

**EFFICIENT DELIVERY OF MULTIMEDIA CONTENT OVER MULTIPATH  
NETWORKS**

A Dissertation  
Presented to  
The Academic Faculty

By

Brian D'Angelo Hayes

In Partial Fulfillment  
of the Requirements for the Degree  
Doctor of Philosophy in the  
School of Electrical and Computer Engineering

Georgia Institute of Technology

August 2018

Copyright © Brian D'Angelo Hayes 2018

# **EFFICIENT DELIVERY OF MULTIMEDIA CONTENT OVER MULTIPATH NETWORKS**

Approved by:

Professor George F. Riley  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Professor Yusun Chang  
School of Electrical Engineering  
*Kennesaw State University*

Professor Raheem Beyah  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Professor John Copeland  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Professor Douglas M. Blough  
School of Electrical and Computer  
Engineering  
*Georgia Institute of Technology*

Professor Jun Xu  
School of Computer Science  
*Georgia Institute of Technology*

Date Approved: July 09, 2018

You can remember me and only that I am gone, or you can cherish my memory and let it  
live on.

*Jesse C. Hayes Jr.*

I dedicate this work to my daughter whom I have yet to meet. I hope to live up to the standard my father (your grandfather) set, that you, unfortunately, will not get a chance to meet in person. You are deeply loved by the Hayes, Campbell, Bonner, and Younger families. No matter what happens in life, me and your mother Kathleen will always love you more than you could ever imagine. As I reflect on pictures of my own father and what he meant to me, I hope to make you just as proud. Your grandmother (Hazel) and great-grandmother (Fannie) (my mother and grandmother) whom you will meet shortly, are unbelievably excited to meet and spoil you. They are the best mother and grandmother

I could have ever prayed for. They have been the rock of the family. I love them unbelievably so, and I am sure you will too. My sister, Sierra (Cece), soon to be Aunt is just amazing. She often says she “looks up to me”, her older brother, but in so many ways I have looked up to her for her strength and compassionate heart. Now for the “big girl on the insides” sweet tooth side of your family (ask your mom about that), the Bonners. They are the best family I could ever know; despite me being scared at first meeting Daddy Bonner and seeing Buster a 50-cal revolver in rural Oregon (note to self for later years). I am truly thankful for them and you will soon get to meet “the claw,” and know more about the military, airplanes, and history than any president. If you end up finding bacon, steaks, ice cream, chocolate, or cake irresistible, we’ll know where that came from. I could go on forever, but I need to finish this dissertation before your mom cattle prods me again :)

With all the love in the world. B Hayes.

## **ACKNOWLEDGEMENTS**

The author would like to thank so many people that surely, a few names will be missed. I would like to start with my wife Kathleen Hayes whom without her support, I would not be at this point literally. She has helped energize and direct me when I was academically drowning without a lifeline. I would also like to thank Hazel Hayes, Jesse Hayes Jr., Sierra Hayes, and Fannie Younger Smalls who have been with me since I began this journey. I hope I've made you all proud and have been a good return on your ample investments of love. Thank you, Tonya, Rashad, Faith and Shayla Harris, you all have been a joy to be around. I want to continue by thanking my extended family Aunt BJ, Uncle Wayne, Uncle John, Little John, Carla, Tae, Regina, and William you all have been there rooting for me and I will never forget it. Thank you, Rosemary Parker, from UMCP in the Center for Minorities in Science in Engineering Program for believing in me. Thank you Abiodun Osho, Ifiok Obot, Aniekan Obot, Lucas Richardson, and the SALSAtlanta Family. The list could go on and on, but I would be remiss if I did not thank my Lord and Savior Jesus Christ.

## TABLE OF CONTENTS

<b>Acknowledgments</b> . . . . .	v
<b>List of Figures</b> . . . . .	ix
<b>Summary</b> . . . . .	xii
<b>Chapter 1: Introduction</b> . . . . .	1
1.1 Multimedia Streaming . . . . .	2
1.1.1 Adaptive Bitrate Streaming . . . . .	3
1.2 Multipath Networking . . . . .	6
1.3 Contribution . . . . .	7
1.4 Organization . . . . .	8
<b>Chapter 2: Background</b> . . . . .	10
2.1 Multimedia Streaming Protocols . . . . .	10
2.1.1 Dynamic Adaptive Streaming over HTTP (DASH) . . . . .	12
2.2 Transport Protocols . . . . .	16
2.2.1 Multipath TCP (MPTCP) . . . . .	16
2.2.2 Quick UDP Internet Connections (QUIC) . . . . .	19
2.3 Software Defined Networking (SDN) . . . . .	22

2.4	Reinforcement Learning (RL) for Adaptive Streaming . . . . .	24
<b>Chapter 3: Extending DASH with MPTCP Server-Push . . . . .</b>		<b>26</b>
3.1	Start-up Accelerator . . . . .	27
3.2	Zero Stall Playback . . . . .	29
3.3	Hybrid Push/Pull . . . . .	30
3.4	Experiment Setup . . . . .	31
3.5	Testbed . . . . .	33
3.6	Evaluation . . . . .	34
3.7	Discussion . . . . .	39
<b>Chapter 4: Flexible Transport Layer for High Bandwidth Content . . . . .</b>		<b>41</b>
4.1	Connection Decision Engine . . . . .	42
4.2	Playback Network Monitoring . . . . .	44
4.3	Testbed . . . . .	46
4.4	Evaluation . . . . .	48
4.5	Discussion . . . . .	53
<b>Chapter 5: Dynamic Transport Layer Selection at Scale using SDN . . . . .</b>		<b>55</b>
5.1	System Model . . . . .	56
5.1.1	Differential Delay Minimization . . . . .	56
5.1.2	Flow Allocation . . . . .	58
5.2	Experiment Setup . . . . .	59
5.3	Evaluation . . . . .	61

5.4	Discussion . . . . .	66
<b>Chapter 6: Reinforcement Learning Adaptive Transport in Multipath Networks</b>		<b>68</b>
6.1	System Design . . . . .	69
6.1.1	Training Procedure . . . . .	70
6.1.2	SAND Assisted Controlled Unfairness . . . . .	72
6.2	Experiment Setup . . . . .	75
6.3	Evaluation . . . . .	77
6.4	Discussion . . . . .	83
<b>Chapter 7: Reinforcement Learning Adaptive Content Delivery Networks (CDN)</b>		<b>84</b>
7.1	Architecture Design . . . . .	86
7.1.1	System Design . . . . .	86
7.2	Experiment Setup . . . . .	90
7.3	Evaluation . . . . .	92
7.4	Discussion . . . . .	98
<b>Chapter 8: Conclusion and Future Work</b>		<b>100</b>
8.1	Conclusion . . . . .	100
8.2	Future Work . . . . .	103
<b>References</b>		<b>109</b>
<b>Vita</b>		<b>110</b>



## LIST OF FIGURES

1.1	Adaptive multimedia streaming flow diagram example. . . . .	4
1.2	MPTCP connection establishment overview. . . . .	7
2.1	High-level UDP functionality . . . . .	11
2.2	High-level TCP functionality . . . . .	12
2.3	XML manifest description example . . . . .	13
2.4	High-level MPTCP functionality . . . . .	17
2.5	High-level transport framing . . . . .	20
2.6	Software defined network layout . . . . .	23
3.1	Example adaptive bitrate streaming flow diagram . . . . .	27
3.2	DASH start-up flow used during experimentation . . . . .	29
3.3	Experimental lab testbed setup for HTTP/2-based experiment . . . . .	34
3.4	Various performance metrics for a given packet loss rate percentage. . . . .	35
3.5	Various performance metrics for a given packet loss rate percentage. . . . .	36
3.6	Various performance metrics for a given packet loss rate percentage. . . . .	37
4.1	MPTCP/QUIC-enabled ABR network. . . . .	42
4.2	QUIC vs. TCP with TLS connection establishment. . . . .	43
4.3	MPTCP/QUIC-enabled client used with an SDN. . . . .	45

4.4	MPTCP/QUIC-enabled client setup used with an SDN network. . . . .	47
4.5	Various performance metrics for a given packet loss rate percentage. . . . .	50
4.6	Various performance metrics for a given packet loss rate percentage. . . . .	51
4.7	Various performance metrics for a given packet loss rate percentage. . . . .	52
5.1	Example adaptive transport flow diagram . . . . .	56
5.2	Network topology . . . . .	59
5.3	Various performance metrics for a given number of users. . . . .	62
5.4	Various performance metrics for a given number of users. . . . .	63
5.5	Various performance metrics for a given number of users. . . . .	64
6.1	DASH OMAF architecture . . . . .	70
6.2	Example SAND architecture . . . . .	73
6.3	Implemented multipath SAND testbed . . . . .	76
6.4	Various performance metrics for a given number of users. . . . .	78
6.5	Various performance metrics for a given number of users. . . . .	79
6.6	Various performance metrics for a given number of users. . . . .	80
6.7	Various performance metrics for a given number of users. . . . .	81
6.8	Various performance metrics for a given number of users. . . . .	82
7.1	Example 360 VR video field of view mapping . . . . .	85
7.2	Flow diagram for neural network-based CDN architecture . . . . .	88
7.3	High-level simulated network topology . . . . .	91
7.4	Various performance metrics for a given number of streams. . . . .	93

7.5	Various performance metrics for a given number of streams. . . . .	94
7.6	Various performance metrics for a given number of streams. . . . .	95
7.7	Various performance metrics for a given number of streams. . . . .	96
7.8	Various performance metrics for a given number of streams. . . . .	97

## **SUMMARY**

This work proposed a unique approach to efficiently deliver high-quality multimedia data by leveraging concepts from various multipath networking models in conjunction with ABR protocols that supplement the reliance on client-side network bandwidth estimation and/or buffer-state. Prior research into the development and evaluation of the performance of multipath-enabled multimedia streaming techniques are presented. Machine learning based multipath-enabled techniques are designed in the development of selective transport protocols for streaming applications and validating their effectiveness.

# **CHAPTER 1**

## **INTRODUCTION**

The use of adaptive bitrate streaming (ABR) services has significantly increased in recent years. Prior to ABR, transmitting multimedia content was more-or-less a one-size-fits-all experience in which a single resolution video was transmitted to viewers watching on widely different screen sizes. As a result, content was often distorted. ABR was developed as a performance management technique allowing content providers to target device specific resolutions while simultaneously adapting to the speed of client's Internet connections. The flexibility and ease of ABR deployment was quickly realized by academics and content providers leading to the deployment of many unique proprietary implementations, such as Microsoft's Smooth Streaming platform [1] or Adobe HTTP Dynamic Streaming (HDS) [2].

