic into a centralized agent. In fact, some of the proposed frameworks extend the idea of DiffServ by utilizing SDN [4]. However, provided the lack of centralized controllers in the current Internet architecture, DiffServ cannot be implemented globally, as there is no standardization of router reconfiguration protocols amongst different vendors. Each Broker relies on vendor-specific device reconfiguration, which imposes a restriction on the network.

In contrast to the described traditional methods, **SDN enabled approaches** tackle all of the problems described above. Through the higher level of abstraction provided by the decoupled controller, one can specify policies without the need to reconfigure low-level settings at each of the forwarding devices. The set of policies and also the different flow classes are unrestricted, allowing for fine-grained tuning based on the needs of the user (in contrast to the maximum of 64 predefined classes in the DS field). The rules can therefore be defined per flow (if necessary) and the controller has the task to apply them properly to the different network elements. In this regard, there are two main aspects to consider:

- providing QoS for a specific tenant or a business customer flow
- providing QoS for a specific application flow

A number of different SDN enabled solutions to those prob-

[1] In the sense of networking and SDN, a flow defines any given set of packets that share some common characteristics (e.g. all packets for a given HTTP connection)

doi: 10.2313/NET-2015-09-1_05

lems exist. The most common method, used by frameworks considered in this survey, is a form of virtual slicing of the available bandwidth. This can be classified as **resource reservation**, where each flow gets assigned a part of the overall transmission capacity. In contrast to that, **per-flow dynamic routing** has been proposed as a viable alternative that does not directly assign any of the resources to a flow. Moreover, approaches with specific focus on the actual **packet en-queuing** and **frameworks for policy enforcement** have been discussed as well. Many of the SDN enabled frameworks also require a form of sampling to constantly monitor the network [2] [8]. If all nodes are monitored, the gathered information can be extrapolated to the whole network [6], thus creating an overview of the network state. This approach creates overhead and needs to be applied in controlled manner.

In the following chapter, each of these aspects is covered in depth and example implementations are presented.

3. QOS ENABLED FRAMEWORKS

The following categories are the most prominent ways in which QoS can benefit from the concept of Software Defined Networking. For each category, first a short introduction to the approach is given. Then one or more frameworks are presented. Each framework's most notable features are explained and the results of available tests are discussed. It is important to notice that the described frameworks do not solely rely on the principles of the category they represent. The distribution among the different sections is based on the main focus of the specific solution.

3.1 Resource reservation frameworks

This is the most common solution for providing QoS, hence the large number of frameworks that fall into this category. Typically, frameworks of this type consist of two main modules: a flow classifier component and an SDN-based rate shaper. The classifiers read the packets' fields and attempt to assign a certain priority to the different flows based on the policies defined in the controller. The rate shapers, on the other hand, install resource reservation rules in an OpenFlow enabled switch based on this classification. The current OpenFlow implementations for the data plane elements (like Open vSwitch) don't explicitly support per-flow rate shaping, so most of the frameworks in this survey attempt to give some form of a universal solution for this problem.

An example for such a framework is **FlowQoS** [3]. This prototype targets small scale networks and utilizes SDN to reconfigure home routers according to a set of rate shaping policies defined by the user. The classification of the different flows is done by two separate modules. The first module performs initial classification of web traffic (HTTP and HTTPS), determined by the respective port numbers (80 and 443). Then further classification of this web traffic is done based on the DNS responses, matching the A and CNAME records against a list of predefined regular expressions. The second module handles the classification of other flows (non HTTP/HTTPS). They are classified based on the flow-tuple[2], the first four bytes sent and received and the sent and received payload sizes. After flows have been successfully classified, the rate shaping takes place. To tackle the lack of an OpenFlow per-flow rate shaping implementation, the framework introduces two "virtual" switches inside the home router. The different inner-switch connections between those two are then configured via Linux's **tc** utility and assigned different rates specified by the controller (predefined by the user). This is a simplified form of network slicing. For more information, please refer to [3]. This is a limited approach, as it only shows how QoS can be applied to a home router. Moreover, there is no data on executed tests and results, compared to the following frameworks. FlowQoS is a small, proof of concept framework that shows how SDN can be used to enable QoS and at the same time simplifies QoS configuration for single users. It can replace the need of manually using Linux's tc utility or iptables tagging, which could be complicated. The authors don't state any further use cases in their paper.

