Application-based QoE support with P4 and OpenFlow

Divyashri Bhat*†, Jason Anderson†, Paul Ruth‡, Michael Zink* and Kate Keahey§
University of Massachusetts Amherst*, University of Chicago†, RENCI‡, Argonne National Labs §
*dbhat,zink@ecs.umass.edu, †jasonanderson@uchicago.edu, ‡pruth@renci.org, §keahey@mcs.anl.gov,

***Abstract*—Although Software-Defined Wide Area Networks (SD-WANs) are now widely deployed in several production networks, they are largely restricted to traffic engineering approaches based on layer 4 (L4) of the network protocol stack that result in improved Quality-of-Service (QoS) of the network overall without necessarily focussing on a specific application. However, the emergence of application protocols such as QUIC and HTTP/2 needs an investigation of layer 5-based (L5) approaches in order to improve users' Quality-of-Experience (QoE). In this paper, we leverage the capabilities of flexible, P4-based switches that incorporate protocol-independent packet processing in order to intelligently route traffic based on application headers. We use Adaptive Bit Rate (ABR) video streaming as an example to show how such an approach can not only provide flexible traffic management but also improve application QoE. Our evaluation consists of an actual deployment in a research testbed, Chameleon, where we leverage the benefits of fast paths in order to retransmit video segments in higher qualities. Further, we analyze real-world ABR streaming sessions from a large-scale CDN and show that our approach can successfully maximize QoE for all users in the dataset.**

I. Introduction

While application protocols such as HTTP have evolved to provide reduced latency and efficient use of network resources [1], traffic engineering paradigms such as Software Defined Networking (SDN) have simultaneously emerged to provide better Quality-of-Service (QoS) through flexible routing and centralized network management. Several large-scale production Content Distribution Networks (CDNs) such as Google [2] have implemented Software-Defined Wide Area Networks (SD-WANs) to efficiently perform application-aware routing at the peering edge. Cisco [3], predicts that downstream application traffic will account for 82% of all Internet traffic by 2021. Moreover, the same report predicts that SD-WAN traffic will account for 25% of all WAN traffic by 2021.

On the application layer, HTTP/2 incorporates several improvements over its predecessor, HTTP/1, which include a) multiplexing several streams into one TCP connection, b) server-push approaches, where content is delivered to a client without explicitly requesting it, and c) header compression for reduced latency. These improvements, particularly stream multiplexing, were devised to reduce page load time such that download requests for embedded objects such as images, video, etc., in a web page can be issued simultaneously (instead of sequentially). Similarly, the QUIC [4] protocol was introduced as a transport layer candidate for HTTP/2 with one basic difference: QUIC is based on UDP and can thus, be used to implement flexible congestion control as well. As protocols become more versatile to support high-bandwidth applications such as Adaptive Bit-Rate (ABR) video streaming, Augmented Reality (AR) and Virtual Reality (VR) applications, network architectures need to adapt in order to meet the demands of such applications worldwide. More recently, the introduction of flexible switch architectures such as [5] have paved the way for line-rate processing of application-layer headers [6]. Our architecture investigates application-based QoS in centrally controlled networks. In particular, this work leverages the capability of protocol-independent packet processors (P4) [5] at the edge of the network to define a custom fixed-length application header and further, translate this into a Q-in-Q (802.1ad) tag [7] for the core network in order to perform QoS routing/provisioning.

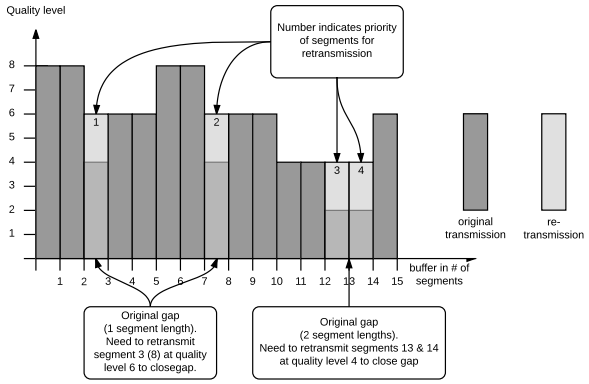


Fig. 1: Example scenario for retransmissions. The QoE of this streaming session can be improved if, e.g., segments 3, 8, 13, and 14 are retransmitted in higher quality, assuming they arrive before their scheduled playout.

