OpenQoS: An OpenFlow Controller Design for Multimedia Delivery with End-to-End Quality of Service over Software-Defined Networks

Hilmi E. Egilmez*, S. Tahsin Dane*, K. Tolga Bagci* and A. Murat Tekalp*
* Koc University, Istanbul, Turkey
E-mail: {hegilmez, sdane, kbagci, mtekalp}@ku.edu.tr

***Abstract*—OpenFlow is a Software Defined Networking (SDN) paradigm that decouples control and data forwarding layers of routing. In this paper, we propose OpenQoS, which is a novel OpenFlow controller design for multimedia delivery with end-to-end Quality of Service (QoS) support. Our approach is based on QoS routing where the routes of multimedia traffic are optimized dynamically to fulfill the required QoS. We measure performance of OpenQoS over a real test network and compare it with the performance of the current state-of-the-art, HTTP-based multi-bitrate adaptive streaming. Our experimental results show that OpenQoS can guarantee seamless video delivery with little or no video artifacts experienced by the end-users. Moreover, unlike current QoS architectures, in OpenQoS the guaranteed service is handled without having adverse effects on other types of traffic in the network.**

I. Introduction

The Internet design is based on end-to-end arguments [1] where the network support is minimized and the end hosts are responsible for most of the communication tasks. This design has two main advantages: Firstly, it allows a unified best-effort service for any type of data at the network layer where service definitions are made at the upper layers (hosts). Secondly, it reduces the overhead and the cost at the network layer without losing reliability and robustness. This type of architecture fits perfectly to data transmission where the primary requirement is reliability. Yet, in multimedia transmission, timely delivery is preferred over reliability. Multimedia streaming applications have stringent delay requirements which cannot be guaranteed in the best-effort Internet. So, it is desirable that the network infrastructure supports some means to provide Quality of Service (QoS) for multimedia traffic. To this effect, the Internet Engineering Task Force (IETF) has explored several QoS architectures, but none has been truly successful and globally implemented. This is because QoS architectures such as IntServ [2] and Diffserv are built on top of the current Internet's completely distributed hop-by-hop routing architecture, lacking a broader picture of overall network resources. Even though MPLS [3] provides a partial solution via its ultra-fast switching capability, it lacks real-time reconfigurability and adaptivity.

Software Defined Networking (SDN) [4] is a paradigm shift in network architecture where the network control is decoupled from forwarding and is directly programmable. This migration of control provides an abstraction of the underlying network for the applications residing on upper layers, enabling them to treat the network as a logical or virtual entity [5].

Among several attempts, OpenFlow is the first successful implementation [6] of SDN which has recently started being deployed throughout the world and has already attracted many commercial vendors [7]. As proposed in SDN, OpenFlow moves the network control to a central unit called controller; while the forwarding function remains within the routers called forwarders (see Fig.1). The OpenFlow controller is the brain of the network where packet forwarding decisions are made on per-flow basis and the network devices are configured accordingly via the OpenFlow protocol, which defines the communication between the controller and the underlying devices. OpenFlow provides complete network visibility, resource monitoring, and network virtualization, allowing sophisticated network management solutions. In this paper, we address a specific networking problem which is providing QoS for multimedia delivery, and present an OpenFlow based solution for it.

This paper proposes OpenQoS, a novel controller design that enables QoS for multimedia delivery over OpenFlow networks. In order to support QoS, we group the incoming traffic as data flows and multimedia flows, where the multimedia flows are dynamically placed on QoS guaranteed routes and the data flows remain on their traditional shortest-path. Our approach is different from current QoS architectures, since we employ dynamic routing which is now possible with OpenQoS. OpenQoS is based on our prior works in [8], [9], [10], where the optimization framework and the results presented in those papers are exploited. Here, we also demonstrate the performance of OpenQoS on a real network with commercial OpenFlow-enabled switches.

The rest of the paper is organized as follows: Section II presents the OpenQoS design that supports QoS over OpenFlow networks. The OpenFlow test network and the OpenQoS implementation are discussed in Section III. Section IV presents the results showing the performance of OpenQoS over the test network. Future directions and application areas of the OpenQoS are proposed in Section V. Concluding remarks are given in Section VI.