As the expectation for ubiquitous high-quality multimedia content increased, so did the need for interoperable non-vendor-specific transmission standards. This led to the deployment of ABR frameworks HTTP Live Streaming (HLS) [3] and MPEG Dynamic Adaptive Streaming over HTTP (DASH also known as MPEG-DASH) [4], allowing for transmission over broad distribution networks. Currently however, network requirements for complex multimedia applications including 360-degree video and ultra-high-definition streaming are outpacing infrastructure capacity [5].

High bandwidth wireless technologies (e.g., LTE and 5G) have significantly improved network performance, however, timeliness and infrastructure costs required to upgrade these systems can significantly reduce available options. Many devices today have access to multiple networks, but do not have a mechanism to pool resources. Furthermore, commonly used ABR implementations often rely on the estimation of network bandwidth and/or buffer-state client-side. This can be challenging and can lead to inaccurate estima-

tions and poor network utilization as a result of inflexible transport protocols.

The objective of this research is to develop a unique approach to efficiently deliver high-quality multimedia data by leveraging concepts from various multipath networking models in conjunction with ABR protocols that supplement the reliance on client-side network bandwidth estimation and/or buffer-state. Machine learning based multipath-enabled techniques are implemented in the development of selective transport protocols for streaming applications and validating their effectiveness. Prior research into the development and evaluation of multipath-enabled multimedia streaming techniques are extensively evaluated and highlight the context in which this work is presented.

## **1.1 Multimedia Streaming**

The term ‘multimedia’ generally refers to the mixture of audio, video, and text information into a singular format. ‘Multimedia streaming’ refers to the transmission of this content. The focus of this research is the transmission of multimedia data over the Internet. Given the diversity of multimedia content, transport approaches developed by major content providers varies and are often not interoperable with one another. Consequently, the classification of streaming architectures are largely based around a few features, namely; content type, transport layer protocols, and adaptation algorithms.

Production level streaming architectures are designed to transmit compressed data as uncompressed content is too large for operation over the Internet at a reasonable scale. In preceding sections are example streaming architectures, in which content is compressed with an encoder and packetized prior to transmission over the network. Once the content has arrived at the receiver, the recombined compressed data is decoded for playback. Major components incorporated into the media streaming experience include a server, client, physical networks, encoder, and application protocols.

### 1.1.1 Adaptive Bitrate Streaming

ABR provides the ability to stream multimedia content in varying network conditions to a wide range of devices. Fig. 1.1 illustrates a high-level flow diagram with sample numbers demonstrating how ABR streaming can be implemented from production to delivery. ABR streaming works by segmenting raw media streams into a series of a few seconds of content called a *chunk*. Each set of bitrates stored on an HTTP server is encoded into *chunks*. The client video player then downloads a *chunk* for playback at a bitrate it can support. This allows players to switch between higher or lower bitrates during a stream based on a player's network and buffer conditions.

Popular ABR multimedia protocols include closed source HTTP Live Streaming (HLS) [3] and open sourced MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [4]. The DASH Industry Forum (DASH-IF) established guidelines for MPEG-DASH usage supported by Netflix, Microsoft, Google, Ericsson, Samsung, Adobe, and others. In addition, DASH-IF has released a open-source continuously updated DASH-IF Reference Client [6] used in this work.

#### *Adaptive Bitrate Algorithms*

Maximizing the quality of experience (QoE), ABR algorithms have to simultaneously optimize the video quality that the system can support while minimizing rebuffering and bitrate switches. Three main algorithmic approaches are commonly used to maintain this goal: network capacity-based adaption, buffer-state adaption, and a hybrid approach. Network-based approaches [7, 8] are highly dependent on accurate throughput estimation, however approximating future bandwidth within a short time window can be difficult due to quickly fluctuating networks. A buffer-state adaption solution was introduced to bypass the reliance on bandwidth estimation [9]. With buffer-state adaptation, buffer levels are monitored which indirectly implies the available bandwidth. However, buffer-state adaptation is not useful when the buffer is empty. Hybrid-based algorithms [9, 10, 11] attempt to take

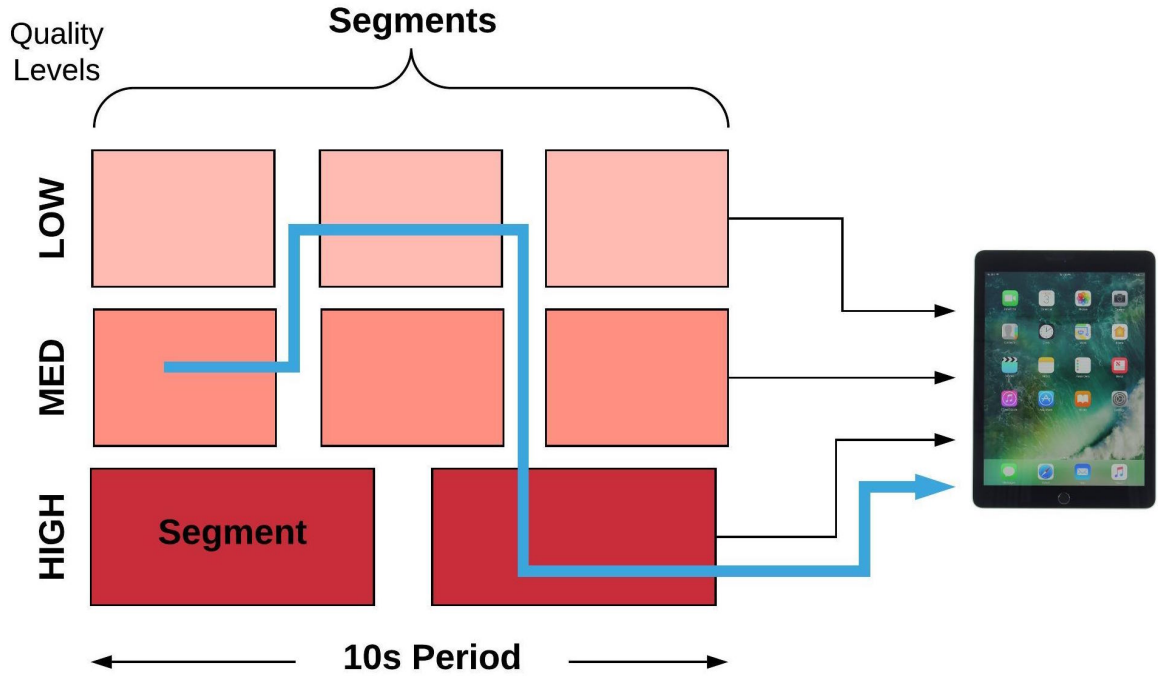


Figure 1.1: Adaptive multimedia streaming flow diagram example.

the best of both classes of algorithms, but can be complex and resource intensive.

Client-side ABR algorithms are challenged with achieving a high QoE when contending with a diverse set of devices and network conditions. 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 [12]. Additionally, ABR algorithms struggle to achieve a steady high-quality stream in unpredictable networks because of the suboptimal bandwidth estimation being linked to chunk duration and inaccurate bitrate decisions [12].

Multipath networking could have the potential to address many of the concerns listed above. A flexible transport protocol in conjunction with ABR algorithms can simultaneously optimize video quality while minimizing multipath differential delay. Through preliminary work, a framework was developed that actively manages multipath connections for streaming video in a large network demonstrating the potential of addressing these issues.



## *Emerging Technologies*

The application of reinforcement learning (RL) to ABR streaming is a burgeoning area of research [13] and techniques focused on improving the QoE have had encouraging results. Previous streaming efforts having been proven to be suboptimal and rely on premade client-side models for bitrate adaption [12]. Conversely, RL allows clients to learn an optimal streaming policy over time without having prior assumptions about the streaming experience.

RL has been proven to be efficient with complex multi-factor problems using Markov Decision Processes (MDP). MDP is a mathematically-based technique for modeling decision making in environments where the outcome is effected jointly by randomness and actions made by a decision maker. In general there are two classes of RL-based algorithms; online, and offline. Online techniques are challenged with making future models in nearly real-time as quickly as possible. This allows for immediate distribution having the advantage of quickly converging solutions. Consequently, results from this method are generally fast, but may not be very accurate in the short term due to not having the benefit of prior training periods. Conversely, offline methods have the benefit of being highly accurate if trained with models that are highly reflective of the environment in which the models are deployed. Disadvantages of offline training are that it may take many hours, days, or weeks to train models that are highly accurate. There is also the disadvantage of not being able to adapt in real-time to traffic changes if the network conditions are wildly different than the traffic in which it was trained. This is can happen in unexpected situations, for example, viral content or when reacting in real-time is the preferred method and would be more responsive.

Another new area of research that has impacted this field is leveraging MPEG's Server and Network Assisted DASH (SAND) [14]. SAND provides unique mechanisms for providing context-aware quality of service (QoS) and experience signaling which is utilized to operate in multipath networks. SAND's proposed core set of QoS parameters are capable

of being checked and shared by data content servers, network infrastructure nodes, DASH clients and content delivery networks (CDN) with the goal of augmenting ABR decision making. SAND makes use of the WebSocket protocol to pass messages among network nodes which provide challenges for CDNs due to limitations of the WebSocket protocol requiring persistent network connections. This has led to an alternate solution chaining MPDs together in a dynamic playlist providing flexibility for content aggregators located anywhere in the network, all with the goal of providing flexible and personalized experiences.

## **1.2 Multipath Networking**

Multipath networking, as the name implies, is a technique of using alternate network paths to transmit data. This idea provides a multitude of capabilities which include tolerating network faults, improved overall capacity, and enhanced security by not sending all data for a connection using a single provider. The challenge of multipath networking comes in the joint management phase of multiple paths, with data streaming from different applications with varying requirements. The ability to keep track of and manage this complex interface is a separate area of research itself. This work focuses more narrowly on transmitting multimedia data and its effects on playback and QoE metrics. Taking advantage of multipath characteristics in ABR streaming over Transmission Control Protocol (TCP) has been explored in the past and has been shown to improve video streaming performance by utilizing several TCP connections simultaneously [7, 15, 16].