Moving from small to large-scale solutions, a more general framework is a part of the **EuQoS project** [4]. The focus of this solution is to enable QoS for *business customer* flows and make it possible to prioritize them on demand. The classification in this approach is based on the value of the DS field in combination with the destination IP of a packet, as only different tenant flows need to be distinguished. This is very similar to the standard approach applied in DiffServ, but also differentiates the tenants based on their IP address.

All routers are reconfigured by the controller. For each regular routing entry in a router two OpenFlow flow table entries[3] are created, which match the same destination IP as the routing entry. One of them matches packets with TOS disabled (low priority) and the other one matches packets with TOS enabled (high priority). By doing this, best-effort and high-priority business traffic can be differentiated. Therefore the number of flow entries is twice the number of regular routing entries. From a scalability perspective, this is acceptable.

For rate shaping, two queues with transmission rates are configured at each router. One matches the TOS enabled packages, the other one handles the rest. This makes the network suitable for differentiation of flows coming from different hosts. However, it does not cover the second type of differentiation - flows of different applications.

In addition to the regular resource reservation, EuQoS focuses on two more aspects:

- **Inter-AS QoS** - For each Autonomous System, one SDN controller is operational. It creates a flow entry at the edge router that sets the TOS field to enabled for a given high-priority flow coming from another AS. Since the only modification of packet fields happens at the border of an AS and the forwarding decisions are based on the TOS value, regular DiffServ forwarding can also take place inside the AS. That is why standard BGP (Border Gateway Protocol) or OSPF (Open Shortest

[2]source IP, source port, destination IP, destination port, transport layer

[3]A flow table entry matches a certain flow (based on predefined rules) and defines an action to be applied to that flow (e.g. forward, drop, change field)

doi: 10.2313/NET-2015-09-1_05

Path) routing can be used at this point. This provides a dynamic way to give priority to certain tenant flows across multiple AS, for example when an SLA between a business customer and a provider is taking place. EuQoS further allows the existing DiffServ brokers to utilize the SDN Northbound API.

- **Failure recovery** - each time a connection fails, the framework automatically ensures that the high-priority traffic does not suffer and is restored to the predefined transmission rate. This also means that if not enough network bandwidth is available (because of the link failure), some of the best-effort packets will not be forwarded immediately.

The framework was tested on the OFELIA [4] testbed, with 100 nodes for single and multiple AS. The results show that the flows with higher priority do take precedence, even in cases where the bandwidth is not enough for all the traffic [4]. The extend of this test exceeds that of any of the frameworks covered in this survey, but EuQoS remains a prototype.

Another SDN framework that utilizes similar rate shaping and classification approaches was presented at **Princeton University** [9]. The main difference is that it proposes an adaptive flow aggregator for the case when QoS needs to be provided per application flow. Services with similar needs are bundled together and handled as one flow in the forwarding devices, which improves scalability [9]. This way, the solution can handle **both** application flows and customer flows. The Princeton framework has a second main focus, which is the *convergence* of the whole network. For that purpose, active sampling of the network state is required. This introduces overhead, bound to put some strain on the network, which is virtually unavoidable with the given goal. The authors don't quantify how much stress is put on the controller, but this aspect should be taken into consideration.

The framework has been tested in a real world setting with 3 OpenFlow switches and 4 hosts, which is a rather small testbed. It shows better utilization of network resources due to the described monitoring. This framework, similarly to the ones preceding it, remains only a prototype.

