Omnidirectional Adaptive Bitrate Media Delivery using MPTCP/QUIC over an SDN Architecture

Brian Hayes[1], Yusun Chang[2], and George Riley[1]

[1]Georgia Institute of Technology, Atlanta, GA, USA, bhayes8@gatech.edu, riley@ece.gatech.edu
[2]Kennesaw State University, Atlanta, GA, USA, yusun@kennesaw.edu

***Abstract*—Omnidirectional streaming requirements can be an impediment to delivering virtual reality (VR) quality content at scale. Leveraging multihomed device network options becomes a viable approach to ease congestion. Multipath TCP (MPTCP) has the ability to utilize multiple networks simultaneously, but its performance can suffer from excessive overhead due to its use of large reordering buffers. In this paper, we propose a unique approach to removing this limitation with a software-defined networking architecture. Using Mininet with Floodlight controllers to manage available paths, we designed a framework that actively manages Quick UDP Internet Connections (QUIC) and/or MPTCP network usage based on network characteristics. Testbed experiments reveal that the start-up buffering delay is decreased by up to 60% or more when using our algorithm in networks with packet loss rates above 2%. These results are favorable even when compared against another modern multipath algorithm that attempts to remedy this issue. Our analysis shows our framework outperforms standard MPTCP in network usage efficiency by up to 50% at packet loss rates above 1%.**

***Index Terms*—MPEG-DASH, SDN, ABR, MPTCP, QUIC**

I. INTRODUCTION

From large social media companies to individual users, omnidirectional media has the potential to dramatically increase bandwidth requirements for streaming. This rapid innovation has led to a demand for more robust multimedia solutions. High bandwidth wireless technologies (e.g., LTE and 5G) have significantly improved network performance. However, timeliness and infrastructure costs required to upgrade systems can limit options. Quickly and effectively leveraging existing infrastructure is essential to meeting future network requirements. Estimates by Cisco [1] indicate that by 2019, 80% of all consumer Internet traffic will be video; up from 64% in 2014.

Adaptive bitrate (ABR) provides the ability to stream multimedia in varying network conditions to a wide range of devices. Fig. 1 illustrates a high-level flow diagram demonstrating ABR streaming from production to delivery. ABR streaming works by taking a media stream and segmenting it into a few seconds of content called a *chunk*. After downloading a manifest file, the client downloads a *chunk* for playback. This allows players to switch between higher or lower bitrates during a stream based on a player's configuration.

Popular ABR multimedia protocols include HTTP Live Streaming (HLS) [2] and MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [3]. The DASH Industry Forum (DASH-IF) established guidelines for MPEG-DASH usage supported by Netflix, Microsoft, Google, Ericsson, Samsung, Adobe, and others. In addition, Google has released an open-source continuously updated VR View [4] used in this work.

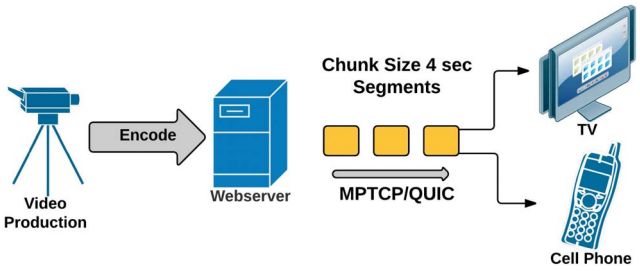


Fig. 1: Example adaptive bitrate streaming flow diagram

QUIC version 38 [5] has been implemented over UDP to address many of TCP limitations, one of which includes placing control into application space at both endpoints rather than kernel space. Fig. 2 illustrates QUIC's connection establishment compared to TCP with TLS. In particular, we are interested in the reduced latency (e.g., 0-round-trip time (RTT) connectivity overhead) and stream-multiplexing support. Today's mobile devices have multiple network interfaces, such as Wi-Fi and LTE, but they are not leveraged simultaneously to pool available resources from existing networks in practice. Multipath TCP (MPTCP) [6] is an experimental standard of the Internet Engineering Task Force (IETF). It is also a backward compatible extension to TCP. MPTCP has a built-in handover mechanism that is transparent to the application layer. This allows individual network links to become available or drop during a connection's lifetime without disrupting the overall end-to-end connection.