The remainder of this paper provides a background of applications such as video streaming that can benefit from our approach in Sect. II. A detailed description of our architecture is given in Sect. III, followed by our Chameleon testbed setup description in Sect. IV. We then present an evaluation of our prototype in Sect. V followed by the Conclusion in Sect. VI.

II. Background

While we envision that several applications today can benefit from application header-based traffic engineering using P4, in this work we target popular applications such as video streaming in order to demonstrate the gains of our approach.

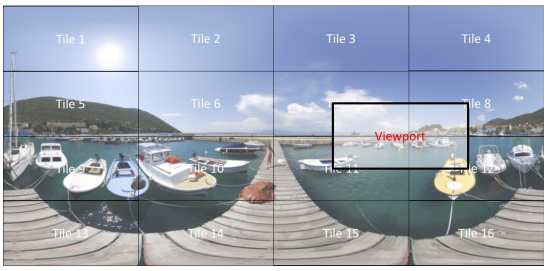


Fig. 2: Tile-based Virtual Reality (VR) video streaming. A viewport is characterized by multiple adjacent tiles that need to be simultaneously downloaded in order to provide a seamless viewing experience to the user.

First, we focus on ABR video streaming as an example and use Figure 1 to show a simplified example of segment qualities inside the video player buffer with different possible quality gaps. In previous work [8], we demonstrated how HTTP/2-based multiplexing can be used to simultaneously fetch multiple qualities of video segments (denoted *retransmissions*) in order to close such gaps and thereby, improve the Quality-of-Experience (QoE) of a client. In this work, we analyze how traffic engineering approaches can also be used to provide improved QoE by retransmitting segments using a faster path, when available.

The second example we consider focuses on more recently evolved 360 streaming applications for VR/AR that are quickly gaining traction as popular video streaming applications. Such high-bandwidth, latency sensitive applications continue to push the boundaries of current network architectures and drive innovation in traffic engineering approaches. Fig. 2 shows an example of tile-based 360 streaming where each scene is split into tiles that are rendered as the user moves their head. As is evident from this figure, such an application would require multiple tiles to be downloaded concurrently in order to render a "Viewport" for the user. Furthermore, each of these tiles may need to be retransmitted in higher qualities if they experience quality gaps as shown in Fig. 1. In order to provide an evaluation of simultaneous tile downloads, in this work we will also analyze the benefits of transmitting multiple streams over a fast path, when available.

In previous work [9], we demonstrated a prototype where HTTP/2 header information can be translated as a QoS requirement using P4-capable network elements to convert application layer header information into a Q-in-Q tag for differentiated routing via the core network using the Bring-Your-Own-Controller (BYOC) feature [10] provided by the Chameleon testbed [11]. In this work, we conduct extensive evaluations in order to analyze the QoE improvement for ABR streaming applications. We also use traces from an actual large-scale CDN to conduct an analysis of real-world ABR sessions to show that P4-based ABR segment retransmissions can be used to maximize the QoE for all sessions in the trace.

III. Design

In this section, we introduce the design of our approach by first describing application header-based traffic engineering followed by an explanation of the system architecture. Although from a designer's perspective it may seem attractive to perform application header-based traffic engineering throughout the network, we believe that from a network management and scalability vantage point, such type of traffic engineering is most suitable for the network edge. In the following, we describe the design of our system where application header-based traffic engineering at the edge can be seamlessly integrated with existing traffic forwarding contexts in core networks.

A. Application header-based Traffic Engineering

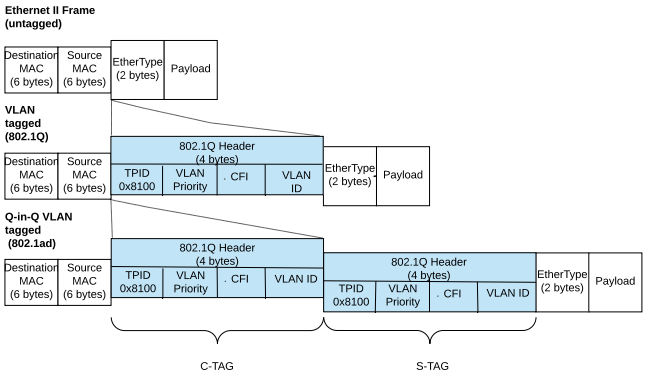


Fig. 3: VLAN tag header insertion in Layer-2. 802.1Q is followed by 802.1ad in order to differentiate between customers on the same VLAN.