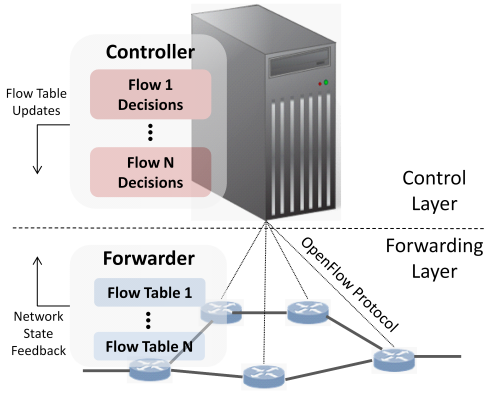


Fig. 1. OpenFlow Architecture

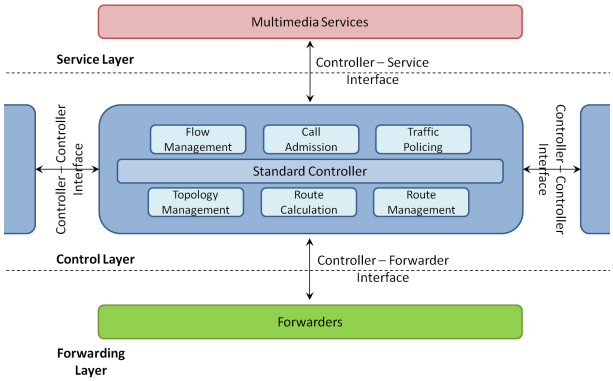


Fig. 2. OpenQoS Controller Design

II. OpenQoS: Controller Design

In this section we introduce OpenQoS. We first discuss its architecture, and then we present the optimization framework for dynamic QoS routing. The organization of this section is as follows: In Section II-A, we present the proposed QoS architecture and OpenQoS running on top. Section II-B discusses the routing mechanism in OpenQoS. The optimization framework for QoS routing is given in Section II-C.

A. QoS Architectures and OpenQoS Design

There is a continuing debate on how to evolve the Internet in order to provide QoS for multimedia traffic. Currently, there is no QoS architecture that is successful and globally implemented. Some researchers argue that fundamental changes should be done to fully guarantee QoS, while others think slight changes are enough to have soft guarantees which will provide the requested QoS with high probability. So, we can group the QoS architectures into two major categories:

- **Integrated Services (IntServ) architectures** provide hard QoS guarantees via resource reservation (bandwidth, buffer) techniques. The mechanism is similar to circuit switched networks (e.g. ATM), and data transmission starts after an end-to-end connection is established. The major problem of IntServ based architectures is that they require fundamental changes in the network core.
- **Differentiated Services (DiffServ) architectures** provide soft QoS guarantees via scheduling (priority queuing). Unlike IntServ, DiffServ requires changes in the edge of the network. Edge routers should have packet classification functionality, and core routers should forward the packets based on their priorities.

In terms of routing, DiffServ applies the same routing mechanism as the Internet does. On the other hand, IntServ uses the Resource Reservation Protocol (RSVP) [11] which inter-operates with any routing protocol to reserve resources along the calculated path. For each connection, QoS routing is performed only once for connection establishment and that connection remains until teardown. In IntServ, applying dynamic QoS routing is hard, since it introduces latency due to reconnection establishment.

With OpenQoS, we propose a new prioritization scheme which is based on routing. In order to fulfill the required end-to-end QoS, we propose dynamic QoS routing for QoS flows (multimedia traffic) while other flows (data) remain on their shortest path. Our approach is different from the current QoS architectures since we use neither resource reservation nor priority queuing (i.e. rate shaping). The main advantage of not using these methods is that the adverse effects of QoS provisioning on non-QoS flows, such as packet loss and latency, are minimized. Complete comparison between OpenQoS and the current QoS architectures is presented in Table I.

OpenQoS is an extension of the standard OpenFlow controller which provides multimedia delivery with QoS. As depicted in Fig.2, OpenQoS offers various interfaces and functions to enable QoS. The main interfaces of the controller design are:

- *Controller - Forwarder Interface*: The controller attaches to forwarders with a secure channel using the OpenFlow protocol to share necessary information. The controller is responsible for sending flow tables associated with data flows, requesting network state information from forwarders for discovering the network topology, and monitoring the network.
- *Controller - Controller Interface*: The single controller architecture does not scale well when the network is large. As the number of the OpenFlow nodes increases, multiple controllers are required. This interface allows controllers to share the necessary information to cooperatively manage the whole network.
- *Controller - Service Interface*: The controller provides an open, secure interface for service providers to set flow definitions for new data partitions and even to define new routing rules associated with these partitions. It also provides a real-time interface to signal the controller when a new application starts a data flow.

The controller should also manage several key functions:

- *Topology management*: This function is responsible for discovering and maintaining network connectivity through data received from forwarders.