3.2 Per-flow Routing Frameworks

This concept distinguishes multimedia flows from regular data flows via a classifier, similar to the resource reservation technique. However, instead of reserving resources (in the sense of bandwidth slices) at each forwarding device, the controller dynamically places the high priority flows on QoS guaranteed routes [2]. This enables dynamic routing, for which SDN is highly advantageous. At the same time, regular data flows remain on their usual routes. In this context, QoS routing is viewed as a Constraint Shortest Path (CSP) problem [2]. For example, one possible constraint is the delay for a multimedia stream. Since this problem is NP-hard, a heuristic is proposed in the OpenQoS framework. To ensure that the path is really optimal, **congestion** of the network is also taken into account. OpenQoS considers a link to be congested, if its utilization exceeds 70% of the possible bandwidth [2].

The main advantage of this approach to resource reservation (as in section 3.1) is minimized latency and packet loss for the non-QoS flows [2].

A representative framework of this type is OpenQoS. It guarantees service (application traffic) delivery on an optimal path and focuses mainly on multimedia flows, such as VoIP or video streaming. The framework itself is based on Floodlight, which is a universal Java controller.

For the generation of effective routes for the high-priority service flows, the controller needs an overview of the current network state (expressed in packet loss, delay, etc. for each link). The network state is collected from the simple forwarding elements by actively sending FEATURE_REQUEST messages for each feature of interest (doing active sampling). The performance of the framework depends on the accuracy of the gathered data, therefore the framework queries the elements each second [2]. This requirement has a negative effect on scalability. Gathering precise data for large networks is inefficient. The computation of the most efficient CSP per flow introduces further overhead for the controller. However, the paper of QoSFlow does not quantify the strain on the SDN controller that the monitoring causes. This aspect of the framework needs to be further investigated.

Another advantage of OpenQoS is its attempt to define flows only using fields from lower OSI levels, as "the packet parsing complexity is lower compared to [..] upper layers", according to [2]. That is why OpenQoS differentiates multimedia flows based on fields in MPLS (Multiprotocol Label Switching), which is generally placed between the data link and network OSI layers.

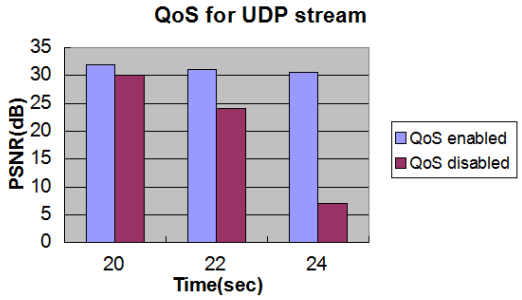

Figure 2: Three samples of PSNR in time, representing the results of an OpenQoS test [2]

The framework has been tested on a network with 3 Pronto 3290 switches, which is not sufficient to overrule the scalability concerns, and still remains a prototype. One of the executed tests compares the performance of two video UDP streams of the same content, with and without enabled QoS. In the period from second 20 to second 24 of the transmission, there is a significant difference in the PSNR (Peak Signal-to-Noise Ratio) of the two streams. The metric quantifies the quality of the received signal by measuring the power of the corrupting noise (loss) during transmission. Accordingly, higher values mean a better service quality. As

Figure 2 demonstrates, the UDP stream with enabled QoS is preserved better during the specified time frame. This in turn shows the effectiveness of the framework when services need to be prioritized.

3.3 Frameworks with Focus on Queue Management and Package Scheduling

The topic of packet scheduling and enqueuing is also important. OpenFlow version 1.3 specifies a standard FIFO queuing of all incoming packets [13]. This could prevent certain flows (e.g. multimedia streams) from meeting the QoS requirements if the network segment is congested. To overcome this limitation, a framework called **QoSFlow** is proposed, which enables the reordering of packets in a given queue. It attempts to introduce the traffic control capabilities of Linux into OpenFlow networks. In this context, the framework implements a couple of different packet scheduling mechanisms [13], the most notable of which is SFQ (Stochastic Fairness Queuing). This algorithms aims at giving each high-priority flow a fair chance to be forwarded, contrasting to standard FIFO queuing. Each flow is assigned to a bucket. Afterwards, in a round-robin manner, flow packets are popped one by one from the respective buckets in an effort to ensure fairness.