1) Q-in-Q: The IEEE 802.1ad standard [7] double-tagging is introduced to allow network service providers to separate traffic from different VLANs as well as customers for better traffic management. Here, we use Q-in-Q tunneling, illustrated in Fig. 3, to translate application-layer header information into link-layer headers at the edge before packets are forwarded to the core network. In particular, we focus on HTTP/2 application headers since they explicitly provide header fields that can be easily interpreted into Q-in-Q tags for better manageability.

HTTP/2 Header Format

Length (2 bytes)	Type (1 byte)	Flags (1 bytes)	Stream ID (4 bytes)

Fig. 4: HTTP/2 Header Format: Common Eight-Byte Header

2) HTTP/2 Header: As the number of objects embedded within a web page began to increase, the overhead due to variable length headers resulted in increased page load times for HTTP/1.1. Contrarily, HTTP/2 introduces a fixed length header and performs header compression in order to reduce perceived latency and increase goodput [12]. It is interesting to note that HTTP/2 explicitly defines the *Stream ID* field (see Fig. 4) to multiplex several streams into a single TCP connection and thus, can be re-interpreted as a Q-in-Q tag

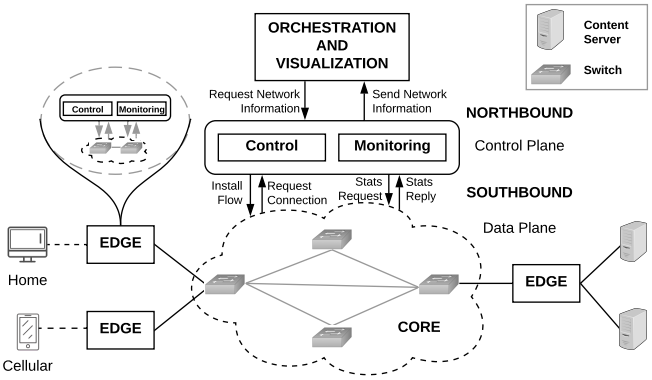


Fig. 5: Architecture for QoE to QoS translation at the edge, which is envisioned as SD-WANs

by any link-layer device. In this work, we redefine the outer customer tag (C-TAG) as a *Stream ID* tag using a flexible, protocol-independent packet processing language, P4 that can be programmed to interpret HTTP/2 headers. For the ABR video streaming application, *Stream ID* is used to differentiate between two distinct bitrate qualities that are simultaneously downloaded by the client as described in our previous work [8].

B. System Architecture

The main focus of our architecture is to translate application-layer header information into link-layer headers. We additionally include a centralized component that allows network providers to orchestrate and visualize their network from a single interface. Figure 5 presents the architecture of our system and consists of the following components:

1) The Core: For our architecture we assume a capability similar to that of a large-scale research testbed, ESNET[1], where the core or backbone network includes a programmable data-plane that performs fine-grained traffic engineering based on L2-L5 header information and is centrally controlled by an independent controller (denoted as *Monitoring* and *Control* in Fig. 5). However, we note that similar functionality can be incrementally deployed in production networks based on MPLS Traffic Engineering (MPLS-TE)[2] techniques as well.

2) The Edge: Innovation at the edge such as SD-WANs [2] is driven by the tremendous growth in downstream application traffic and the advent of cloud computing. Here, the edge network also includes a programmable data-plane of several flexible switches that are centrally controlled by an independent controller. However, these switches can perform fine-grained traffic engineering using L5 header information as well. In order to peer with the core, the edge controller must translate information in L5 headers to L2-L4 headers before sending packets out into the core.

3) Orchestration and Visualization: Our system also includes a centralized component that aggregates monitoring information from the various controllers in order to provide a visual representation of network performance.

The diagram also shows clients that connect to the edge via wireless home networks and cellular networks and receive requested content from a server connected to a different edge. The two edge networks are connected by the core. In the following section, we describe the setup for our experiments including the platform and tools we use to run QoE translation experiments.

IV. Setup

A. Chameleon Testbed

The Chameleon testbed is a deeply reconfigurable testbed that is suitable for large-scale prototyping of distributed systems and software defined networks. In this work, we leverage the recently released Bring-Your-Own-Controller feature [10] along with previously existing capabilities of the Chameleon Cloud to create a prototype of our architecture. Figure 6 shows the setup of our testbed, which is described next in more detail.