QUIC version 38 [17] has been implemented over User Datagram Protocol (UDP) to address many of TCP limitations, one of which includes placing control into the application space at both endpoints rather than kernel space. Detailed illustrations of QUIC's connection establishment compared to TCP with TLS are shown in Chapter 2. Of particular interest is QUIC's reduced latency (e.g., 0-round-trip time (RTT) connectivity overhead) and stream-multiplexing support. Today's mobile devices have multiple network inter-

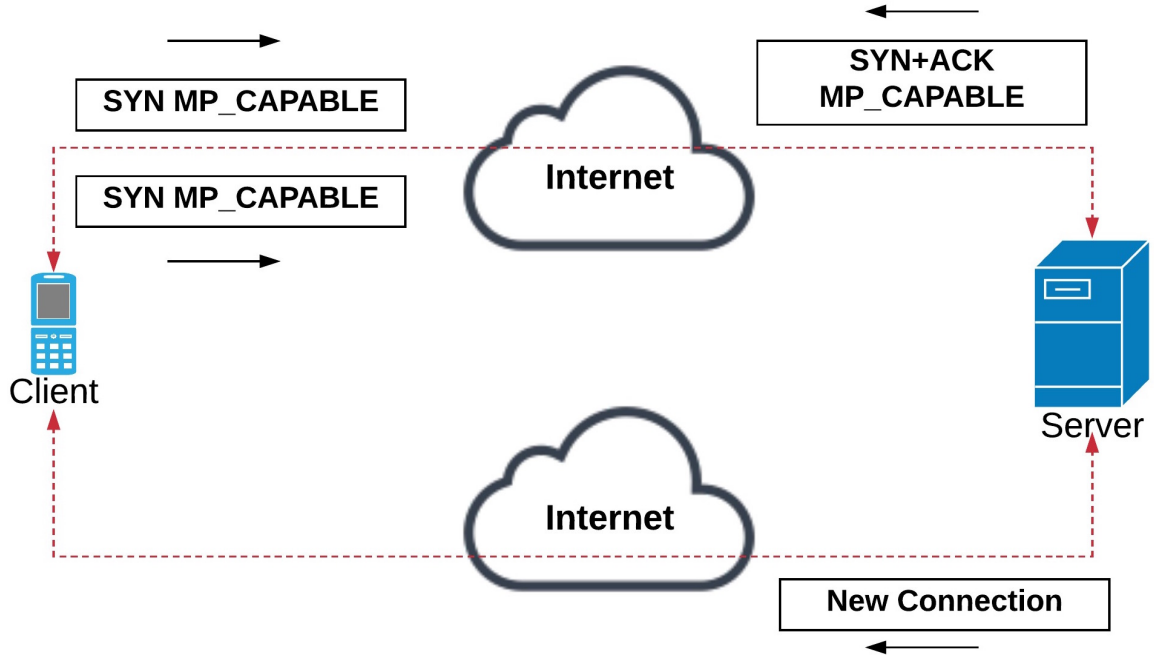


Figure 1.2: MPTCP connection establishment overview.

faces, such as Wi-Fi and LTE, but they are not leveraged simultaneously to pool available resources from existing networks in practice.

MPTCP is an experimental standard of the Internet Engineering Task Force (IETF). It is also a backward compatible extension to TCP (see Fig. 1.2). 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. Considering multipath networking performance potential, an evaluation of existing multimedia transport protocols in this new environment logically follows as an area of interest.

### 1.3 Contribution

The focus of this dissertation is to provide original solutions to efficiently and reliably deliver high-quality multimedia data while leveraging multipath networks with innovative ABR controls. The contributions of this work are multifaceted.

Initially presented is a comprehensive review and analysis of notable existing adaptive

bitrate streaming and multipath networking techniques. Similar approaches are brought forward to discuss the limitations and strengths of each class.

Following the background literature is a proposed approach to extending DASH with HTTP/2 server-push to MPTCP. This approach suggests the use of modular HTTP/2 server push-based strategies leveraging MPTCP to remove client-side estimation of network bandwidth and buffer-state with the goal of improving start-up delay and overall performance. Recognizing several limitations of native MPTCP, this research proposes leveraging MPTCP and QUIC by way of a software-defined network (SDN) environment. The SDN network allows for the seamless transition from TCP to UDP-based transport protocols while establishing streaming sessions in a small testbed environment.

Continuing the work on SDN performance improvements at a small scale led to the discussion and implementation of large scale simulation of competing streaming sessions. Benchmark parameters were set and the implications for users and content providers were documented before and after optimizations were made.

Presented next is a unique architecture for machine learning-based multipath-enabled techniques implemented for the development of selective transport protocols for streaming applications while validating their effectiveness. Incorporating network assistance from SAND and SDN into driving the streaming experience is investigated thoroughly. Finally, building on this general performance architecture, a framework for leveraging existing CDNs to provide video specific customizable streaming experiences are presented.

## **1.4 Organization**

The dissertation is organized as follows: Chapter 1 provides an introduction and explains contribution goals of this work. A discussion of existing literature and techniques are presented in Chapter 2. In Chapter 3, a technique for extending and improving DASH with HTTP/2 push in MPTCP is outlined and its relationship to the remaining research is discussed. Chapter 4 presents details of a flexible transport technique for high bandwidth

video content. Chapter 5 builds upon the initial testing of the previous chapter by expanding the network to multiple competing users. Chapter 6 considers the lessons learned from the previous chapters and results in a shift by developing a learning adaptive transport selection framework for use in multipath networks. Chapter 7 expands upon the previous chapters, but focuses more heavily on customizable streaming experiences for individual videos using CDNs and small neural networks instead of the generalized system developed previously. Lastly, Chapter 8 is dedicated to drawing conclusions from this research and providing future direction to further develop this work.

## **CHAPTER 2**

### **BACKGROUND**

This chapter discusses research related to multimedia streaming, multipath transport protocols, learning adaptive transport and SDN techniques as it relates to this research. In particular, the first section defines multimedia streaming and continues to further explain ABR and associated algorithms. The next section examines multipath networking techniques and the relationship to multimedia streaming. The final two sections provide the framework and outline current learning adaptive transport techniques and SDN-based multimedia transport.

#### **2.1 Multimedia Streaming Protocols**

The ability to support and transmit multimedia content over the Internet has developed as existing Internet protocols have matured. As a result, creative ways for making existing technologies compatible with modern multimedia requirements have been developed. TCP and UDP Internet transport options were each developed with the corresponding protocols HyperText Transfer Protocol (HTTP) [18] and Real Time Protocol (RTP) [19] respectively at the application layer.

RTP was developed specifically for real-time end-to-end audio and visual applications such as Web Real-Time Communication (WebRTC). This is a popular choice for live streaming events to a large number of users or for teleconferencing. As mentioned above, RTP uses UDP, which is applied primarily when timeliness is a higher priority than reliability. The major drawback to RTP-based applications is the fact that they traditionally operate using UDP. UDP is a best-effort connectionless protocol and within managed networks, generally functions as designed in Fig. 2.1. To compensate for the limitations of UDP, RTP is used with the RTP Control Protocol (RTCP) to monitor the performance

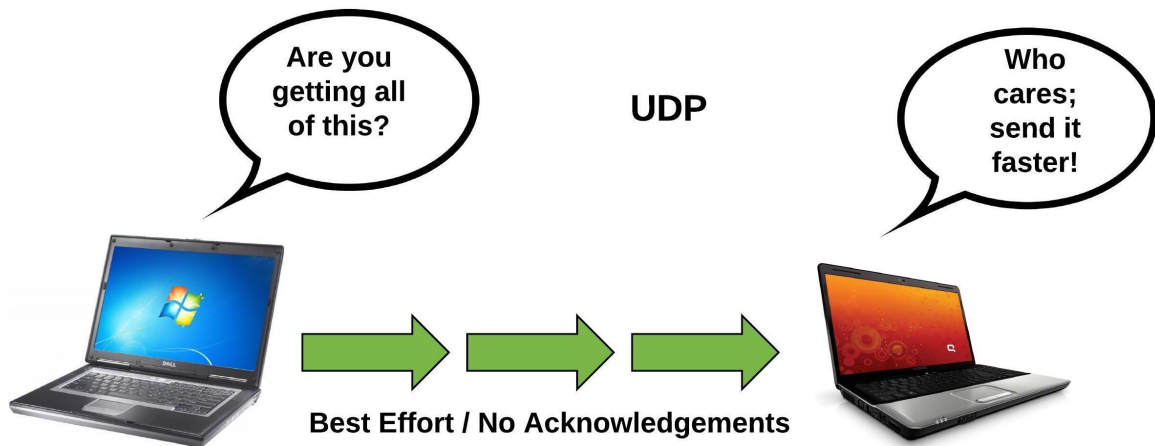


Figure 2.1: High-level UDP functionality

of an RTP stream. However, outside of managed networks across the Internet, there are many systems and firewalls that do not accept unsolicited UDP packets for security and other management reasons. Furthermore, UDP generally has no notion of congestion control or fairness in which UDP packets can easily overwhelm networks resulting in poor performance. There have been many revisions and other variants to the protocol over time supporting features such as newer audio/video codecs, and secure transmission. The details of these variants are outside the scope of this work and are not widely deployed.