However, there are some restricting factors as well. The framework is limited to 8 queues per switch port. This limitation exists because of the used slicing mechanism specified in OpenFlow 1.0. While it is not caused by the framework itself, this fact still needs to be taken into account. Another disadvantage is that it requires that the switches run a Linux distribution instead of a vendor-specific firmware. Therefore this solution is not directly applicable to the Internet in its current state.

QoSFlow has been tested with up to 3 TPLink 1043ND commercial switches [13], which is a small test network. In one of the tests on the performance of the SFQ algorithm, the QoE (Quality of Experience) is again quantified by the Peak Signal-to-Noise Ratio (PSNR), similarly to the OpenQoS case. For two different video flows the behavior of SFQ and standard FIFO scheduling is compared. SFQ reaches a higher PSNR value (which is better) in both cases. According to the QoSFlow paper, the difference between the two approaches is 48.57 % in the first case and 68.57 % in the second case, in favor of SFQ against FIFO [13], meaning that the improved scheduling does lead to better video quality, as far as the user perception is concerned.

3.4 Frameworks for Policy Enforcement

The major problem with predefined SLAs, in regard to QoS, is that with the current Internet architecture there is not a single standardized and flexible way to apply those to the network. Most of the newest technologies that are applied to achieve those SLAs are also proprietary. Therefore the need for flexible and scalable management tasks emerges.

Via the Northbound API of an SDN controller, it is possible to dynamically define SLAs and then enforce them on the underlying forwarding plane through the southbound API. One of the proposed SDN-enabled solutions is PolicyCop, based on the Floodlight controller. It differs from other existing DiffServ approaches [8] as it is not restricted by static traffic classes and the coarse-grained granularity that comes with them. The framework is easily extensible, as it has a layered architecture. Each layer communicates with the others via RESTful JSON APIs, hence the possibility to easily add new layers or exchange old ones. It further allows to dynamically create new flow classes, in contrast to the static classes in DiffServ [8]. The solution is flexible, as it does not require any special resources, other than OpenFlow-enabled switches.

The policy management itself is done by 2 components: a Policy Validator and a Policy Enforcer. The Policy Validator monitors the network to ensure that policies are being applied properly. To make this possible, information per flow and per flow table needs to be actively gathered by the controller. This is achieved by inserting packet probes in the network.[8] Meanwhile, passive statistics are also gathered from packets redirected to the controller. This results in the monitoring of metrics describing the network state, like bandwidth usage, residual capacity, number of dropped packets, latency, error-rate and jitter [8]. For more information on the monitoring framework that PolicyCop uses, please refer to [16]. Similar to OpenQoS, this could lead to a serious overhead and puts the scalability of the framework in question. As a result of the inquired network state, the Policy Enforcer places new rules in the forwarding elements, to account for possible changes or deviations from the defined policies.

The network has been tested in an environment of 5 Open vSwitches and 4 hosts. PolicyCop successfully manages to restore broken policies and the corresponding flows' throughput in a scenario with 4 different services running simultaneously [8]. However, the provided testbed is too small and can only be seen as a "Proof of Concept".

3.5 Detailed comparison

From all presented frameworks, only **EuQoS** has been tested for a larger scale network. Although the tests of all other frameworks manage to show their capabilities, they can be considered insufficient because of the small test-bed sizes. In contrast to the others, **EuQoS** also directly supports standard DiffServ approaches. Nevertheless, that restricts the differentiation of flows to single hosts (high priority business customers), differentiation based on services and applications is not possible. The framework proposed by **Princeton** handles this problem well, aggregating similar services to a single flow in addition to the per-tenant flow classification. However, this system has been tested in a much smaller testbed than the EuQoS framework. Different from those two more general approaches, **OpenQoS** focuses on multimedia traffic only, using per-flow routing instead of resource reservation. However, while this technique can arguably be more efficient path-wise, it introduces an overhead due to active sampling of the system state, which is not necessarily required in a resource reservation framework. **QoSFlow**, on the other hand, is an example of a support framework that defines additional improvements of QoS via SDN methods, since it is concerned only with the actual order of the queues during the packet scheduling. Further investigation of the possible integration of this framework with the others is encouraged. The major problem in this case is that