Software-defined Networking (SDN) is meant to address problems of traditional static network architecture. SDN decouples decisions about where data is sent from the hardware that does the forwarding. We leverage SDN in our testbed to dynamically control MPTCP's available paths in a way that is transparent to MPTCP and the applications. To date, there has been little focus on the performance of omnidirectional media in lossy multipath networks. In short, we propose a unique approach to removing a limitation of MPTCP using software-defined networking

978-1-5090-5019-2/17/$31.00 ©2017 IEEE

to effectively stream omnidirectional video. Additionally, we design a framework that actively manages the use of QUIC or MPTCP to initiate connections based on network characteristics for modern multihomed devices.

The paper is organized as follows: Section II summarizes prior work and some limitations. Section III discusses each strategy of the proposed protocol in detail. Section IV covers the setup of our testbed and experiment parameters. Section V presents our results and figures. Finally, Section VI discusses the strengths, limitations, and future work of our proposed solution.

II. Related Work

Research conducted related to this topic involve the implementation of QUIC comparing its goodput, and link utilization to SPDY and HTTP [7]. Biswal et al. [8] primarily focused on measuring web page load times in wired and cellular networks, concluding that QUIC is robust under poor network conditions. Sonkoly et al. [9] designed a large-scale multipath testbed with GEANT OpenFlow to allow for the testing of real world traffic performance. Another study focused on multipath data centers with optical software-defined networks [10]. Sandri et al. [11] conducted a study that focused on improving the performance of shared bottlenecks using OpenFlow and MPTCP. Nam et al. [12] investigates multipath networks with POX controllers comparing MPTCP to SPTCP. Their algorithm focused primarily on bandwidth estimation. Many attributes specific to video streaming, like buffering delays, were not explored. Google researchers claim 50% of the traffic from Chrome browsers leverage QUIC [13]. Furthermore, Google claims that QUIC's performance improvement is increasingly noticeable to video services (e.g., YouTube) resulting in 30% less buffering.

Taking advantage of multipath characteristics in ABR over TCP has been explored in the past [14]–[16] and could improve video streaming performance by utilizing several TCP connections simultaneously. Another study [17] investigates HTTP/2 server push with MPTCP concluding that MPTCP's large reorder buffer is problematic in low bandwidth networks. These studies however primarily rely on bandwidth estimation or buffer state on the client-side. Our work looks to build upon these previous efforts in the area of multipath data transport to improve ABR video streaming. To our knowledge, we have not found previous research using MPTCP and QUIC for ABR optimization in lossy networks.

Client-side ABR algorithms are challenged with achieving a high quality of experience (QoE) in unpredictable network conditions. These fundamental problems are discussed in the following study by Bae et al. [18]. In particular, there are issues with suboptimal bandwidth estimation tied to *chunk* duration and poor bitrate decisions. This is due to the lack of correct bitrate encoding per *chunk*, and poor network utilization as a result of inflexible transport protocols. MPTCP with SDN and QUIC can address many of these concerns.

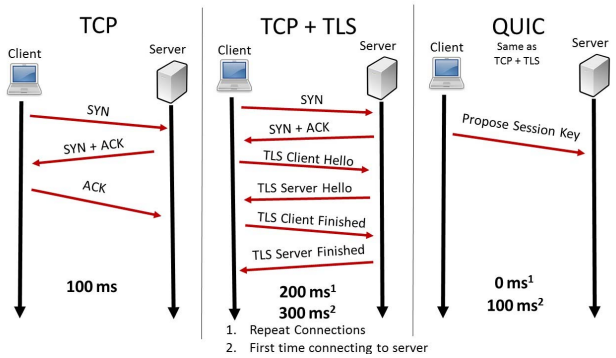


Fig. 2: QUIC RTT Connection Establishment Overview

Adaptable transport protocols in conjunction with ABR algorithms can simultaneously optimize video quality while minimizing rebuffering and bitrate switches. We have developed a framework that actively manages QUIC and MPTCP connections for streaming omnidirectional video in lossy networks addressing concerns mentioned in prior work.