HTTP was developed for delivering data on the Internet and relies on reliable transport protocols. During this time, TCP was used as its primary means of transport and is still widely used today because it supports reliable transport. TCP guarantees the delivery of all data and presents the data in the order it was sent to applications (see Fig. 2.2). TCP has built-in network congestion control mechanisms and has become the de-facto standard for HTTP delivery and many other protocols. HTTP scales well because it operates as a query/response protocol. This allows servers in the case of websites to service many users at once, which is more efficient. Furthermore, it is well supported by almost every client and server on the Internet resulting in most security policies handling TCP traffic properly in all types of networks, even unmanaged ones in the field. Similar to RTP, there have been a few major versions updated to the HTTP protocol including end-to-end en-

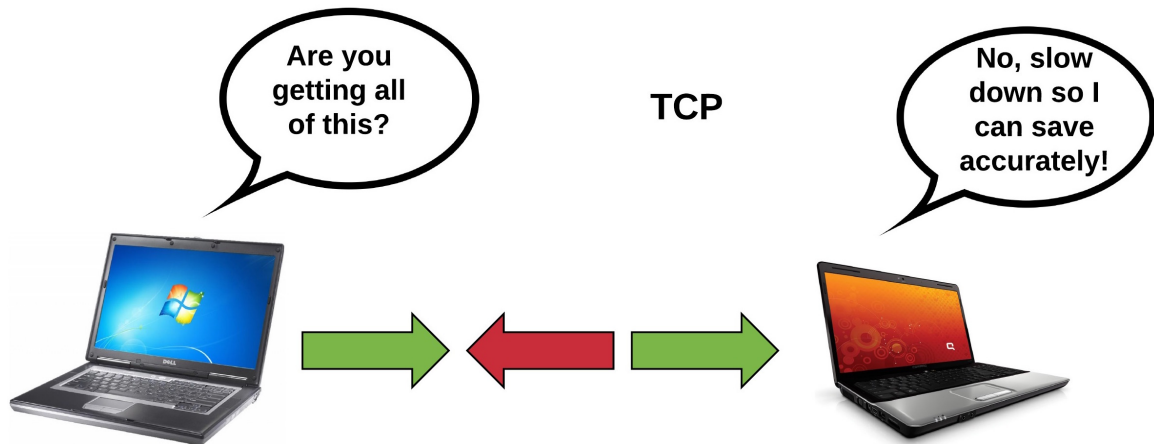


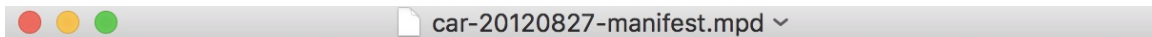
Figure 2.2: High-level TCP functionality

encryption capabilities. One notable version of HTTP is the latest version; HTTP/2 [20], that was implemented to address many of the limitations of HTTP 1.0/1.1. These include lack of support for pipelining, header compression, and prioritization of requests in modern browsers. HTTP/2 allows for the multiplexing of multiple HTTP requests over a single connection, has eliminated the head-of-line blocking issues seen in earlier iterations of the protocol, and enables server push. HTTP/2 server push allows a server to push content to a client without explicitly requesting it. This work is particularly interested in expanding the server push feature across multiple networks to improve video performance in lossy networks. HTTP/2 is currently supported by all major web services and was developed based on work from Google's SPDY protocol, which is described in a later subsection.

### 2.1.1 Dynamic Adaptive Streaming over HTTP (DASH)

Given the ubiquity of support for HTTP across the Internet and the growing popularity of streaming live and on-demand multimedia content came the formation of HTTP-based ABR streaming. In contrast to limitations of UDP-based techniques following traditional HTTP traffic and firewall rules, HTTP-based protocols can use the standard port 80 without special considerations. As mentioned in Chapter 1, ABR streaming works by taking a raw media stream and segmenting it into a series of a few seconds of content called a *chunk*.





```

<MPD xmlns="urn:mpeg:DASH:schema:MPD:2011" mediaPresentationDuration="PT0H3M1.63S"
minBufferTime="PT1.5S" profiles="urn:mpeg:dash:profile:isoff-on-demand:2011"
type="static">
  <Period duration="PT0H3M1.63S" start="PT0S">
    <AdaptationSet>
      <ContentComponent contentType="video" id="1" />
      <Representation bandwidth="4190760" codecs="avc1.640028" height="1080" id="1"
mimeType="video/mp4" width="1920">
        <BaseURL>car-20120827-89.mp4</BaseURL>
        <SegmentBase indexRange="674-1149">
          <Initialization range="0-673" />
        </SegmentBase>
      </Representation>
      <Representation bandwidth="2073921" codecs="avc1.4d401f" height="720" id="2"
mimeType="video/mp4" width="1280">
        <BaseURL>car-20120827-88.mp4</BaseURL>
        <SegmentBase indexRange="708-1183">
          <Initialization range="0-707" />
        </SegmentBase>
      </Representation>
      <Representation bandwidth="869460" codecs="avc1.4d401e" height="480" id="3"
mimeType="video/mp4" width="854">
        <BaseURL>car-20120827-87.mp4</BaseURL>
        <SegmentBase indexRange="708-1183">
          <Initialization range="0-707" />
        </SegmentBase>
      </Representation>
      <Representation bandwidth="686521" codecs="avc1.4d401e" height="360" id="4"
mimeType="video/mp4" width="640">
        <BaseURL>car-20120827-86.mp4</BaseURL>
        <SegmentBase indexRange="708-1183">
          <Initialization range="0-707" />
        </SegmentBase>
      </Representation>
      <Representation bandwidth="264835" codecs="avc1.4d4015" height="240" id="5"

```

Figure 2.3: XML manifest description example

Each set of bitrates stored on an HTTP server is encoded into *chunks*. The client video player then downloads a *chunk* for playback at a bitrate it can support. This allows players to switch between higher or lower bitrates during a stream based on a player's network and buffer conditions. Fig. 2.3 shows a sample manifest file needed to adapt to varying bitrates throughout the lifetime of the streaming session.

DASH along with HLS [3] are widely deployed instances of HTTP-based ABR techniques. Apple is the developer and producer of the HLS protocol and uses it for all of its devices, however the operation of the protocol itself is largely proprietary. DASH, also known as MPEG-DASH [4] on the other hand is open-sourced and an international standard protocol. The DASH Industry Forum (DASH-IF) established guidelines for MPEG-DASH usage supported by Netflix, Microsoft, Google, Ericsson, Samsung, Adobe, and others. DASH

operates similarly to other ABR techniques, but differs from HLS in that *chunk* segments are usually two to four seconds in duration instead of the ten second segments commonly seen in HLS. DASH also does not require specific video codecs to be used. Because DASH is widely supported and operates similarly to HLS, it is used in this work. However, due to the similarities in each of the protocols, DASH could be replaced with HLS and would likely produce similar results.

Research assessing the performance of DASH has found promising results. One study aiming to improve the performance of multimedia content by designing a transport protocol involved the implementation of DASH over SPDY (precursor to HTTP/2). Mueller et al. [21] explored this concept and assessed the protocol’s overhead and performance finding that SPDY is robust against increasing RTT. Many attributes specific to video streaming however, such as buffering delays and video start-up delays, were not explored. Other research has examined the use of small segment duration that leverage server push to enable low latency live streaming [22, 23]. Even though video streaming was taken into consideration in these studies, its effectiveness in a multipath or lossy environment was minimally explored. Other researchers have attempted to use DASH over HTTP/2 with a proxy and SPDY to mimic the features and performance of HTTP/2 aiming to improve video loading performance [24]. These studies only considered a TCP link in a lossless environment. Furthermore, they relied on proxies which can artificially increase the start-up latency experienced by clients during experimentation due to the TLS requirement in HTTP/2.

Current ABR techniques rely on the traditional HTTP client/server model with bitrate decisions being made by the client and the server responding accordingly. This allows DASH clients to freely choose the quality for each *chunk* requested from the available resolutions with the goal of maximizing the QoE. Many systems in use today implement techniques that attempt to stabilize buffer levels to avoid rebuffering events. This notion intuitively makes sense, however this idea does not take into account other important QoE metrics and has been shown to be a suboptimal solution [12]. The structure of videos and

the effects that delay compression and errors have on QoE are generally ignored in many of today's simple streaming heuristics. HTTP/2 based techniques combined with the speed of modern networks can help compensate for these limitations.

Coming to a consensus on universal metrics for QoE is a research area that is constantly debated. However, there are a few metrics that researchers generally agree upon that correlate with QoE [8]. The first is bandwidth availability; high bandwidth implies a high bitrate indicating a higher QoE. This however, is not a linear affect; increases in bandwidth when the bandwidth is low positively affects QoE. Conversely, when the bandwidth is already high, increases in bandwidth marginally affect QoE. Buffer stalls are another metric that lower client QoE. More importantly however, is the frequency and duration of the stalls. Repeated stalls in a short amount of time seem to lower QoE the fastest. Start-up buffering times is another important metric. The affects of this however depends on the duration of the content. A longer buffering period at the beginning of a long movie for example is tolerable, but a long buffering period for short content like Instagram stories will cause clients to abandon early. Lastly, frequent oscillations of bitrate can be jarring, especially when there are large jumps in the bitrate ladder. There are three main approaches to balance these priorities: network capacity-based adaption, buffer-state adaption, and a hybrid approach. Network-based approaches [7] use past bandwidth measurements to predict future performance. The obvious limitation is what to do when there are no past bandwidth measurements. This is where buffer-state adaption solutions come into play by ignoring bandwidth measurements all together [9]. By focusing only on the buffer level, the client does not have to predict future network performance and is only concerned with maintaining a buffer threshold. One limitation with this system is clients could be stuck at lower resolutions until the system fills its buffer to a threshold. The last approach is to combine both buffer and network-based solutions into a hybrid algorithm [10, 11]. Hybrid algorithms balance out the weaknesses of each technique while leveraging their respective strengths, however this can quickly become complicated to implement.

MPEG-DASH and other traditional ABR techniques lack personalization, leverage inaccurate network models, and use rigid control rules leading to suboptimal performance. These drawbacks stem from their lack of network-wide knowledge and control. There is no consensus with media players about how they make bitrate control decisions. Oftentimes those decisions are in conflict with one another causing network inefficiencies. This work looks to fill that gap without the need of a managed network (i.e. SDN).

## **2.2 Transport Protocols**

### 2.2.1 Multipath TCP (MPTCP)

Modern ABR streaming protocols over TCP have become increasingly complex to overcome inflexible legacy transport protocols. 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. As discussed earlier, TCP-based protocols are the dominant and preferred transport protocol. Many content producers (i.e. Netflix and YouTube) use TCP exclusively to support their millions of users' multimedia needs. MPTCP [25] is an experimental standard of the Internet Engineering Task Force (IETF). It is a backward compatible extension to TCP that allows single TCP connection multiplexing over all available network interfaces without any changes to existing applications. 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. Fig. 2.4 illustrates an example MPTCP flow diagram with two endpoints attached to three separate networks of varying bandwidth. Fig. 2.5 compares the frame format of TCP, MPTCP and QUIC.

Transparent to application and network, MPTCP has the ability to bring together multiple networks with different IP addresses into a single logical TCP session. As far as the application is concerned, it sends its data down to the transport layer to be sent out reliably and in order using traditional TCP sockets. MPTCP, being an extension to TCP, adds sup-

plemental options to the TCP header that can be used to support MPTCP. Being transparent to the network, MPTCP can traverse Internet proxies, firewalls, and NATs like traditional TCP. All modifications to support MPTCP are at the kernel level. MPTCP provides many of the same features of TCP which include couple congestion control so that if there was a common bottleneck, MPTCP would take up no more than a single TCP flow at that bottleneck node. It also has the added benefit of increased throughput when some networks are under utilized.