TABLE I
COMPARISON OF QOS ARCHITECTURES

	Flow Support	Type of Guarantee	Complexity	Effects on other flows	Mechanism
IntServ	Individual flows	Hard & end-to-end	High	High (due to reservation)	Resource reservation
DiffServ	Aggregated flows	Soft & hop-by-hop	Medium	Medium (priority queuing)	Scheduling, priority queuing
OpenQoS	Multiple flows	Soft & end-to-end	Low	Low (only based on routing)	Dynamic QoS routing

- *Route management*: This function is responsible for determining the availability and packet forwarding performance of routers to aid the route calculation. It requires collecting the up-to-date network state from the forwarders on a synchronous or asynchronous basis.
- *Flow management*: This function is responsible for collecting the flow definitions received from the service provider through the controller-service interface, and efficient flow management by aggregation.
- *Route calculation*: This function is responsible for calculating and determining routes for different types of flows. Several routing algorithms can run in parallel to meet the performance requirements and the objectives of different flows. Network topology and route management information are input to this function along with the service reservations.
- *Call admission*: This function denies/blocks a request when the requested QoS parameters cannot be satisfied (i.e. there is no feasible route), and informs the controller to take necessary actions.
- *Traffic policing*: This function is responsible for determining whether data flows agree with their requested QoS parameters, and applying the policy rules when they do not (e.g. pre-empting traffic or selective packet dropping).

B. Per-Flow Routing in OpenQoS

The current Internet does not allow routing on per-flow basis. When a packet arrives at a router, it checks the packet's source and destination address pair with the entries of the routing table, and forwards it according to predefined rules (e.g. routing protocol) configured by the network operator. On the other hand, OpenFlow provides the flexibility of defining different types of flows to which a set of actions and rules can be associated. For example, one type of flow may be forwarded using a shortest path routing algorithm while the other flows may follow manually configured routes over the network. So, each flow (i.e. packet) can be treated differently at the networking layer.

In OpenFlow, we can define flows in many ways. Flows can contain same or different types of packets. For example, packets with the TCP port number 80 (reserved for HTTP) can be a flow definition, or packets having RTP header may indicate a flow which carries voice, video or both. In essence, it is possible to set flows as a combination of header fields as illustrated in Fig.3, but the network operator should also take the processing power limitations of the network devices

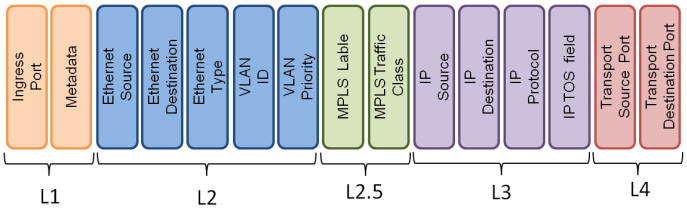


Fig. 3. Flow identification fields in OpenFlow

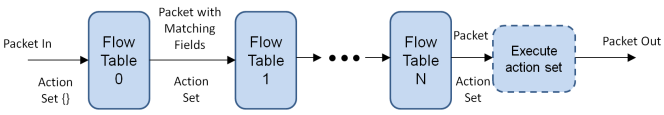


Fig. 4. Flow tables and their pipelined processing

(routers or switches) into account. In order to avoid complex flow table lookups, flow definitions should be cleverly set and aggregated if possible. In OpenFlow, network devices store the flows and their associated rules in flow tables which are processed as a pipeline shown in Fig.4. The goal of the pipelined processing is to reduce the packet processing time.

OpenQoS exploits OpenFlow's flow-based forwarding paradigm so that we can differentiate data and multimedia traffic. Multimedia flows may be determined by using the following packet header fields or values:

- Traffic class header field in MPLS,
- TOS (Type of Service) field of IPv4,
- Traffic class field in IPv6,
- If multimedia server is known, source IP address,
- Transport source and/or destination port numbers.

It is desirable to define flows according to lower layer (L2, L3) packet headers since the packet parsing complexity is lower compared to processing up to upper layers (L4). Therefore, we propose to define multimedia flows using fields in MPLS which is considered in between data link and network layer (L2.5), and provides ultra-fast switching capability. But, in some cases upper layer header fields may also be required for better packet type discrimination, and OpenFlow allows the flexibility of defining flows using upper layer (L4) header fields. Besides, flow definitions may not rely on current IP. Any addressing scheme with service level information can be used to define multimedia type flows.

In order to calculate the QoS routes, it is essential to collect up-to-date global network state information, such as delay,

bandwidth, and packet loss rate for each link. The performance of any routing algorithm is directly related to how precise the network state information is. Over large networks, collecting the network state globally may be challenging due to the scale of the network. The problem becomes even more difficult in the Internet because of its completely distributed (hop-by-hop) architecture. OpenFlow eases this task by employing a centralized controller. As illustrated in Fig.1, instead of sharing the state information with all other routers, OpenFlow forwarders directly send their local state information to the controller. Then, the controller collects the forwarders' state information and computes the best feasible routes accordingly.