all forwarding devices are required to run Linux, which is currently not realistic in a normal network setting. Lastly, **PolicyCop** focuses on the actual policy management, giving administrators full control to efficiently change the way traffic is handled. All of the covered frameworks are prototype solutions.

A detailed summary of all the relevant information can be found in *Apendix 1.*

4. VIABILITY OF SDN AS A QOS DOMAIN

Considering the data from the different framework tests, it is clear that there are advantages when using SDN controllers to provide QoS.

The **EuQoS** framework shows that it can transmit all high priority traffic without any loss in a medium data rate scenario (2.4 Mb/s to 7 Mb/s traffic per host) with a 30% to 70% ratio of high priority to best-effort traffic in the network [4]. **QoSFlow** suggests a possible gain in QoE of up to 48.7% when implemented properly [13]. As seen in section 3.4, **PolicyCop** can maintain and enforce policies once again when they have been broken.

However, the results of those frameworks (with the exception of EuQoS) were obtained in small networks, consisting of not more than 5 forwarding devices. Therefore, more thorough investigation is needed.

The configuration of traffic queues is another important aspect of QoS. Protocols such as OF-Config and OVSDB deal with the issue of configuration of resources (like queues), but are not properly supported by the mainstream OF controllers. A standalone solution for this problem called **QueuePusher** has been proposed, as an extension to the controller Floodlight. It is based on the OVSDB protocol and describes the mechanisms of building priority queues for the different QoS enabled flows [7].

This is an example of how some important aspects of SDN are still not available, despite their specification in the OpenFlow protocol.

Overall, the fact that a lot of the mentioned frameworks need to extend a controller or implement a missing protocol feature, combined with all of them being prototypes, shows that the concept of SDN still needs to be better accommodated. Nevertheless, all of the frameworks show significant gains in different aspects of QoS which cannot be neglected. While it is true that SDN introduces a single point of error [6] via the centralization of the controller, distributed solutions have been proposed [4], mitigating this concern.

5. RELATED WORKS

A comprehensive survey of the different aspects of SDN has already been conducted [1]. The work covers all major aspects of the SDN domain, also specifying a list of traffic engineering solutions with regard to QoS. However, a comparison or categorization of the SDN enabled QoS frameworks is not present. Also, no advantages and disadvantages are discussed and no specific implementation details or test figures are covered.

6. FUTURE OUTLOOK. CONCLUSION

The proposed QoS frameworks all benefit from the idea behind SDN. Since they are still prototypes, most of them are continuously developed further. **OpenQoS** proposes a future application of "per-flow routing" in the area of Multiple Description Coding (MDC). A source stream is encoded into independent bitstreams that can be decoded separately. There is a requirement that every bit stream should take a different path. This is achievable via the dynamic routing possibilities of OpenQoS.

Regarding the whole Internet, the Control Exchange Points (CXP) [14] has been proposed as an architectural model that is supposed to improve end-to-end QoS across different domains. This suggests a new type of business relationship between ISPs, where each provider defines a "partial path" for a flow and the CXP component combines all of them together.

In this paper different types of SDN enabled QoS solutions have been evaluated. While small scale frameworks are beneficial to the individual user, larger scale implementations represent ideas applicable to the whole Internet architecture. Despite the fact that Software Defined Networking is still a relatively new approach, the detailed categorization and comparison have revealed important positive effects in regard to QoS , achievable only through a decoupled and easily manageable control plane.