III. Proposed System

In this section, we propose two new ideas to improve omnidirectional video streaming in lossy environments by leveraging MPTCP, QUIC, and SDN into a single algorithm. The first strategy is called a start-up decision engine, in which we designed an SDN application linked with Google VR Viewer to reduce the client initialization delay by up to 60% in lossy networks. This is done by utilizing SDN's ability to monitor network statistics on various paths in real time. Using that information, our algorithm dynamically modifies the most appropriate transport protocol. The second strategy called playback network monitoring uses MPTCP's *MPTCP_INFO* option and SDN's network path statistics. The algorithm monitors the number of retransmits each subflow experiences along with continually monitoring the average RTT of each subflow. If either metric surpasses a tunable threshold, the network is modified to improve performance. The algorithm can add paths if available or asynchronously abort unsustainable quality levels on individual networks without waiting for timeouts. Finally, we integrate each strategy into one larger protocol and evaluate its overall performance.

A. Start-up Decision Engine

For our first contribution, we implemented a start-up decision engine. The goal of the start-up decision engine is to reduce the initialization overhead. Studies [12], [14], [15], [17] show that systems that use unbalanced multipath networks often experience worse performance than simply using a single network. MPTCP, in particular, can suffer from bloated reordering buffers causing data to arrive in bursts instead of a steady stream which can create problems for omnidirectional video streaming. In low bandwidth networks, MPTCP and standard TCP can be slow to react

properly for streaming video. QUIC being UDP-based provides much better performance for streaming video according to Google [13].

We propose a way to reduce start-up delays for clients (reference the connection setup of Algorithm 1). If only one path is available to the server, meaning there is only one active network available to the client, then the application defaults to using the QUIC protocol. This is because it performs better than TCP in most situations [13]. If two networks are available to the client, the SDN application compares the differences in the average RTT for each network path. If the difference in the average RTTs are larger than 9:1 [19] (e.g. 90Mbps and 1Mbps) the algorithm ignores the slower path and only uses the QUIC protocol on the lowest RTT network. Lastly, if there are 2 or more networks available and the difference between networks compared with the lowest RTT path is less than 9:1, those paths are retained and MPTCP is utilized. MPTCP's performance in balanced networks performs generally better than single path networks [14] because of the aggregate throughput. The RTT threshold is tunable and can be modified for other applications. This simple but elegant use of SDN, QUIC, and MPTCP has implications for start-up delay improvements for short form video content as well (e.g., Snapchat, Vine, etc.). Drawbacks of this method, however, include networks that do not allow UDP-based streaming nor support SDN networks. For our experiments, we do not consider this situation as we are primarily concerned with delivering video to networks that do support these options. However, we can mitigate this effect in real-world use by falling back to traditional TCP which MPTCP supports.

B. Playback Network Monitoring

For our second contribution, we used SDN and MPTCP's *MPTCP_INFO* option to maintain continuous high-quality playback. *MPTCP_INFO* provides applications with statistics at the MPTCP-level (e.g., state, retransmits, tcp_info for all other subflows, etc.). Another problem MPTCP experiences is the potential for a high number of retransmissions and duplicate ACKs in unbalanced networks [12]. This occurs when packets are buffered too long due to being delivered out of order, in which they arrive at the application late.