C. Optimization of Dynamic QoS Routing

We pose the dynamic QoS routing as a Constrained Shortest Path (CSP) problem. It is crucial to select a cost metric and constraints where they both characterize the network conditions and support the QoS requirements. In multimedia applications, the typical QoS indicators are packet loss, delay and delay variation (jitter). However, some QoS indicators may differ depending on the type of the application, such as:

- *Interactive multimedia applications* that have strict end-to-end delay requirements (e.g. 150-200 ms for video conferencing). So, the CSP problem constraint should be based on the total delay.
- *Video streaming applications* that require steady network conditions for continuous video playout; however, the initial start-up delay may vary from user to user. This implies that the delay variation is required to be bounded, so the CSP problem constraint should be based on the delay variation.

In our formulation, a network is represented as a directed simple graph $G(N, A)$, where N is the set of nodes and A is the set of all arcs (also called links), so that arc (i, j) is an ordered pair, which is outgoing from node i and incoming to node j. Let R_{st} be the set of all routes (subsets of A) from source node s to destination node t. For any route $r \in R_{st}$ we define cost f_C and delay f_D measures as,

$$f_C(r) = \sum_{(i,j)\in r} c_{ij} \tag{1}$$

$$f_D(r) = \sum_{(i,j)\in r} d_{ij} \tag{2}$$

where c_{ij} and d_{ij} are cost and delay coefficients for the arc (i, j), respectively. The CSP problem can then be formally stated as finding

$$r^* = \arg\min_r \{f_C(r) \mid r \in R_{st}, f_D(r) \leq D_{max}\} \tag{3}$$

that is, finding a route r which minimizes the cost function $f_C(r)$ subject to the delay variation $f_D(r)$ to be less than or equal to a specified value D_{max}. We select the cost metric as follows,

$$c_{ij} = g_{ij} + d_{ij} \ \forall (i, j) \in A \tag{4}$$

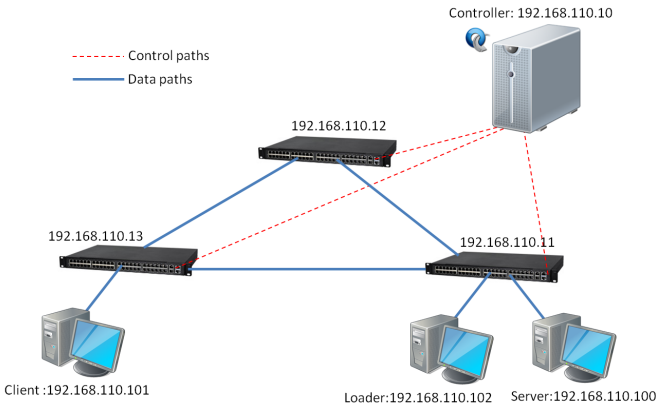


Fig. 5. OpenFlow Test Network

where g_{ij} denotes the congestion measure for the traffic on link (i, j) and d_{ij} is the delay measure. OpenQoS collects necessary parameters g_{ij} and d_{ij} using the *route management* function.

The CSP problem stated in (3) is known to be NP-complete, so there are heuristic and approximation algorithms in the literature. For the *route calculation* function of OpenQoS, we propose to use the Lagrangian Relaxation Based Aggregated Cost (LARAC) algorithm which is a polynomial-time algorithm that efficiently finds a good route without deviating from the optimal solution in $O([n+mlogm]^2)$ time [12]. When the *route management* function updates the QoS indicating parameters or the *topology management* function detects a topology change, the *route calculation* function runs the LARAC algorithm to solve the CSP problem of (3). Then, the controller updates the forwarders' flow tables accordingly. Hence, the QoS routes are dynamically set.

III. OpenFlow Test Network and OpenQoS Implementation

A. Test Network

We deployed the OpenFlow test network composed of three OpenFlow enabled Pronto 3290 switches, one controller and 3 host computers. As shown in Fig.5, the switches are connected in a triangular shape to have path diversity. The video streaming server and the client are connected to different switches, while the traffic loader inserting cross-traffic into the network is connected to the same switch that the server connects. Each switch initiates a secure connection to the controller using the OpenFlow protocol (see dashed lines in Fig.5). The controller runs our OpenQoS implementation which is described in detail in Section III-B.

B. Implementation of OpenQoS

We implement OpenQoS over a standard OpenFlow controller, Floodlight [13]. There are also several standard controller alternatives such as NOX [14], Beacon [15], Maestro [16] to implement OpenQoS, but currently Floodlight is the most stable controller. Floodlight is an open source controller

written in Java. It provides a modular programming environment so that we can easily add new modules on top and decide which existing modules to be run.