1) HTTP/2 application: For the HTTP/2 application we instantiate two bare-metal nodes: one that runs a Web server using the open source `Caddy` (version=0.10.10) software and another that runs an ABR video streaming client, `AStream`[3], that we modify to use an open-source Python-based library, `hyper`[4], that downloads video content using HTTP/2.

2) P4 Switch[5]: We install the behavioral model, `BMV2`[5], software switch in a bare-metal node, which emulates a P4-capable switch, and then use the `P4Runtime`[6] tool to programmatically install rules into each switch. In the future, we plan to replace this component with a hardware ASIC P4 switch [13] that has only recently become available.

3) OpenFlow Switch: OpenFlow [14], a widely-used implementation of SDN, is available to experimenters as a Virtual Forwarding Context (VFC), a functionality provided by Corsa switches, which enables each user to provision a nearly-isolated instance of an OpenFlow (v1.3) switch. A single Corsa DP2000 switch deployed within the Chameleon testbed at each site provides experimenters with performance measurements for a virtual instance of a hardware switch comparable to those used in production networks today. Each VFC can be programmed to connect to an individual SDN controller such that each user's experiment is truly isolated from others. After HTTP/2 headers are translated into Q-in-Q tags as described in Sect. III-A2, application packets are forwarded through the Corsa switch into the core network.

4) Core Network: In this setup, we provision two VLAN circuits (denoted as *Circuit1* and *Circuit2*) between the University of Chicago (UC) and the Texas Advanced Computing Center (TACC) using the Advanced Layer-2 Service (AL2S)

[1]https://www.es.net/network-r-and-d/experimental-network-testbeds/100g-sdn-testbed/

[2]https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/multiprotocol-label-switching-traffic-engineering/

[3]https://github.com/pari685/AStream

[4]https://github.com/Lukasa/hyper

[5]https://github.com/p4lang/behavioral-model

[6]https://github.com/p4lang/PI

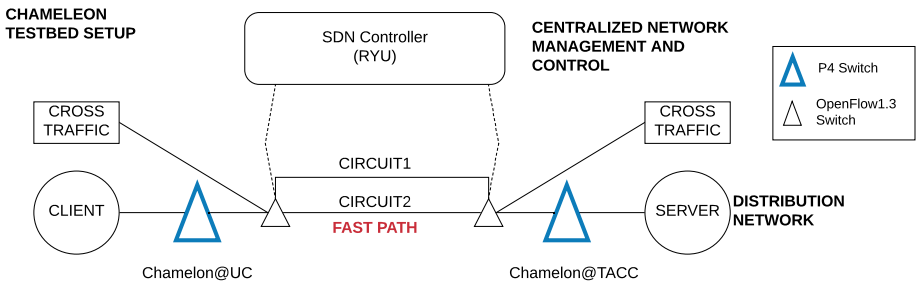


Fig. 6: Chameleon Testbed Setup: A HTTP/2 based video streaming client uses two disjoint paths to request multiple qualities of the same segment. Note: The disjoint points provide us with a setup to run controlled measurements with differential QoS features and study their effects on ABR retransmissions.

implemented by Internet2, which is a network service provider for collaborative research[7]. Note that the cross traffic nodes are bare-metal machines used to limit network bandwidth to 1Gbps using `Iperf3`[8] on *Circuit1*.

5) Centralized Management and Control: For orchestration and visualization, we use Jupyter Notebooks [15], an open-source web tool particularly suited for reproducible experiments. For this evaluation, Jupyter runs inside Chameleon and provides us with a single interface not only to run the controller and the ABR video streaming application but also to visualize network traffic and QoE metrics.

V. Evaluation

A. Download Time and Throughput

Figure 7 illustrates the performance improvement for ABR segment download time and throughput with the use of P4-based traffic engineering. In order to provide statistical significance to our evaluations, each experiment is repeated ten times and average and standard deviations are presented. *Circuit1* and *Circuit2* are 10Gbps isolated paths provisioned by AL2S between Chicago and Austin with an inherent delay of 30ms and a loss of 0%. Here, we consider a *Baseline* case, where all ABR segment are transmitted on the same path versus the case where retransmitted segments are sent over a prioritized path (denoted *FAST PATH* in Fig. 6). In order to observe the benefits of using a *FAST PATH* for application data transfer, all experiments described in this section are conducted by increasing loss and delay on *Circuit1*, unless stated otherwise. In order to make our analysis more generally applicable, we first describe performance improvements for segment download time and application throughput, which are subsequently translated to QoE metrics for ABR streaming towards the end of this section.