The connection in progress section of Algorithm 1 presents this solution. The SDN application continues like before in the connection setup by monitoring the average difference in the lowest RTT to each subflow. If a path falls below the 9:1 threshold, mentioned in the start-up connection, the SDN application requests a path change through the SDN controller. If another path is not available that meets that requirement, that path is removed from use, but is constantly monitored in case its performance improves to be rejoined at a later time with *MP_JOIN*. Additionally, *MPTCP_INFO* provides the number of retransmissions a path experiences over the lifetime of the connection. Monitoring the number of retransmissions for each flow indirectly correlates with the size of the out of order buffer. As a result, to preemptively keep it small for better performance, the SDN application will eliminate paths from use when using MPTCP that surpass a retransmit threshold. Similar to the RTT threshold, this is a tunable value based on the application. We used a threshold of 10% above the number of retransmits of the highest RTT network in use. This study [15] showed retransmissions above that threshold tend to excessively bloat the receive buffer in low latency networks. Lastly, when the client's playback buffer is critically low and there is only one path available, but the connection was started with MPTCP, we send a *RST_STREAM* frame. The server immediately upon receiving this signal ends the connection and the client starts a 0-RTT QUIC connection with the server. Using this technique allows for more path diversity to absorb network disruptions on available paths. An additional benefit of this strategy is that during low bandwidth periods, it avoids three RTTs that are usually required in connection establishments. This technique will be increasingly effective in networks with higher RTTs. A drawback of this method, however, is the potential of frequent oscillations between MPTCP and QUIC. This situation can be mitigated by changing the retransmit threshold for dropping the connection.

Algorithm 1 MPTCP/QUIC SDN Decision Engine

procedure ROUTE-SELECTION
    retransThres ← restransmission threshold
    rttThres ← threshold for the change in RTT
    Δ*rttList* ← RTT compared to lowest available
    rttList ← live avg RTT for each available path
    numSubFlows ← number of subFlows available

Connection Setup:
    if *numSubFlows* $\equiv 1$ **then** use QUIC protocol
    if (*numSubFlows* $\equiv 2$) and (Δ*rttList* $< rttThres$) **then** use MPTCP
    else
        if *numSubFlows* $\equiv 2$ **then** use QUIC on lowest RTT path
    if *numSubFlows* > 2 **then** using *rttList* if any path Δ*rttList* $> rttThres$ remove path using SDN if two or more paths remain use MPTCP otherwise use QUIC on lowest RTT

Connection in Progress:
    while *numSubFlows* > 0 **do** monitor retransmissions and avg RTT on each path if any pass *rttThres* or *retransThres* remove the path unless there is only one remaining, in that case restart the connection with QUIC protocol