In our implementation of OpenQoS, we add two major modules to enable *route management* and *route calculation* functions discussed in Section II-A. The *topology management* function has already been implemented in Floodlight and we directly used that module in OpenQoS. These functions are essential building blocks of our controller design which makes dynamic QoS routing possible. Yet, the OpenQoS implementation is still incomplete. Firstly, *controller-to-controller*, *controller-to-service* interfaces must be defined, and then the functions using these interfaces (*flow management, call admission, traffic policing*) must be implemented. Since we concentrate on QoS in this paper, we left these open issues as future works.

1) Route Management: The route management module provides one of the key functions in the OpenQoS controller. It collects the up-to-date network state information such as link speed, available bandwidth and packet drop counts from the forwarders. The controller requests various statistics from forwarders by sending `FEATURE_REQUEST` messages, and in return forwarders send `FEATURE_REPLY` messages containing requested statistics. These messaging mechanisms are described in detail in OpenFlow specification v1.0 [17].

In order to support dynamic QoS, it is essential to keep the network state information up-to-date. The performance of the route calculation depends on the accuracy of the collected data. So, OpenQoS controller periodically collects available bandwidth for each link. The period is set to 1s since it has been shown that the Internet traffic behaves like independent Poisson distribution in sub-second time scales [18].

After receiving the available bandwidth measures from the forwarders, the route management module

- detects whether there is a congestion event in any of the links.
- determines link cost parameters to be used in the optimization problem stated in (3).

Each link can be in two states: congested or non-congested. In practice, a link is assumed to be congested if the utilization of that link exceeds 75% - 85%. We consider that a link is congested if that link is 70% bandwidth utilized. The link costs are determined by using the exact same formula in (4), where the congestion measure is found as,

$$g_{ij} = \begin{cases} \frac{T_{ij} - 0.7 \times B_{ij}}{T_{ij}}, & 0.7 \times B_{ij} < T_{ij} \\ 0, & 0.7 \times B_{ij} \geq T_{ij} \end{cases} \quad (5)$$

where T_{ij} is the total measured traffic amount in bps and B_{ij} is the max achievable bandwidth in bps on link (i, j). Note that, in (5), the *non-congested* links have 0 congestion measure value. The delay parameter d_{ij} in (4) is set to 1 which simply corresponds to hop-count. This is because the current OpenFlow switch implementations do not have any support on collecting delay related statistics (total delay, jitter).

In order to add an event based dynamicity to the QoS routing, the *route manager* signals forwarders when QoS routes need to be rerouted. This signalling can be achieved by deleting a specific flow entry. After a QoS flow entry is deleted, the forwarders cannot match newly coming packets, therefore they ask the controller to define new flow entries which causes multimedia packets to be rerouted. The flow deletion is triggered in two cases: (1) If a previously *non-congested* link is now *congested*, we delete the flow entries matching multimedia (QoS) packets in the flow tables of the forwarders. (2) If a previously *congested* link is *non-congested* in the last 3 periods, we again delete the flows accordingly. We require 3 periods of *non-congested* state to ensure there are no fluctuations in the traffic rate on the links.

2) Route Calculation: In Floodlight, route calculation is done when a `PACKET_IN` message arrives to the controller. It calculates the shortest path route and pushes flow definitions to the switches along that path accordingly. On the other hand, OpenQoS first checks if it is a multimedia packet or not, based on pre-defined flow setups described in Section II-B. Then, the *route calculation* module calculates two paths between the source and destination pair of the incoming packets. One path is the QoS optimized path and the other is the shortest path. Note that the QoS routes are calculated using the LARAC algorithm as described in Section II-C. Currently, we only detect multimedia packets but it can be easily modified to add new routing policies to new type of services.

IV. Results

To demonstrate the performance of our OpenQoS implementation, we built a video streaming environment over a real OpenFlow test network shown in Fig.5. Throughout the tests, we used a well-known test sequence "*in to tree*" having 500 frames with the resolution of 1280×720. We looped the raw video sequence reversely once to have 1000 frames lasting about 40s. We then encoded the looped sequence in H.264 format using the *ffmpeg* encoder (v.0.7.3) [19] at three different average bit-rates to have

- *Stream 1* at 1800 kbps (32.55dB),
- *Stream 2* at 900 kbps (30.57dB),
- *Stream 3* at 450 kbps (28.75dB).

These three H.264 video streams are used in two test scenarios presented in the next subsections.