For network operators, increased demand and bandwidth requirements creates challenges for streaming with many competing users. Advancements in wireless technologies (e.g., LTE and 5G) have significantly improved network performance in an effort to mitigate this issue. However, opportunity and infrastructure costs required to upgrade media systems can quickly limit options. Dynamically leveraging existing infrastructure is necessary to meet future network requirements. Estimates by Cisco [5] indicate that by 2019, 80% of consumer Internet traffic will be video; up from 64% in 2014. MPTCP has the potential to fill this void and pool available network resources. Potential benefits of multipath transport techniques include load balancing between different networks paths, increased connection resiliency, maximizing current network infrastructure, and increased mobility.

Taking advantage of multipath characteristics in ABR over TCP has been explored in the past [7, 15, 16] and could improve video streaming performance by utilizing several TCP connections simultaneously. These studies however, do not take into account the large

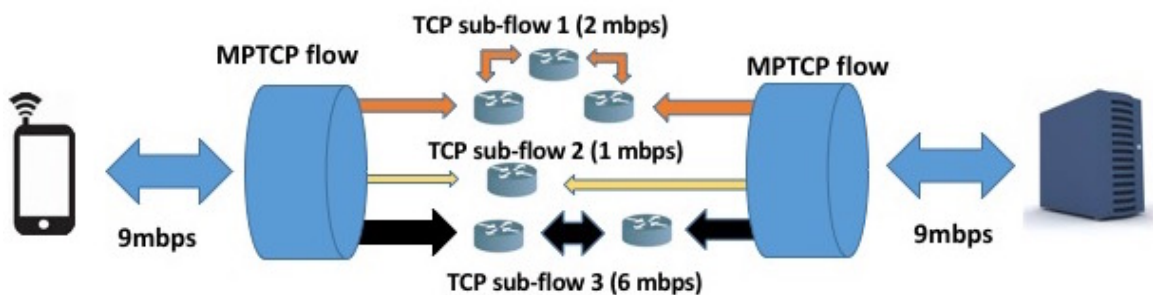


Figure 2.4: High-level MPTCP functionality

scale of content delivery networks and primarily rely on bandwidth estimation or buffer-state on the client-side with no feedback from the server. One study [26] found favorable results using multiple TCP links, but included the concept of investigating HTTP/2 server push with MPTCP concluding that MPTCP's large reorder buffer was problematic in low bandwidth networks. A recent study designed and implemented a proof of concept testbed with two users evaluating the performance of MPTCP in an SDN [27]. Findings from both of these studies did not consider operating at a larger scale, but did establish a foundation for this research. This work looks to build upon these previous efforts in the area of multipath data transport to improve ABR video streaming. Previous research has not been done using MPTCP with HTTP/2 for ABR optimization in lossy networks nor have there been multipath techniques unreliant on client-side network estimation and/or buffer-state. There is huge potential in deploying the combination of MPTCP and HTTP/2 and there is a need for new ABR algorithms to fully realize what is possible.

SDN research has led to various multipath delivery solutions. Sonkoly et al. [28] designed a large-scale multipath testbed with GEANT OpenFlow to allow for the testing of real world traffic performance. Looking more closely at self contained networks, another study focused on multipath data centers with optical SDNs [29]. These and other previous research has often ignored shared bottlenecks that frequently occur. Sandri et al. [30] conducted a study that focused on improving the performance of shared bottlenecks using OpenFlow and MPTCP. Nam et al. [31] took a different approach by changing controller types and investigating 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. At the network layer, there are many other versions of TCP that attempt to pool network resources similar to MPTCP (i.e. Stream Control Transmission Protocol (SCTCP) [32]). However in practice, many of these techniques have failed to find community wide adoption. To date, there has been little focus on the performance of state of the art ABR video streaming in lossy

HTTP/2 multipath networks.

Peer-to-peer networking is another example of users pooling networking resources together using a distributed architecture to accomplish a larger goal without the need for a single coordination node. More recent efforts in reducing bandwidth costs across the Internet has led to the development of content delivery networks (CDN). CDNs are a cooperative of network components across the Internet whose main purpose is to cache content spatially close to end-users for better response time and availability. CDNs have helped improve user QoE in unreliable network conditions however, CDN innovation has become stagnate in developing solutions for personalizing video content delivery for individual users [33]. Bandwidth requirements are a primary limitation for VR streaming. Research has shown a 720p VR video can require anywhere from 50 Mbps and 500 Mbps for 4K videos [34].

CDN-based efforts aimed at reducing networking costs have been utilized for years. Video sources in CDN-based delivery networks push content to a number of delivery servers at network edges. Clients then in turn download content from the delivery servers directly instead of the video source. YouTube and many other large content providers use this technique to shorten start-up delays and reduce network strain. Unfortunately, this is largely where research into CDN operation has slowed, leading to increasing server loads as users demand higher quality content. Leveraging existing CDNs is a focus of this work for tailoring the streaming experience without exhausting more resources.

### 2.2.2 Quick UDP Internet Connections (QUIC)

Application layer ABR algorithms in use today were not developed with multipath networks in mind. Many implementations are hard-coded using simple heuristic threshold parameters which make adapting to varying network conditions problematic. Furthermore, there are issues with suboptimal bandwidth estimation related to inconsistent video encoding leading to inaccurate bitrate decisions. There is a long history of UDP based methods for transmitting content on the Internet. This work is focused on a new protocol designed

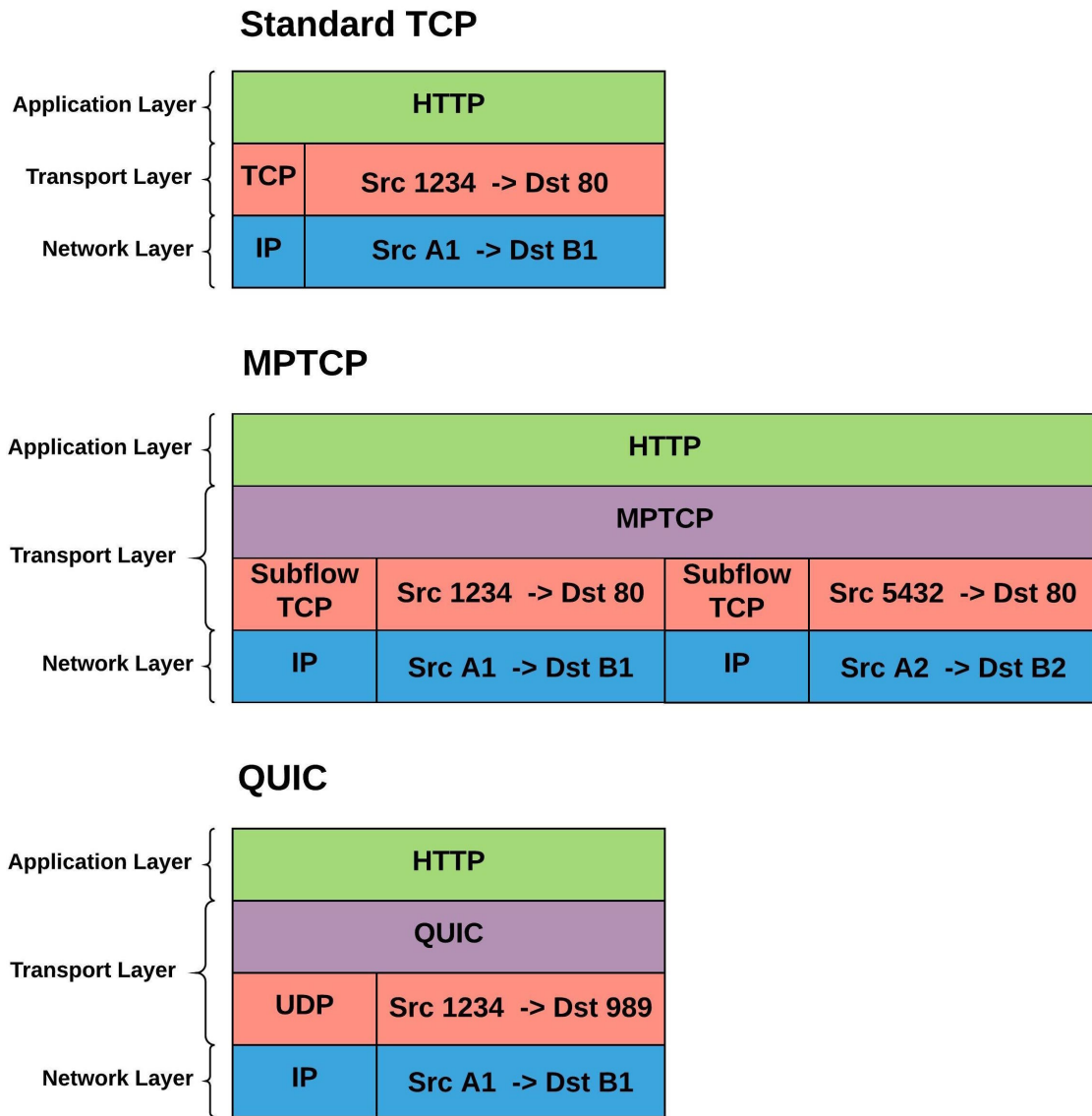


Figure 2.5: High-level transport framing

by Google called QUIC [35], which is used by Gmail, Google Drive, and many others. Google researchers indicate 50% of the traffic from Chrome browsers leverage QUIC [36]. Furthermore, Google claims that QUIC's performance improvement is increasingly noticeable within video services (e.g., YouTube) resulting in 30% less buffering. As a result, researchers have taken keen interest to study its performance instead of solely TCP-based methods. One researcher conducted experiments involving the implementation of QUIC comparing its goodput and link utilization to SPDY and HTTP [37]. This study unfortu-



nately did not focus on various network types. Biswal et al. [38] focused on measuring web page load times in wired and cellular networks, concluding that QUIC is robust under poor network conditions.

QUIC's objective is to provide at minimum the same level, but in most instances better functionality than TCP with TLS security into an application level protocol that operates over UDP. QUIC's choice to make an application level protocol operate over UDP allows for quick and easy protocol updates. Google controls both endpoints, the Chrome browser application and the Google service/servers, enabling protocol iterations on a large scale. Traditionally, updating a transport protocol meant users needed to update the kernel of their operating system (OS) and hope that other OS vendors would follow suit. QUIC aims to reduce latency in mobile and wire line networks while solving issues with multiple streams over a single TCP connection all the while maintaining comparable TLS privacy and security. To eliminate the head-of-line blocking issue, QUIC multiplexes various streams with their own ID into a single stream. This way, the client can continue to receive data without being limited by slower multiplexed streams. This is accomplished with each multiplexed stream using its own congestion control protocol per stream instead of at the connection level.