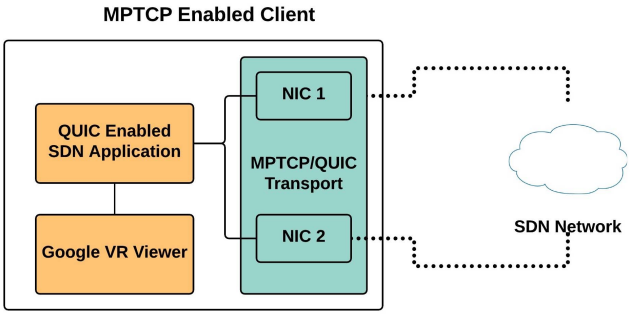


Fig. 3: MPTCP/QUIC enabled client setup used during experimentation

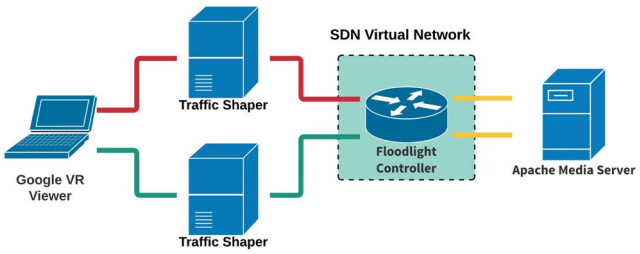


Fig. 4: Experimental lab testbed setup

IV. Experiment Setup

As a proof of concept, we implemented these two strategies into one solution since each focuses on a different portion of the streaming experience. All of the experiments were conducted using our own encoded ABR VR videos captured with Google's Cardboard App [20]. The content is encoded with x264 [21] at 20 different well-spaced bitrates between 45 Kbps and 3.8 Mbps. To compare the performance of SPTCP, TCP, and MPTCP with our implementation, we performed the following experiments while varying the network conditions. For all experiments, the aggregate bandwidth available on the MPTCP network matched the bandwidth of the single TCP flow. For example, if the conventional TCP network has 2.6 Mbps available, the MPTCP equivalent will have two networks available of 1.3 Mbps for each network. From the start of the experiment to $t = 120s$, the available bandwidth is 2.6 Mbps. At $t = 120s$ until $t = 240s$, the available bandwidth is reduced to 1.3 Mbps. At $t = 240s$, the available bandwidth is increased to 2.6 Mbps until the end at $t = 360s$. Based on measurements from past studies, the client to server RTT was set to 120ms and the 4s segmentation was chosen [22], [16]. The packet loss rate varied at discrete points of 1% increments from 0% to 5% based on [23]. Given many existing studies are evaluated in lossless networks, real-world representative results occur when considering lossy networks.

We logged data from the experiments via the JavaScript console provided by the Google Chrome web browser and Wireshark captures. To compare performance, we studied the following metrics:

1) *Start-up buffering delay*; the time from when the first video segment is requested to when playback begins.
2) *Playback buffer size*; the number of seconds of video stored in the client buffer ready for playback.
3) *Average connection throughput*; the average throughput of the connection over the last 1s of the connection.
4) *Number of quality switches*; the number of times during the connections lifetime that the quality of the video is adjusted from one bitrate to another.
5) *Rebuffering time*; time spent waiting as the result of an empty playout buffer during a streaming session.
6) *Network Efficiency*; what percentage of the available network bandwidth is used.

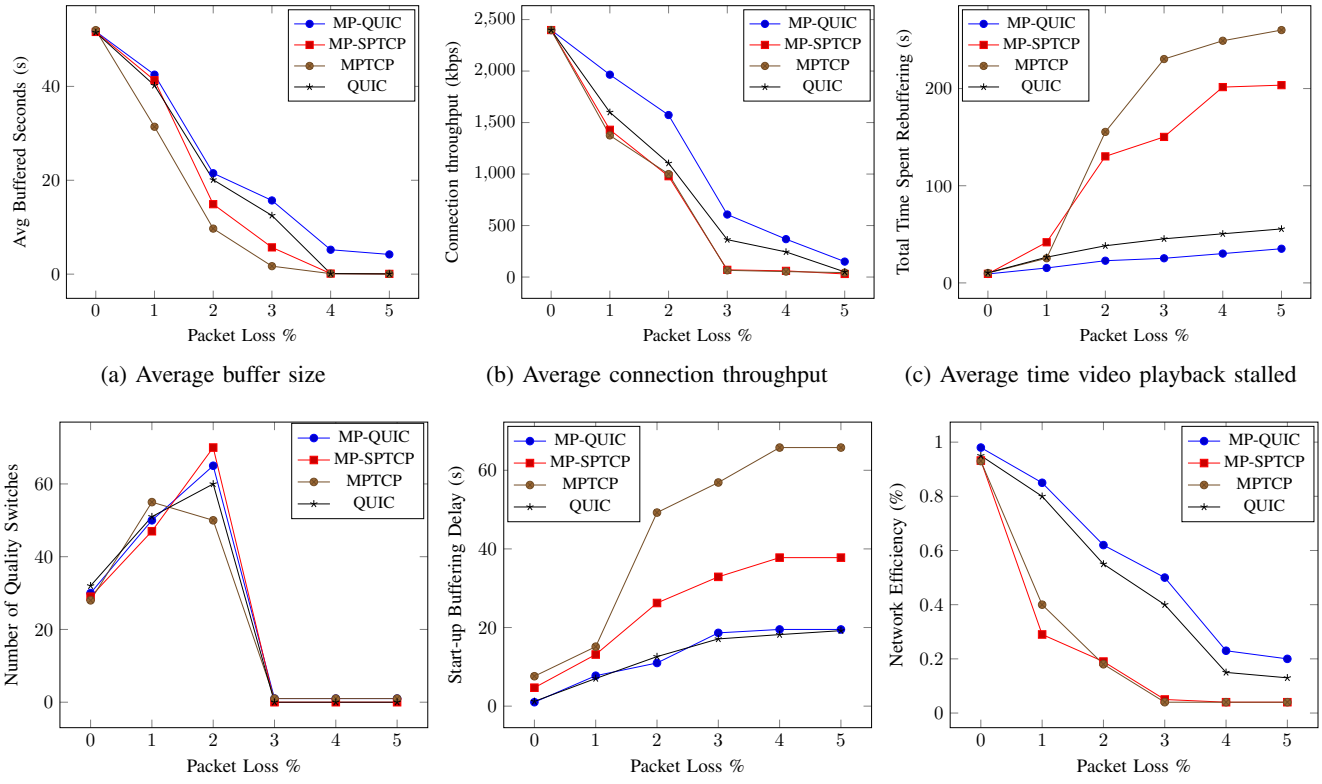
Fig. 3, shows the SDN application running alongside the Google VR Viewer which queries the SDN controller in the network to retrieve the average RTT delay along each path available. It interfaces with the Floodlight controller in the Mininet SDN network used for experimentation.