A. Streaming over UDP

We created a scenario where two copies of the *Stream 1* are sent from the server residing at 192.168.110.100 to the client with the IP address 192.168.110.101 (see Fig.5). The server uses VLC media player [20] to stream videos using RTP/UDP. One copy of the video is sent to the destination port 5004 while the other copy is sent to port 5005. To show the performance difference in terms of QoS, we matched the multimedia flows (QoS flows) to the transport port number, 5004. Thus, the video packets destined to port 5004 are identified as being part of a multimedia flow by the OpenQoS controller and routed accordingly, while the other video (destined to port

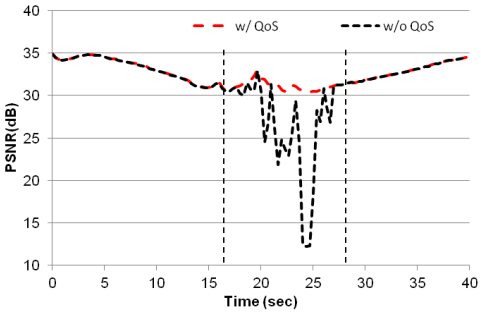


Fig. 6. Best case result of UDP streaming

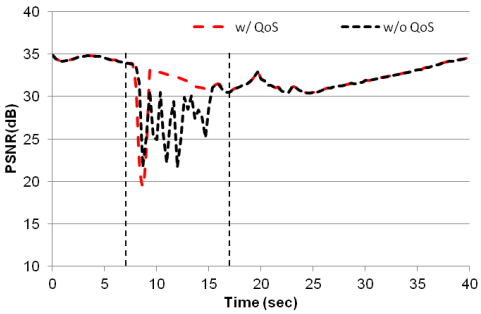


Fig. 7. One case result of UDP streaming

5005) is considered as a data flow which has no QoS support (i.e. best-effort). In each test, 10 second long cross-traffic is sent from the loader (192.168.110.102) to the client once at a random time. The client runs two VLC player sessions, listening RTP/UDP packets at ports 5004 and 5005, to save the received videos. We expect to see distortions in the video received on port 5005 during the cross-traffic while the other video received on port 5004 will be rerouted and affected little or not at all in terms of video quality.

We decode the received videos using *ffmpeg* and measure their qualities using the peak signal to noise ratio (PSNR) values with respect to the original raw video. The results are given in Figs.6 and 7 which are in terms of received video quality (PSNR) versus time. The vertical dashed lines mark the start and end times of the cross-traffic.

The best case result is shown in Fig.6 where the video with QoS support (w/ QoS) is not affected from the cross traffic and approaches full video quality, while the video without QoS support (w/o QoS) has significant amount of quality loss. However, in Fig.7, the video with QoS also suffers, it is recovered in less than 1s. After repeating the scenario 20 times, we observed that the average loss recovery period is 0.76s. Most of the time the user watching the video is not disturbed from the quality loss in such a small interval even if we use UDP which does not guarantee reliable delivery at all.

B. HTTP-based Adaptive Streaming

We built a test scenario similar to the one discussed in Section IV-A where TCP is employed as a transport protocol

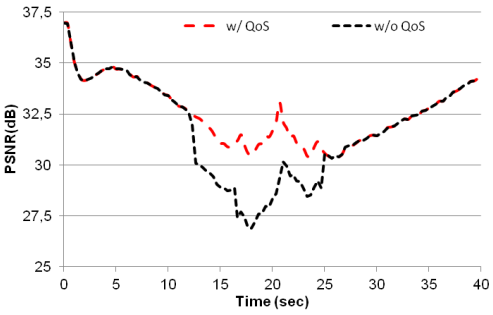


Fig. 8. Adaptive HTTP streaming result

instead of UDP. The server sends

- *Stream 1* with QoS support,
- a video without QoS support chosen adaptively among *Stream 1, 2* and *3*.

For adaptive video streaming we used *Adobe Flash Media Server 4.5* [21]. At the server side, each video stream (*Stream 1, 2* and *3*) is fragmented into 4 second long sub-streams and an associated *m3u8* playlist is created. At the client side, the VLC player first downloads the *m3u8* playlist and then selects an appropriate sub-stream rate-adaptively. While the loader (see Fig.5) applies 10 second cross-traffic, the video with QoS (i.e. *Stream 1*) is rerouted, and the video without QoS is rate adapted. Fig.8 illustrates the quality difference between the rate adaptation (w/o QoS) and the QoS rerouting (w/ QoS) of a sample test. We repeat the same test scenario over 30 times, and we do not observe any quality loss in the video with QoS.

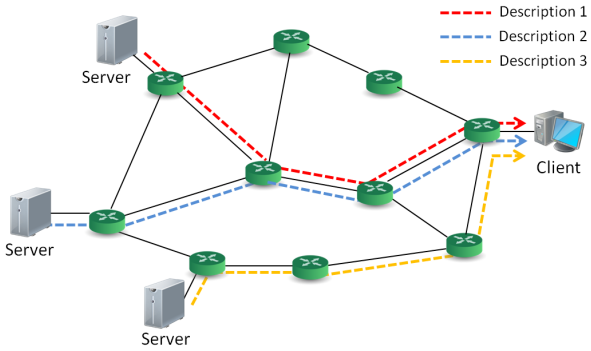
V. Future Directions

In this section, we present some future application areas of the proposed OpenFlow based architecture.