UDP allows for lower latency when establishing connections. TCP requires a three way handshake and with the integration of TCP and TLS-like features, QUIC eliminates two complete handshake cycles. This is detailed in Chapter 4. QUIC can even eliminate handshakes all together using what is called a 0-RTT connection. This connection takes place when QUIC moves the transport layer control to the endpoint's application space, thereby allowing rapid nonintrusive innovation. When a QUIC connection is established, the client can simply send a SYN packet with an encrypted session key. If the key is still valid on the server, the connection can proceed without any acknowledgment packets. The idea of 0-RTT is not a novel concept as this idea has been illustrated in former studies, TLS Snap Start [39] and TCP Fast Open [40]. Traditionally, a network connection consists of

linking a source port and IP address together. In QUIC, the connection paradigm decouples the source port from the IP address, as connections are made using a connection ID in the packet header instead. QUIC uses forward error correction (FEC) after a set number of packets that is configurable to reduce the need for data retransmissions. An early QUIC study conducted by Connectify [41] analyzed the performance of QUIC in three scenarios focusing on connection establishment latency, and download times concluding that QUIC outperforms standard HTTP when there is more than 1% packet loss on the network. The reasons for this was that QUIC uses FEC unlike HTTP. This research is interested in exploiting QUIC's latency improvements (e.g., 0-RTT connection setup) and stream multiplexing, combining application and transport decisions.

### **2.3 Software Defined Networking (SDN)**

End user requirements for the delivery of high-quality media content creates unique challenges when considering operations on a large scale. Systematically taking advantage of existing network capacity is multifaceted and complex. MPTCP can leverage numerous networks simultaneously, but its performance can suffer in congested environments due to inherent trade-offs between responsiveness and load balancing. Considering a technique that mitigates these limitations is of particular interest to the research community. SDN (see Fig. 2.6) is an abstraction paradigm meant to address problems of traditional static network architecture. SDN decouples decisions about where data is sent from the hardware, thereby enabling flexible network configurations. SDN can be leveraged to dynamically control QUIC and MPTCP available paths in a way that is invisible to transport and application layers, particularly when operating among many users simultaneously. Research has shown promising results using network simulators combined with Mininet [42], a network emulation system that runs a set of hosts, switches, routers and links with controllers (i.e., Floodlight [43]) to manage large networks. OpenFlow [44] is a widely used protocol in Mininet that allows controllers to communicate with switches over a secure channel. The

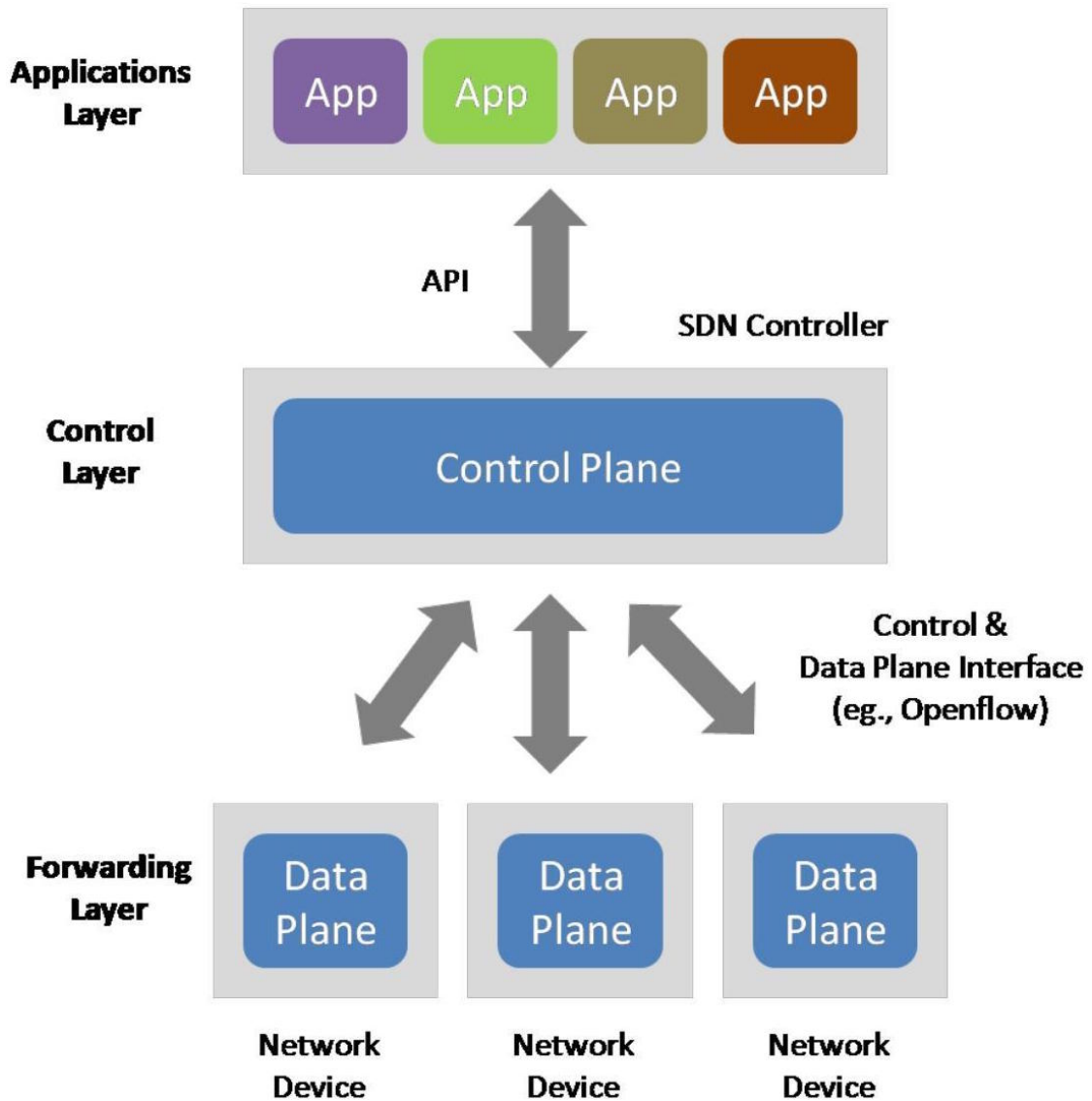


Figure 2.6: Software defined network layout

controller generally has a global view of the network topology and communicates with switches obtaining instantaneous network traffic information. Using network-wide information, the controller can decide to optimize traffic flow patterns in real-time. This work looks to design a framework that actively manages QUIC and/or MPTCP usage with a large number of users.

Uzakgider et al. [45] used an SDN and machine learning techniques to optimize the time to reroute traffic flows while reducing packet losses and quality changes while keep-

ing the built-in MPTCP scheduler constant. Conversely, Kukreja et al. [46] designed a small testbed evaluating the performance of each built-in MPTCP scheduler in an SDN architecture. Results from this study indicated that the lowest RTT scheduler performs the best for jitter intolerant applications, but the round robin scheduler utilized the network more efficiently. Conversely, Sonkoly et al. [28] designed a massive multipath testbed with GEANT OpenFlow to allow for the testing of real world traffic performance. After incorporating the strengths and improvements from the studies mentioned above, a comparison is made to Nam et al. [31] which similarly investigated multipath networks with POX controllers. This research has built upon these previous efforts in the field of multipath data transport to improve video streaming.

## **2.4 Reinforcement Learning (RL) for Adaptive Streaming**

Years of video streaming application development has primarily focused on client-side video players optimizing the streaming experience by themselves. In spite of this abundance of research into streaming algorithms, many experience the same limiting factors. The ABR rules in place making bitrate decisions are often based around static threshold decision rules. Oftentimes these techniques are based around single path networks even though clients have access to multiple networks. Networking models used by clients are often inaccurate and unable to predict and react to network bandwidth fluctuations. For the above mentioned reasons along with others, user QoE results in a suboptimal experience. Within the research community, there has been a resurgence of machine learning as promising results have been shown within problem spaces that are well defined but difficult to predict outcomes ahead of time. As a result, the application of learning-based techniques, in particular reinforcement learning in adaptive bitrate streaming, is a recent development discussed within the literature. Early studies have limited their work to simulated synthetic network traces [47, 48] and as a result, rely on a state space small enough to store a value per state in an allocated computer memory array. This decision limits the scal-

ability of their designs. Mao et. al [49] used RL and found that it is especially effective in highly variable complex networks where heuristic algorithms underperform. Despite these findings, this work is limited due to modern high bandwidth video requirements not being considered nor multipath networks. Seminal studies have often relied on deep Q-networks (DQN), however when comparing DQN to Asynchronous Advantage Actor-Critic (A3C) [50] modeling, A3C is an improvement in utilization [51]. A3C has the advantage of combining the benefits of Q-learning and policy iteration methods, and as a result, is used in training the models in this work. The effects of RL in multipath networks are often not taken into consideration which limits the flexibility of many designs. Hooft and Mao et. al [49, 52] used RL to train ABR algorithms on a larger scale, however in both studies, video types were not distinguished. Additionally, these studies did not consider 360 VR video requirements nor multipath networks which this work is trying to aid in addressing. A more recent study [53] does consider 360 VR videos with learning algorithms however, is limited to SDN networks only. Looking to expand on this research using realistic parameters that do not rely on managed networks is of particular interest to this assessment.

Load balancing is an issue not mentioned previously that RL is able to address within multipath networks that single path networks do not have to worry about. Research has illustrated that unbalanced multipath networks often underperform single path networks [7]. Buffer bloat results from unbalanced flows leading to bursty data arrival, which can be problematic for streaming applications. TCP-based protocols in highly varying network conditions can be ineffective, which is why protocols oftentimes use UDP. UDP-based QUIC has been found to provide better performance given these situations. This work incorporates the strengths and improvements from the studies mentioned and is used to compare this system to many leading algorithms.