After requesting the web page and receiving a link to the MPD, the client requests the MPD from the DASH server. The MPD describes the bitrate's data structure available for download. Based on information contained within the MPD, the client then requests SegmentBase *index segments* for each available bitrate. *Index segments* provide critical information related to time, byte ranges, and stream access points (used for seeking within a video) for corresponding bitrate media segments.

The architecture of our testbed, shown in Fig. 4, supports QUIC version 38, MPTCP v0.91.0 and TCP. The testbed consists of four nodes and a Floodlight Mininet controller; the nodes are a client, a server, and two traffic shapers. All nodes have two network interface cards connected to links that are one gigabit capable. All nodes are running 64-bit Ubuntu 14.04 Linux. The TCP congestion control algorithm used was Cubic and the MPTCP congestion control algorithm used wVegas. The traffic shaping nodes are individual Linux boxes that control all network related parameters such as RTT and packet loss rate with Linux Network Emulator (netem). Also, the traffic shapers control the maximum achievable throughput with Linux traffic control system (tc) and the hierarchical token bucket (htb), which is a classful queuing discipline (qdisc).

V. Evaluation

The plots below are the result of experimentation in our physical testbed. For each plot, the packet loss rate was set at the discrete points of 0%, 1%, 2%, 3%, 4% and 5%. Each data point represents the average over a series of 20 runs at each configuration. MP-QUIC refers to our solution presented in the figures. MP-SPTCP refers to our implementation of the idea of using bandwidth thresholds presented by Nam et. al [12] instead of RTT and retransmission thresholds. Lastly, MPTCP and QUIC refer to their standard implementation without modification.



(a) Average buffer size

(b) Average connection throughput

(c) Average time video playback stalled

(d) Average number of quality switches

(e) Average start-up buffering delay

(f) Network usage efficiency

Fig. 5: Various performance metrics for a given packet loss rate percentage.