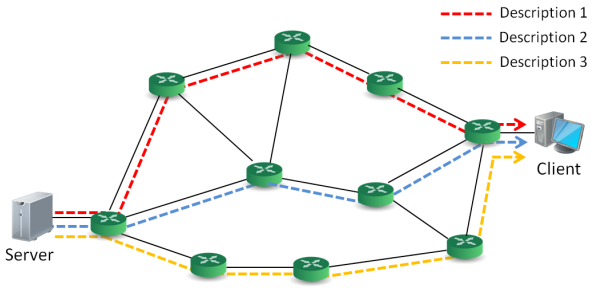
A. Video Streaming with Multiple Description Coding

Multiple description coding (MDC) is a technique [22] that encodes a source into multiple bitstreams (descriptions), where each description is independently decodable. Receiving only one description is sufficient for continuous playout, while the quality improves as the number of received descriptions increase. However, the loss of compression efficiency, the transmission overhead and high encoder/decoder complexity are the major drawbacks of MDC.

The main objective of MDC is to provide error resilience to packet losses by utilizing independent paths. Each description should be sent over different routes; because, in general, average route behaviour provides better performance than the behaviour of any individual random route. However, current Internet determines a single route (or single multicast tree) for source and destination pairs, so MDC cannot have path diversity when there is a single multimedia source (server). In order to take the advantage of MDC in the Internet, different descriptions have to be distributed over different sources to enable multi-path diversity. Hence, current MDC-based multimedia delivery proposals are limited to peer-to-peer (P2P) and content distribution networks (CDNs). On the

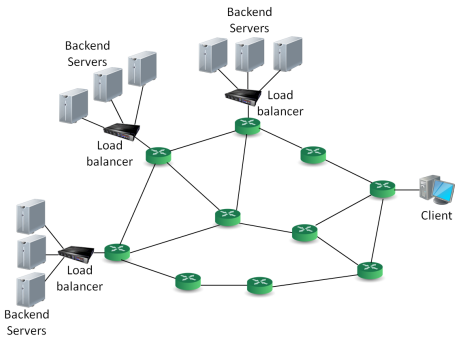


(a)

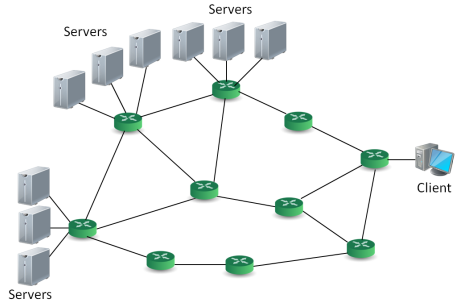


(b)

Fig. 9. Streaming three MDC descriptions to a client (a) over the Internet from multiple servers, (b) over OpenFlow from a single server



(a)



(b)

Fig. 10. Load balancing (a) over the Internet is limited to server selection, (b) over OpenFlow allows the joint selection of servers and routes

other hand, OpenFlow removes this limitation with its per-flow routing capability. In OpenFlow, each MDC description can be defined as a different flow and therefore, descriptions can be placed on disjoint or partially disjoint routes even if there is a single multimedia server. The routes of each description can be found using k-disjoint shortest path or can be further optimized by using constrained based disjoint routing algorithms. Unlike current works in literature, MDC streaming over OpenFlow does not require distributing descriptions among servers placed around the network, as illustrated in Fig.9.

B. Load Balancing in Content Distribution Networks (CDNs)

Load balancing is a networking methodology that distributes the workload across the network elements. The purpose of load balancing is providing a service from multiple servers by choosing an appropriate server. Therefore, it is essential for networking technologies such as content delivery networks (CDN), domain name systems (DNS) and newly emerging cloud services. It is usually implemented by a load balancing switch (i.e. load balancer) which forwards a request coming from a client to one of the servers which, in general, replies to the load balancer. This operation is done without the client which is unaware of the presence of the load balancer and other backend servers. A load balancer selects a server by using a variety of scheduling algorithms which may consider factors such as servers' reported load, servers' up/down frequencies, location of the servers (i.e. propagation delay), type of the requested content and the amount of traffic assigned to a server.

Content delivery networks are distributed system of servers that serve contents such as web objects, documents and multimedia to end-users with high availability and high performances. Especially, most of the current state-of-the-art multimedia streaming applications (e.g. live and on-demand streaming) over the Internet rely on CDNs, and load balancing is an integral part of the CDN. However, in the Internet, only server-based load balancing is possible. We can overcome this deficiency by using OpenFlow. In OpenFlow load balancing can be considered as a network primitive which does not require additional equipment that implements load balancing functions. Also, OpenFlow (Aster*x controller [7]) enables joint optimization of server and route selection which is not possible in the Internet, as illustrated in Fig.10.