7. REFERENCES

[1] Diego Kreutz, Fernando Ramos, Christian Rothenberg, Siamak Azodolmolky, Steve Uhlig: *Software Defined Networking: A Comprehensive Survey*, page 25, Proceedings of the IEEE, Volume 103, Jan, 2015

[2] Hilmi Egilmez, S. Dane, K. Bagci, A. Tekalp: *OpenQoS: OpenFlow controller design for multimedia delivery with end-to- end Quality of Service over Software-Defined Networks*, APSIPA ASC, Hollywood, CA, 3-6 Dec, 2012

[3] M. Seddiki, Muhammad Shahbaz: *FlowQoS: QoS for the Rest of Us*, HotSDN?14, Chicago, 22 Aug, 2014

[4] S. Sharma, D. Staessens, D. Colle, D. Palma: *Implementing Quality of Service for the Software Defined Networking Enabled Future Internet*, EWSDN, Budapest, 1-3 Sep, 2014

[5] G. Araniti, J. Cosmas, A. Iera, A. Molinaro: *OpenFlow over Wireless Networks: Perfomance Analysis* BMSB, Beijing, 25-27 Jun, 2014

[6] Daniel Raumer, Lukas Schaighofer, Georg Carle: *MonSamp: A distributed SDN application for QoS monitoring* FedCSIS, Warsaw, 7-10 Sep, 2014

[7] David Palma et al., *The QueuePusher: Enabling Queue Management in OpenFlow*, EWSDN, Budapest, 1-3 Sep, 2014

[8] M. F. Bari et al., *PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks*, SDN4FNS, Trento, 11-13 Nov, 2013

[9] Wonho Kim et al. *Automated and Scalable QoS Control for Network Convergence*, INM/WREN, San Jose, CA, 27 Apr, 2010

[10] R. Braden et al., *Integrated Services in the Internet*

Architecture: An Overview, Internet Engineering Task Force, RFC 1633, Jun, 1994

[11] S. Blake et al., *An Architecture for Differentiated Services*, Internet Engineering Task Force, RFC 2475, Dec, 1998

[12] K. Nicolas et al., *A Two-bit Differentiated Services Architecture for the Internet*, Internet Engineering Task Force, RFC 2638, Jul, 1999

[13] A. Ishimori et al. *Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking*, EWSDN, Berlin, 10-11 Oct, 2013

[14] V. Kotronis et al. *Control Exchange Points: Providing QoS-enabled End-to-End Services via SDN-based Inter-domain Routing Orchestration*, ONS, Santa Clara, CA, 3-5 Mar, 2014

[15] K. Nichols et al. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, Network Working Group, RFC 2474, Dec, 1998

[16] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba *PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks*, NOMS, Krakow, 5-9 May, 2014

doi: 10.2313/NET-2015-09-1_05

	Framework	Scale	Type	Testbed	Flexibility	Guarantees
Resource reservation	FlowQoS	Small scale (home routers)	Prototype	-	Restricted to home routers	-
	EuQoS	Medium to large-scale (possible across multiple AS)	Prototype	OFELIA testbed, 100 nodes	Compatible with BGP and OSFP and therefore to the current Internet	Flow of prioritized tenant has precedence, failure recovery
	Princeton framework	-	Prototype	3 ProCurve 5406zl switches	-	Network wide optimization & convergence
Per-flow path optimization	OpenQoS	Medium scale (supposedly, since the network needs to be monitored)	Prototype	3 Pronto 3290 switches	Scalability problems	Multimedia traffic with enabled QoS performes better
Queuing and scheduling	QueryPusher	Any	Prototype	-	Floodlight extension	-
	QoSFlow	Any	Prototype	Up to three TPLink 1043ND	Requires Linux installed at each router; limited to 8 queues per switch port	Improvement in QoE for QoS enabled traffic (up to 48 %)
Policy Enforcement	PolicyCop	Medium scale (supposedly, since the network needs to be monitored)	Prototype	5 switches running Open vSwitch	Compatible with all switches with OpenFlow support, easily extensible due its layered architecture	Enforcement of policies

Table 1: Apendix 1

doi: 10.2313/NET-2015-09-1_05