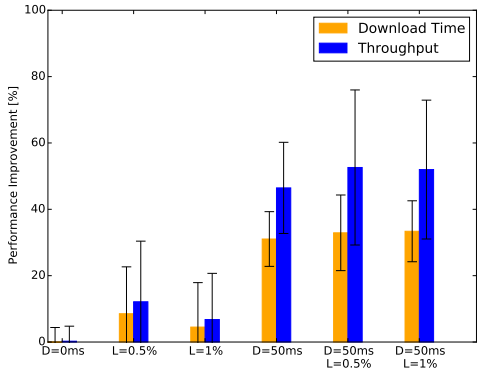
1) Single HTTP/2 Streams: Fig. 7a presents experiments where a single stream is provisioned on each path, i.e., original segments are downloaded on *Circuit1* and retransmitted segments are downloaded on *Circuit2*. The download time and application throughput for simultaneous segment downloads is compared against a *Baseline* case where both original and retransmitted segments are downloaded on *Circuit1*. Here, we evaluate five different network conditions of increasing delay (denoted *D*) and loss (denoted *L*) on *Circuit1* in order to study the benefit of using the *FAST PATH* for a single retransmission stream. While it is obvious that no improvement is observed in the case where *D=0* with no added delay on both circuits, significant improvements for download time and throughput are observed under different network conditions. We note that relatively higher performance improvements are observed when compared with high delay networks and we attribute this to the following: The application we show here uses TCP which is impacted by its slow start behavior and therefore, high round-trip times drastically impact the average throughput, especially for downloads of a short duration.

2) Multiple HTTP/2 Streams: While the above experiment shows L5 header-based traffic engineering is feasible and beneficial for ABR streaming sessions where a maximum of two streams are downloaded simultaneously, emerging applications such as 360 VR/AR streaming generally consist of multiple concurrent streams per session as illustrated in Fig. 2. Thus, Fig. 7b extends the evaluation of our system to include performance improvements when multiple segments are downloaded simultaneously. Since considerable improvement is observed in the case where *Circuit1* is characterized by a delay of 50ms and average loss of 0.5%, we focus on this scenario for evaluating cases where the number of streams is systematically increased. We note that while benefits are highest when a single stream is transmitted on each path, throughput is improved by about 30% and segment download time by more than 20% in cases when the number of streams per path are increased to two, four and eight, respectively. In order to further understand the scalability of the system, the next set of experiments focusses solely on the processing overhead observed at the P4 software switch when the number of streams is systematically increased.

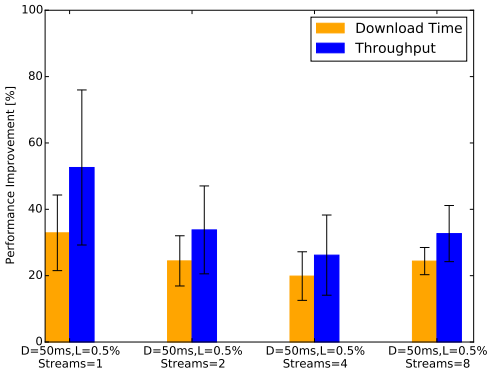
3) Processing Overhead - BMV2: Fig. 8 shows the percentage of processing overhead for the software switch in comparison to the case where a single stream is downloaded on each path. As the number of streams exponentially increases per path, we observe that the duration of segment download is correspondingly higher. We note that since these experiments are conducted with zero added delay and loss on each path and the software switch runs within a standalone bare-metal machine, any differences observed can be attributed to the

[7]https://www.internet2.edu/products-services/advanced-networking/layer-2-services/#features-al2s

[8]https://iperf.fr/iperf-download.php



(a) Original and retransmitted segment streams are downloaded on separate paths.



(b) Several streams are downloaded simultaneously where the number of streams is symmetrically distributed between two paths.

Fig. 7: Performance improvements for file transfer time and application throughput under varying network conditions. Comparisons with a baseline case where all streams are downloaded on a single path, i.e., *Circuit1* show significant benefits when a second, faster path is used for downloading retransmitted segments.

CPU processing time on the software switch.