C. Enabling Cross Layer Design in the Internet and OpenFlow Wireless

In the literature there are cross-layer designs for QoS routing over wireless networks (e.g. ad-hoc and sensor networks)[23], [24], but they cannot be implemented on wired networks. The Internet is a closed environment where researchers cannot easily experiment their ideas related to the core network such as routing. This is because, current Internet router vendors provide a hardware and associated software which is not open to its users. OpenFlow removes the boundaries of the traditional Internet; it provides completely open and programmable networking environment to the operators, enterprises, independent software vendors and users. It also allows researchers to develop their ideas similar to the cross-layer approaches

as in wireless networks. Even though our focus is on wired networks in this paper, there is an initial implementation of wireless extension of OpenFlow (OpenRoads)[25] on which OpenQoS can be implemented with a little effort.

VI. Conclusion

OpenQoS is a novel approach to stream video over OpenFlow networks with QoS. It is different from the current QoS mechanisms since we propose dynamic QoS routing to fulfill end-to-end QoS support which is possible with OpenFlow's centralized control capabilities over the network. Unlike other QoS architectures, OpenQoS minimizes the adverse effects (such as packet loss and latency) on other types of flows. Inspection of our experimental results yields the following observations:

- OpenQoS working along with TCP outperforms the state-of-the-art, HTTP-based multi-bitrate adaptive streaming, under network congestion.
- OpenQoS can guarantee seamless video delivery with little or no disturbance experienced by the end users even if an unreliable transport protocol, such as UDP, is used.
- If a reliable transport protocol, such as TCP, is used, OpenQoS can guarantee full video quality.

References

[1] J. H. Saltzer, D. P. Reed, and D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, Nov. 1984.

[2] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," RFC 1633, Internet Engineering Task Force, June 1994.

[3] E. Rosen and Y. Rekhter, "BGP/MPLS VPNs," RFC 2547, Internet Engineering Task Force, 1999.

[4] Open Networking Foundation. [Online]. Available: http://opennetworking.org

[5] Open Networking Foundation (ONF), "Software defined networking: the new norm for networks," 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/openflow/wp-sdn-newnorm.pdf

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[7] OpenFlow Consortium. [Online]. Available: http://openflowswitch.org

[8] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "An optimization framework for qos-enabled adaptive video streaming over OpenFlow networks," *IEEE Trans. on Multimedia*, to appear.

[9] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over OpenFlow networks: an optimization framework for QoS routing," in *Proc. IEEE International Conference on Image Processing (ICIP)*, Sept. 2011, pp. 2241–2244.

[10] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow networks," in *Proc. IEEE International Conference on Image Processing (ICIP)*, Sept.-Oct. 2012, pp. 2237–2240.

[11] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (rsvp) – version 1 functional specification," RFC 2205, Internet Engineering Task Force, September 1997.

[12] A. Juttner, B. Szviatovski, I. Mecs, and Z. Rajko, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE INFOCOM*, vol. 2, Apr. 2001, pp. 859–868.

[13] Floodlight. [Online]. Available: http://floodlight.openflowhub.org

[14] Nox. [Online]. Available: http://noxrepo.org

[15] Beacon controller. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home

[16] Z. Cai, A. L. Cox, and T. S. Eugene Ng, "Maestro: balancing fairness, latency and throughput in the OpenFlow control plane," Rice University Technical Report TR11-07, 2011.

[17] OpenFlow switch specification v1.0. [Online]. Available: http://openflow.org/wp/documents/

[18] V. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Communications Magazine*, vol. 32, no. 3, pp. 70–81, Mar. 1994.

[19] ffmpeg. [Online]. Available: http://ffmpeg.org

[20] VLC media player. [Online]. Available: http://videolan.org/vlc

[21] Flash media streaming server 4.5. [Online]. Available: http://www.adobe.com/products/flash-media-streaming.html

[22] V. Goyal, "Multiple description coding: compression meets the network," *Signal Processing Magazine, IEEE*, vol. 18, no. 5, pp. 74 –93, sep 2001.

[23] S. Misra, M. Reisslein, and X. Guoliang, "A survey of multimedia streaming in wireless sensor networks," *Communications Surveys Tutorials, IEEE*, vol. 10, no. 4, pp. 18–39, quarter 2008.

[24] Q. Zhang and Y.-Q. Zhang, "Cross-layer design for qos support in multihop wireless networks," *Proceedings of the IEEE*, vol. 96, no. 1, pp. 64 –76, jan. 2008.

[25] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "Openroads: empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, Jan. 2010.