### **CHAPTER 3**

#### **EXTENDING DASH WITH MPTCP SERVER-PUSH**

This section summarizes the techniques and findings from preliminary research. As mentioned in Chapter 1, the objective of the preliminary work was to begin to design and develop a unique approach to efficiently deliver high-quality multimedia data in multipath networks specifically by introducing techniques to improve the efficiency and response time to network and user requirements. By utilizing concepts illustrated in various multipath networking models in conjunction with multipath-aware ABR protocols that supplement the reliance on client-side network bandwidth estimation and/or buffer-state, it is hypothesized that leveraging multipath networks intelligently can result in shorter media start times, higher bitrate content, fewer stalled playback buffers and increased network awareness in lossy network conditions when compared with non-multipath aware techniques. To begin investigating these claims, a system was established for improving server response times in local area networks. Through the use of this system, a tool for analyzing multimedia performance on a local area network and dynamic transport selection was developed. Validated metrics from prior work informed the performance benchmarks studied for experimentation.

The de-facto application streaming standard over the Internet is HTTP/1.1. However, as detailed in Chapter 2, HTTP/2 is becoming widely supported replacing HTTP/1.1 as the preferred method for browsing and streaming multimedia on the World Wide Web. HTTP/2, being backwards compatible with HTTP/1.1, is completely cache and proxy friendly with existing networking technologies. Taking advantage of the new features of HTTP/2 such as server push enables researchers to investigate new streaming solutions; especially when considering multipath networks which many studies fail to consider within modern networks. Commonly used ABR frameworks rely primarily on the estimation of

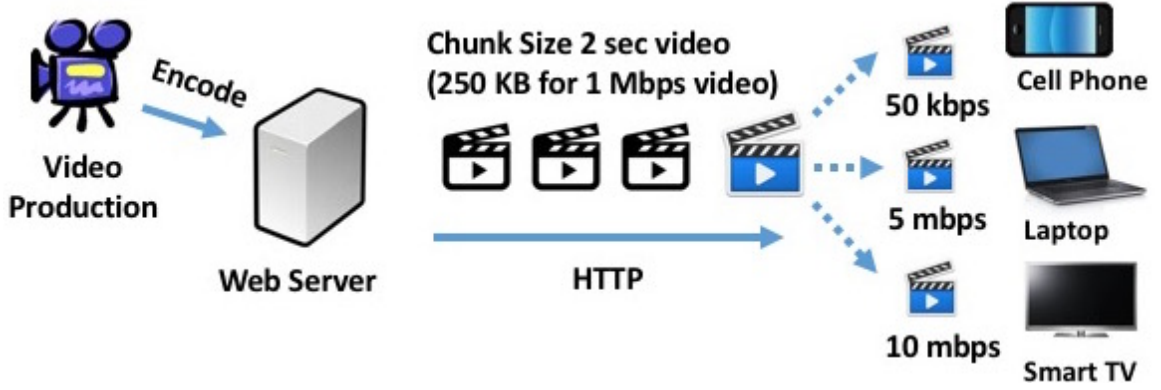


Figure 3.1: Example adaptive bitrate streaming flow diagram

network bandwidth and/or buffer-state client-side (see Fig. 3.1). Inaccurate estimation quickly leads to performance degradation.

Using three modular HTTP/2 server push-based strategies leveraging MPTCP is an approach to removing the client-side estimation of network bandwidth and buffer-state with the goal of improving start-up delay and overall performance. The first strategy is called a start-up accelerator, in which a server is designed to reduce the client initialization overhead in lossy networks. This is done by filling the client’s initialization buffer more quickly than traditionally possible. The second strategy is playback stall elimination by using MPTCP’s *MPTCP\_INFO* option to asynchronously abort unsustainable quality levels on individual networks without waiting for timeouts. For the third strategy, a hybrid push/pull model was developed leveraging a unique two-way communication between multipath clients and servers. Each side has timely information about the video and networks that traditionally would be unobtainable when making streaming decisions. Finally, a systems integration design incorporating each modular strategy into one larger protocol was formed to evaluate the system’s overall performance.

### 3.1 Start-up Accelerator

A start-up accelerator (see Algorithm 1) is used to reduce the initialization overhead. During the initialization phase, *index segments* provide critical information related to corre-

sponding bitrate media segments. *Index segments* are small byte-range requests on the order of hundreds bytes [4]. Traditionally, after the media presentation description (MPD) is delivered, each *index segment* for each bitrate is requested one after another. Hundreds of bytes per request are quickly delivered however, the more bitrate options available, the more RTT the client has to wait for before playback rate selections can begin.

---

**Algorithm 1** Video Rate Adaptation Start-up Accelerator

---

```

1: procedure STARTUP-ACCEL
2:    $numBitrates \leftarrow$  number of representation IDs
3:    $numSubFlows \leftarrow$  number of subFlows in use
4:    $i \leftarrow 0$ 
5:   while:
6:     if  $i \geq numTcpFlows$  then return done
7:     if  $(numBitrates \% numSubFlows) \neq 0$  then
8:       Server Push  $(numBitrates \% numSubFlows)$ 
9:       bitrates over subflow  $i$ 
10:  for:
11:    Server Push  $\left\lfloor \frac{numBitrates}{numSubFlows} \right\rfloor$  bitrates over subflow  $i$ 
12:     $i \leftarrow i + 1$ 
13:    if  $i \leq numTcpFlows$  then
14:      break for
15:  break while

```

---

The accelerator (see Fig. 3.2) is a potential way to reduce the overhead serving cached data by the server and can also improve start-up delays for clients. HTTP/2 frames are able to be prioritized allowing clients to give servers a priority ranking of some assets over others. Using this feature, the clients prioritize which initialization assets they are interested in, then the HTTP/2 server push feature is used to transmit multiple byte range responses for all requested *index segments* at once. *Index segments* are on the order of hundreds of bytes and are required by each client before any playback decisions are made. As a result, aggregating them into one push message, even with multiple clients simultaneously accessing the same content, will not generate enough data to create a substantial bottleneck. With MPTCP, the increased byte range response traffic is distributed over available network interfaces starting on the path with lowest RTT until its congestion window is full. This



simple but elegant use of HTTP/2 server push has the biggest impact on start-up delays for short form video content (e.g., Snapchat, Vine, etc). The drawbacks of this method, however, include if a client already has the video's initialization data from a previous viewing session in its cache. Pushing the data again would be a waste of bandwidth. For these experiments, this situation is not considered as they are primarily concerned with delivering video to first-time viewers. However, this effect can be mitigated in real-world use by having the client send a *RST\_STREAM* frame to abort the initialization push. Additionally, if the client has the initialization data in the cache, a request for the MPD might not actually occur as it hits its cache before sending requests over the network.

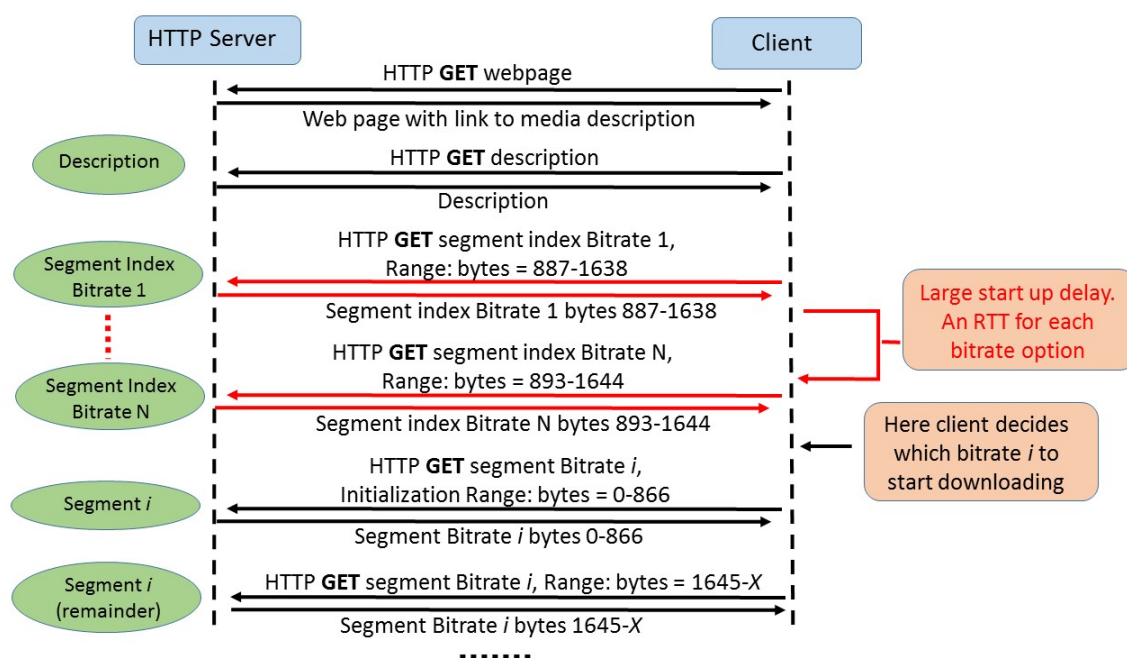


Figure 3.2: DASH start-up flow used during experimentation

### 3.2 Zero Stall Playback

For the second contribution, HTTP/2 is used to maintain continuous playback. HTTP/2 can eliminate head-of-line blocking which is a major cause of playback stalls seen in HTTP 1.0/1.1 at the application level by multiplexing multiple streams per connection. As a result, the server can transmit a separate *chunk* per stream per network connection with a caveat

that the bandwidth available is higher than the lowest quality level available on the server. This assumption can be made due to the fact that no system would be able to transmit data successfully when the bandwidth falls below the lowest bitrate.

For the design of the HTTP/2 MPTCP based protocol, when the client's playback buffer is critically low, using MPTCP's *MPTCP\_INFO* option the highest RTT network receives a *RST\_STREAM* frame. Reducing the delivery delay of critical segments, a configurable  $n$  segments are pushed without a single HTTP Get request from the client. These frames are not sent, however, if only one network is available. This  $n$  push feature is optimized to eliminate buffer stalls in the worst performing networks. A WebSocket is used to facilitate the communication between the server and the client for unscheduled push encounters. The server immediately upon receiving this signal starts pushing the lowest available bitrate so the client will always have something available in its buffer. Using MPTCP with 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 an RTT that is usually required when switching bitrates. This technique will be increasingly effective in networks with higher RTTs. A drawback of this method, however, is the potential of frequent oscillations between higher and lower bitrates. This situation can be mitigated by not automatically dropping to the lowest bitrate, but landing somewhere in the middle. Given this, the algorithm is more aggressive to maintain continuous playback based on user experience modeling [54].