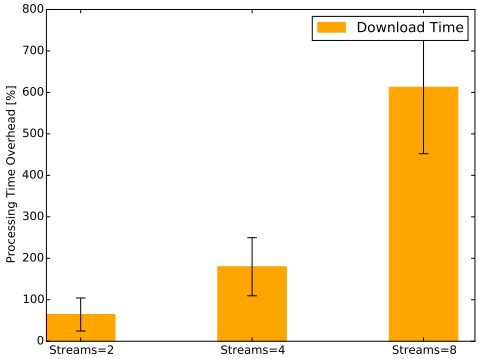


Fig. 8: Processing time overhead for the P4 software switch with simultaneous HTTP/2 streams shows that performance is CPU-bound.

B. QoE metrics for ABR Streaming

For the evaluation of the performance impact of the traffic engineering approach we present in this paper on ABR streaming, we make use of the following metrics, which are widely used in related work:

Average Quality Bitrate (AQB): One of the objectives of quality adaptation algorithms is to maximize the average quality bitrate of streamed video. Note that an increased application throughput contributes to an improvement in the average quality bitrate for ABR video streaming. For a comprehensive QoE representation, we need to combine this metric with the *Number of Quality Switches* which is explained below.

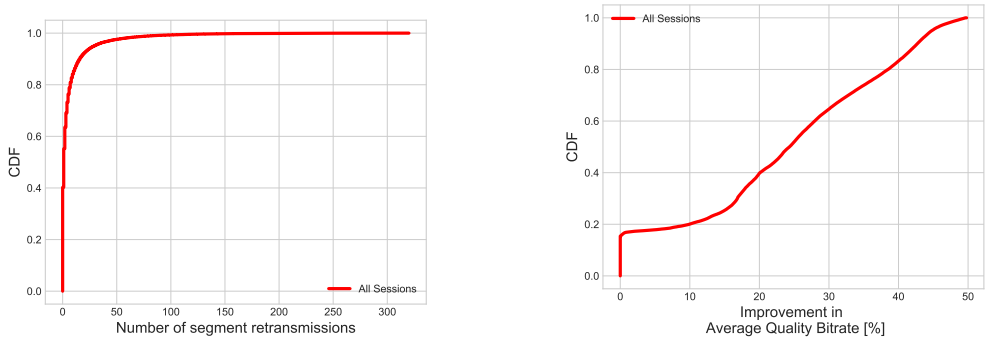
Number of Quality Switches ($\#QS$): This metric is used together with AQB to draw quantitative conclusions about the perceived quality (QoE). For example, for two streaming sessions having the same AQB, the session with the lower $\#QS$ will be perceived better by the viewer [16]. A reduction in the number of quality gaps due to retransmissions results in a corresponding reduction in the number of quality switches for an ABR streaming session.

C. Trace-based simulation

In this section, we analyze actual sessions from a popular CDN (Akamai) [17] in order to obtain an estimate of QoE improvements when differentiated traffic engineering is performed for original and retransmitted segments. This dataset contains video streaming session information for a 3-day period in June 2014. The ABR streaming traffic in this trace contains 5 million video sessions originating from over 200,000 unique clients who were served by 1294 edge servers around the world. For each streaming session, each individual segment request is logged, which allows us to reconstruct the quality of the segments received at the client.

Fig. 9a shows the average number of retransmissions per session that need to be made in order to close quality gaps as illustrated in Fig. 1. We note that about 36.19% of sessions require at least one retransmission while some sessions may even require up to 300 retransmissions [8].

Fig. 9b illustrates the percentage improvement in average quality bitrate required in order to maximize QoE for each session. For example, 20% of the video streaming sessions in this dataset only need a 10% improvement in average quality bitrate in order to maximize their QoE. When we combine this with the results obtained in Fig. 7a, assuming that a similar quality profile is observed in our testbed network, we note that nearly all sessions can maximize their QoE by using a high priority path for retransmitting segments in a higher quality. These benefits are especially significant when the default path suffers from large round-trip times as high as 50ms. Further, the number of quality changes, $\#QS$, are minimized by performing retransmissions using the *FAST PATH* whenever necessary, since this is a natural by-product of closing gaps.



(a) (b)

Fig. 9: Analysis from a real-world trace show projected QoE improvements with the use of ABR segment retransmissions.