Shown in Fig. 5a, we compared the average number of seconds of video in the buffer vs. packet loss rate for each algorithm. This plot does not show the quality of the buffered seconds of video. Evaluating Fig. 5a with Fig. 5b when the packet loss rate is zero, each algorithm perform comparably. This is expected since each algorithm is able to operate with no packet loss. These results are consistent with previous studies [14]–[16]. As the packet loss rate increases, MP-QUIC and QUIC perform better when compared to MP-SPTCP or MPTCP. This is due to the fact that MP-QUIC performs more like a pure UDP based protocol and less like MPTCP when packet loss rates increase. MP-SPTCP still uses TCP which is known to not perform well in high packet loss environments. Further, MPTCP relies on timeouts when switching paths, unlike our modified version. Except for MP-QUIC and QUIC, when packet loss rates reached 5%, Fig. 5a and Fig. 5b illustrate that standard MPTCP and MP-SPTCP were unable to maintain usable bandwidth over the lifetime of the connection.

Illustrated in Fig. 5c, MP-QUIC and QUIC spend significantly less time stalled with an empty buffer than standard MPTCP and MP-SPTCP. Our protocol even outperforms canonical QUIC. This is attributed to the low latency of our protocol being leveraged quickly and effectively with our SDN controller providing timely network statistics. Additionally, our start-up decision engine made a significant difference when experiencing high packet loss rates when stalls were imminent, Fig. 5c.

As mentioned in the introduction, one factor affecting QoE is the size of the MPTCP reorder buffer and the number of retransmissions MPTCP has to make during a connection for unbalanced connections. In Fig. 5d, at 4% and 5% loss rates, MPTCP and MP-SPTCP are unable to make any quality switches and remain at the lowest quality level the entire duration of the experiment. Interestingly, at packet loss rates of 1% and 2%, MP-SPTCP has fewer quality switches than the others. We conclude that this is the case for a few reasons. One reason is that MPTCP is not able to successfully transmit as much data through the network and, as a result, tends to select the lowest bitrate consistently. In comparison, the others try to ramp up quality more often due to a higher bandwidth. As a result, the bitrate selection oscillates more frequently.

As shown in Fig. 5e, MP-QUIC, and QUIC with 0% packet loss are faster than MPTCP and MP-SPTCP while in the start-up phase of a video connection. Concluding from the analysis, this is a result of the overhead of MPTCP and MP-SPTCP establishing a full mesh of connections between the client and server. However, this figure does not present the quality of the video right after start-up. MPTCP and MP-SCTCP at a 0% packet loss starts up slightly slower than MP-QUIC and QUIC, but with a higher bitrate. At high packet loss rates, MP-QUIC and

QUIC begin playback more quickly and with a higher bitrate than traditional MPTCP and MP-SPTCP. Finally, Fig. 5f reveals that MP-QUIC and QUIC utilize more of the available bandwidth than the other configurations. MP-QUIC outperforms QUIC in high-loss networks, as well as high bandwidth low-loss networks.

VI. CONCLUSION

Proposed in this paper are techniques for improving omnidirectional ABR video performance using MPTCP/QUIC over an SDN architecture. We focused on improving the start-up experience and overall streaming by designing two strategies to leverage SDN. The use of SDN enabled us to design an algorithm to actively switch between MPTCP networks and QUIC. By using MPTCP/QUIC over SDN, we were able to leverage path diversity for increased performance in low-loss networks and perform well in high-loss networks. As a result, the start-up buffering delay decreased by 60% or more when using MP-QUIC as compared to MPTCP or MP-SPTCP when packet loss rates were above 2%. Playback stall time also decreased by more than 10% when packet loss rates were above 5%. Furthermore, MP-QUIC utilized the network 9% more efficiently with packet loss rates above 0%. These results demonstrate that our algorithm outperforms standard QUIC, MPTCP and MP-SPTCP in increasingly lossy networks by 15% or more in many key streaming metrics. These preliminary results were for a controlled proof of concept and may vary with multiple users.

Limitations to our work include the scale of our experimentation. We look to expand this study using a larger testbed with Planet Lab. We also plan to examine other MPTCP path management algorithms to further compare our work. Finally, we plan to obtain results under wireless conditions and where there more than two networks available.