Although we present relative improvements to QoE with the use of P4 software switches to perform application header-based traffic engineering, it is important to note that their performance is CPU-bound and not I/O bound. It is, therefore, recommended to consider software switches only for prototyping systems and implement hardware switches such as Barefoot's Tofino [13] in production networks.

VI. Conclusion

In this work, we show how flexible switches at the edge can be used to translate application layer header information into link layer headers to differentially route distinct qualities of ABR video segments in order to improve the average quality bitrate of a HTTP/2-based video streaming application. We performed evaluations in a geographically distributed testbed, Chameleon, using open source orchestration and visualization tools. Our results show that application header-based traffic engineering techniques can vastly improve users' QoE. In order to conduct large-scale performance evaluations of QoE improvements for P4-based traffic engineering approaches, our future work will focus on using hardware switches that process L5 headers at line-rate speeds.

VII. Acknowledgments

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357 and the DyNamo grant awarded by the National Science Foundation, Office of Advanced Cyberinfrastructure under contract number #1826997.

References

[1] R. P. Mike Belshe and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," Internet Requests for Comments, RFC Editor, RFC 7540, May 2015. [Online]. Available: https://tools.ietf.org/rfc/rfc7540.txt

[2] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood, M. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 432–445. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098854

[3] Cisco, "The zettabyte era: Trends and analysis," 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.pdf

[4] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 183–196. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098842

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[6] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 121–136.

[7] *Provider Bridges*, IEEE Std. 802.1ad, 2006.

[8] D. Bhat, R. Deshmukh, and M. Zink, "Improving qoe of abr streaming sessions through quic retransmissions," in *Proceedings of the 26th ACM International Conference on Multimedia*, ser. MM '18. New York, NY, USA: ACM, 2018, pp. 1616–1624. [Online]. Available: http://doi.acm.org/10.1145/3240508.3240664

[9] D. Bhat, J. Anderson, P. Ruth, M. Zink, and K. Keahey, "Application-based qos support with p4 and openflow: A demonstration using chameleon," *Semantic Scholar*, 2018.

[10] P. Ruth, "Software-defined networking with chameleon," 2018. [Online]. Available: https://chameleoncloud.readthedocs.io/en/latest/technical/networks/networks_sdn.html

[11] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Mambretti, P. Rad, and P. Ruth, "Chameleon: a scalable production testbed for computer science research," in *Contemporary High Performance Computing vol. 3. Ed. Jeff Vetter.*, 2017.

[12] I. Grigorik and Surma, "Introduction to http/2," 2019. [Online]. Available: https://developers.google.com/web/fundamentals/performance/http2/

[13] Barefoot, "Tofino: World's fastest p4-programmable ethernet switch asics," 2019. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino/

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, Mar. 2008.

[15] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, "Jupyter notebooks-a publishing format for reproducible computational workflows." in *ELPUB*, 2016, pp. 87–90.

[16] M. Zink, J. Schmitt, and R. Steinmetz, "Layer-encoded video in scalable adaptive streaming," *IEEE Transactions on Multimedia*, vol. 7, no. 1, pp. 75–84, Feb 2005.

[17] D. K. Krishnappa, M. Zink, and R. K. Sitaraman, "Optimizing the video transcoding workflow in content delivery networks," in *Proceedings of the 6th ACM Multimedia Systems Conference*, ser. MMSys '15. New York, NY, USA: ACM, 2015, pp. 37–48. [Online]. Available: http://doi.acm.org/10.1145/2713168.2713175

[18] D. Bhat, "Application header-based p4 support in chameleon." 2019. [Online]. Available: https://github.com/dbhat/lcndemo2018/tree/master/reproducibility_artifacts

APPENDIX A
DEMO: APPLICATION-BASED QOE SUPPORT WITH P4 AND OPENFLOW

A. Abstract

This system has been tested and evaluated in Chameleon [11], a deeply reconfigurable testbed that is suitable for large-scale prototyping of distributed systems and software defined networks. In order to experiment with Chameleon, you will need an account for access which is available free to the research community. We will use the testbed setup described in Fig. 6 to systematically compare QoE metrics such as average quality bitrate for two traffic engineering approaches: one where ABR video streaming retransmissions are differentially routed using a flexible switch and another where retransmissions are classified as regular HTTP traffic without any preferential treatment. Since all of the experiment components are located in a public cloud, for this demonstration we will require a large monitor with a HDMI connector to allow conference attendees to view as well as use the Jupyter web instance to interact with our experiment and two power outlets. The following subsystems are included (More details and source code can be found in [18]:

B. Chameleon Testbed

In this work, we leverage the recently released Bring-Your-Own-Controller feature along with previously existing capabilities of the Chameleon Cloud to create a prototype of our architecture. In order to reserve resources within the Chameleon testbed you will first need to create an account, following which an XML description file (also known as a Heat template) must be used for resource specification. Figure 6 shows the setup of our testbed, which we describe in detail.

1) HTTP/2 application: For the HTTP/2 application we instantiate two bare-metal nodes: one that emulates a Web server using the open source `Caddy` (version=0.10.10) software and another that emulates an ABR video streaming client, `AStream`[1], that we modify to use an open-source Python-based library, `hyper`[2], that downloads video content using HTTP/2.

2) Cross Traffic: The cross traffic nodes are bare-metal machines used to create various network congestion scenarios using `Iperf3`[3] for controlled experiments. For these experiments, we use UDP traffic to create a constant bit-rate traffic in order to emulate a congested network path.

```
iperf3 -c <server_ip> -t 20 -i 5
-b 900Mbps -u -t 500
```

3) P4 Switch [5]: We install the behavioral model, `BMV2`[4], software switch components in a bare-metal node, which emulates a P4-capable switch, and then use the `P4Runtime`[5] tool to programmatically install rules into each switch. Note that we use two different implementations of the P4 switch; one for the Baseline case where all streams are routed on a single network path (available as basic_single.p4 in [18]) and another for the test case where half of the streams are split equally between *Circuit1* and *Circuit2* (available as basic.p4 in [18]).

4) Network Emulations: In order to emulate varying loss and delay on *Circuit1*, we make use of *tc*[6] to create a traffic class and *NetEm*[7] in order to define network conditions for the traffic class. An example command which is used to induce a round-trip delay of 50ms and a loss of 0.5% is as follows:

```
sudo tc qdisc add dev <interface_name>
root netem delay 25ms loss 0.5%
```

5) OpenFlow Switch: OpenFlow [14], a widely-used implementation of SDN, is available to experimenters as a Virtual Forwarding Context (VFC), a functionality provided by Corsa switches, which enables each testbed user to provision a nearly-isolated instance of an OpenFlow (v1.3) switch. After HTTP/2 headers are translated into Q-in-Q tags as described in Sect. III-A2, the application packets are forwarded through the Corsa switch into the core network.

6) Core Network: In this demonstration, we provision two VLAN circuits (denoted as *Circuit1* and *Circuit2*) between University of Chicago (UC) and Texas Advanced Computing Center (TACC) using the Advanced Layer-2 Service (AL2S) implemented by Internet2, which is a network service provider for collaborative research[8].

7) Centralized Management and Control: For orchestration and visualization, we use Jupyter Notebooks [15], an open-source web tool particularly suited for reproducible experiments. For this demonstration, Jupyter runs inside Chameleon and provides us with a single interface not only to run the controller and the ABR video streaming application but also to visualize network traffic and QoE metrics. Our OpenFlow controller is implemented using the Python-based *RYU*[9] application, where we sepcify forwarding contexts that include VLAN tag identifiers for flow rule specification.

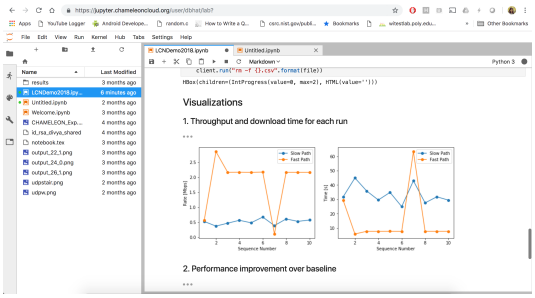


Fig. 10: JupyterLab: Web GUI for orchestration and live visualization of experiments on Chameleon

[1] https://github.com/pari685/AStream
[2] https://github.com/Lukasa/hyper
[3] https://iperf.fr/iperf-download.php
[4] https://github.com/p4lang/behavioral-model
[5] https://github.com/p4lang/PI
[6] https://linux.die.net/man/8/tc
[7] http://man7.org/linux/man-pages/man8/tc-netem.8.html
[8] https://www.internet2.edu/products-services/advanced-networking/layer-2-services/#features-al2s
[9] https://osrg.github.io/ryu/