REFERENCES

[1] "Cisco visual networking index: Forecast and methodology, 2014-2019 white paper," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html, accessed: 2017-04-01.

[2] "Apple http live streaming," https://developer.apple.com/resources/http-streaming, accessed: 2017-04-01.

[3] T. Stockhammer, "Dynamic adaptive streaming over http –: Standards and design principles," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 133–144. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943572

[4] "Google vr view," https://github.com/googlevr/vrview, accessed: 2017-04-1.

[5] R. Hamilton, J. Iyengar, I. Swett, A. Wilk, and Google, "Quic: A udp-based secure and reliable transport for http/2," draft-hamilton-early-deployment-quic-00, Internet Engineering Task Force, jul 2016. [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00

[6] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," RFC 6824, Internet Engineering Task Force, jan 2013. [Online]. Available: https://tools.ietf.org/html/rfc6824

[7] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: An experimental investigation of quic," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. New York, NY, USA: ACM, 2015, pp. 609–614. [Online]. Available: http://doi.acm.org/10.1145/2695664.2695706

[8] P. Biswal and O. Gnawali, "Does quic make the web faster?" in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

[9] B. Sonkoly, F. Nmeth, L. Csikor, L. Gulys, and A. Gulys, "Sdn based testbeds for evaluating and promoting multipath tcp," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 3044–3050.

[10] S. Tariq and M. Bassiouni, "Qamo-sdn: Qos aware multipath tcp for software defined optical networks," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, Jan 2015, pp. 485–491.

[11] M. Sandri, A. Silva, L. A. Rocha, and F. L. Verdi, "On the benefits of using multipath tcp and openflow in shared bottlenecks," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, March 2015, pp. 9–16.

[12] H. Nam, D. Calin, and H. Schulzrinne, "Towards dynamic mptcp path control using sdn," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 286–294.

[13] "A quic update on googles experimental transport." http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html, accessed: 2017-04-01.

[14] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath tcp performance over wireless networks," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: ACM, 2013, pp. 455–468. [Online]. Available: http://doi.acm.org/10.1145/2504730.2504751

[15] Y. P. G. Chowrikoppalu, "Multipath adaptive video streaming over multipath tcp," Ph.D. dissertation, Intel, 2013.

[16] Q. D. Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "A first analysis of multipath tcp on smartphones," in *17th International Passive and Active Measurements Conference*, vol. 17. Springer, March-April 2016.

[17] B. Hayes, Y. Chang, and G. Riley, "Adaptive bitrate video delivery using http/2 over mptcp architecture," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, June 2017, pp. 68–73.

[18] S. Bae, D. Jang, and K. Park, "Why is http adaptive streaming so hard?" in *Proceedings of the 6th Asia-Pacific Workshop on Systems*, ser. APSys '15. New York, NY, USA: ACM, 2015, pp. 12:1–12:8. [Online]. Available: http://doi.acm.org/10.1145/2797022.2797031

[19] Y. C. Chen and D. Towsley, "On bufferbloat and delay analysis of multipath tcp in wireless networks," in *2014 IFIP Networking Conference*, June 2014, pp. 1–9.

[20] "Google cardboard camera app," http://storage.googleapis.com/cardboard-camera-converter/index.html, accessed: 2017-04-01.

[21] "x264," http://www.videolan.org/developers/x264.html, accessed: 2017-04-01.

[22] S. Wei and V. Swaminathan, "Low latency live video streaming over http 2.0," in *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, ser. NOSSDAV '14. New York, NY, USA: ACM, 2014, pp. 37:37–37:42. [Online]. Available: http://doi.acm.org/10.1145/2578260.2578277

[23] M. V. J. Heikkinen and A. W. Berger, "Comparison of user traffic characteristics on mobile-access versus fixed-access networks," in *Proceedings of the 13th International Conference on Passive and Active Measurement*, ser. PAM'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 32–41. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28537-0_4