### **3.3 Hybrid Push/Pull**

Traditional adaptation algorithms are client pull-based, pushing all of the adaptation complexity to the client. Client-side rate adaptation has many benefits, from reduced server computation to server scalability. However, the client is missing information needed for improved bitrate selection. At the server, there is complete information about the data being served compared to the rigid and imprecise nature of the client-side MPD. The MPD

specifies the average bitrate across a video’s entire encoding, but each *chunk*’s actual bitrate can be much higher or lower than advertised. The web server, however, knows the exact bitrate of each *chunk*. This work is not proposing a server-side adaptation technique, but rather proposing a hybrid model specifically leveraging HTTP/2 for use with MPTCP.

For the third contribution, the client can use standard client-based pull decisions or relay timely buffer-state and network capacity information using WebSockets to the server. Instead of estimations, the server can now make more accurate hybrid-based server push decisions. Using MPTCP’s *MPTCP\_INFO*, the algorithm can monitor the performance of each subflow and correlate its performance with information received over WebSockets. This opens up new classes of decision algorithms based on individual subflows instead of total connection performance. For example, this algorithm can preemptively push bitrates expecting an RTT that it knows a certain subflow can support even if the total connection RTT is different. For experimentation, the decision logic chooses the bitrate quality of the segments to push to the client. This choice relies on a simple tunable weighting of the buffer level, quality, and outstanding data on the network, for the experiments, equal linear weighting was chosen. The number of simultaneously pushed segments were fixed to three to avoid pushing too many segments at once, but can be adjusted for different situations. Lastly, the client at any time can asynchronously notify the web server through a *RST\_STREAM* frame to abort its current push and request a higher or lower bitrate if desired. A potential drawback of this technique is pushing too many or too few segments at once in quickly changing network conditions. However, this concern can be addressed by tuning the weights.

### **3.4 Experiment Setup**

As mentioned above, presented here is the detailed experimental setup of a proof of concept implementation combining three techniques into one cohesive idea as each focuses on a different portion of the streaming experience. All of the experiments were conducted using the

DASH-compliant ‘Big Buck Bunny’ [55], an animated movie. The content was encoded with x264 [56] at 20 different well-spaced bitrates between 45 Kbps and 3.8 Mbps. To compare the performance of TCP and MPTCP, the following experiments were conducted while varying the network conditions. For all experiments, the aggregate bandwidth available on the MPTCP network matched the bandwidth of the single TCP flows. For example, if the conventional TCP network had 2.6 Mbps available, the MPTCP equivalent had two networks available of 1.3 Mbps for each network. From the start of the experiment to  $t = 120s$ , the available bandwidth was 2.6 Mbps. At  $t = 120s$  until  $t = 240s$ , the available bandwidth was reduced to 1.3 Mbps. At  $t = 240s$ , the available bandwidth was 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 [16, 22]. The packet loss rate varied at discrete points of 1% increments from 0% to 5% based on [57]. Given many existing studies are evaluated in lossless networks, real-world representative results occur when considering lossy networks.

Data was logged from experiments via the JavaScript console provided by the Google Chrome web browser and Wireshark captures. To compare performance, studied were 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.

On the client-side, the solution is built on top of the code for the default algorithms used by the DASH-IF reference client [6]. One of the algorithms is primarily network-capacity-based, whereas the other, called BOLA [9], is a buffer-based algorithm.

As shown in Fig. 3.2, the actual initialization flow with byte ranges of data used for experimentation is displayed. 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. Before the client can begin playback, *index segments* are needed from each bitrate available. This creates a problem given a large network RTT combined with a broad range of bitrate encodings, resulting in a start-up playback penalty.

### 3.5 Testbed

The architecture of the testbed, shown in Fig. 3.3, supports both MPTCP v0.90.0 and TCP. The testbed consists of four nodes and a gigabit switch; 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). The traffic shapers also control the maximum achievable throughput with Linux traffic

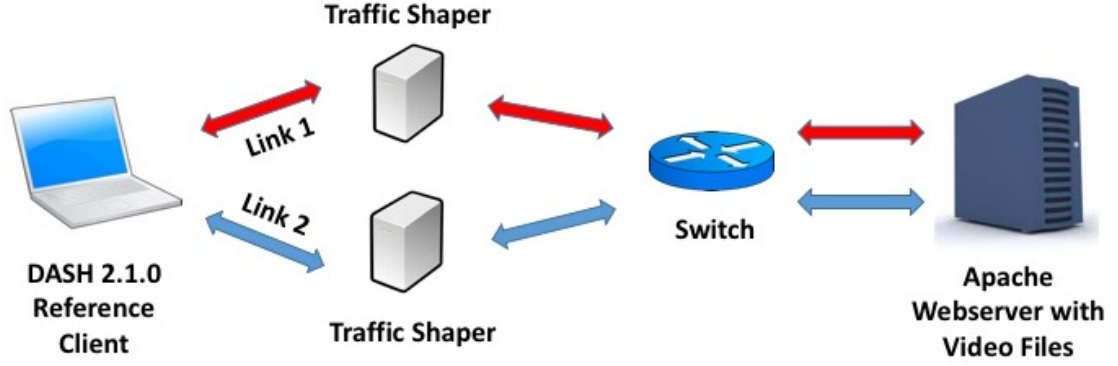


Figure 3.3: Experimental lab testbed setup for HTTP/2-based experiment

control system (tc) and the hierarchical token bucket (htb), which is a classful queuing discipline (qdisc).

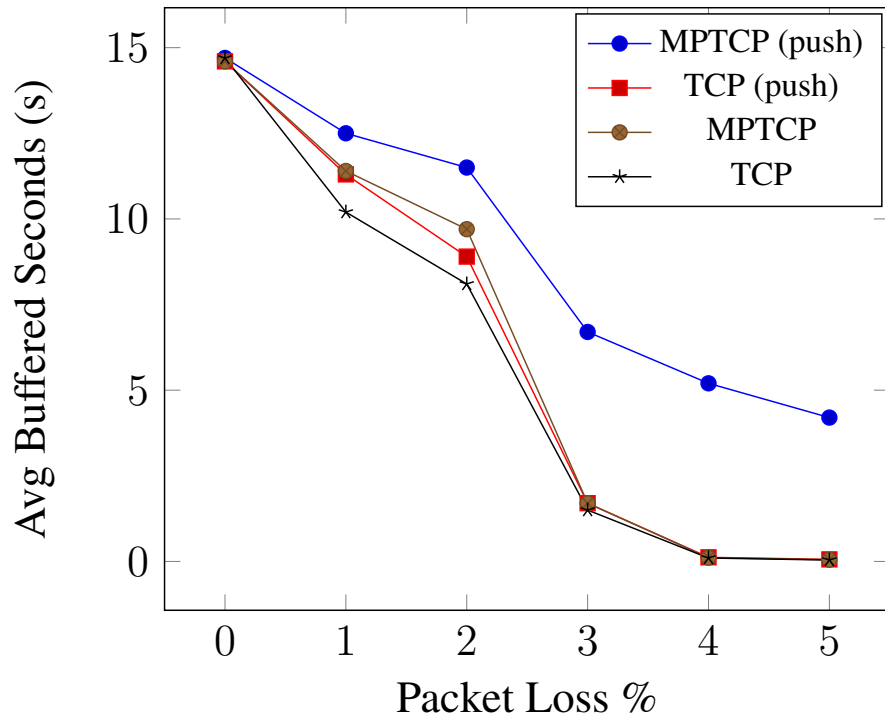
The HTTP/2 Apache v.2.4.23 web server, which stores all the related DASH video segments and a manifest file, was configured to use persistent connections. The client machine uses the DASH capable browser Google Chrome v49.0.2623.87 for playback using the JavaScript-based DASH-IF Reference Client v2.1.0 while making sure to clear the browser cache in between tests. The client’s buffer capacity is 30s for content under ten minutes and 60s for content over ten minutes.

The default stable buffer target is 12s. TLS is supported over MPTCP, however, the effects of the required TLS connection in HTTP/2 for DASH across multiple paths was not the focus of this research, but will be investigated in future work.

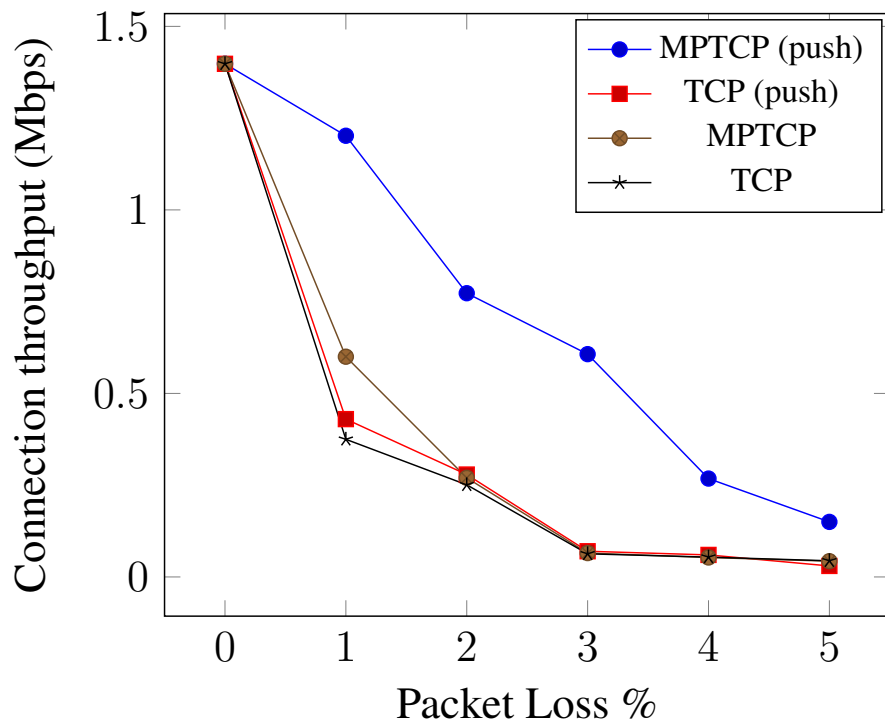
### 3.6 Evaluation

The plots below are the result of experimentation in the 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. MPTCP(push) and TCP(push) refer to the solutions presented in this chapter being enabled as compared to their standard implementation without modification.

Shown in Fig. 3.4a, a comparison between the average number of seconds of video in

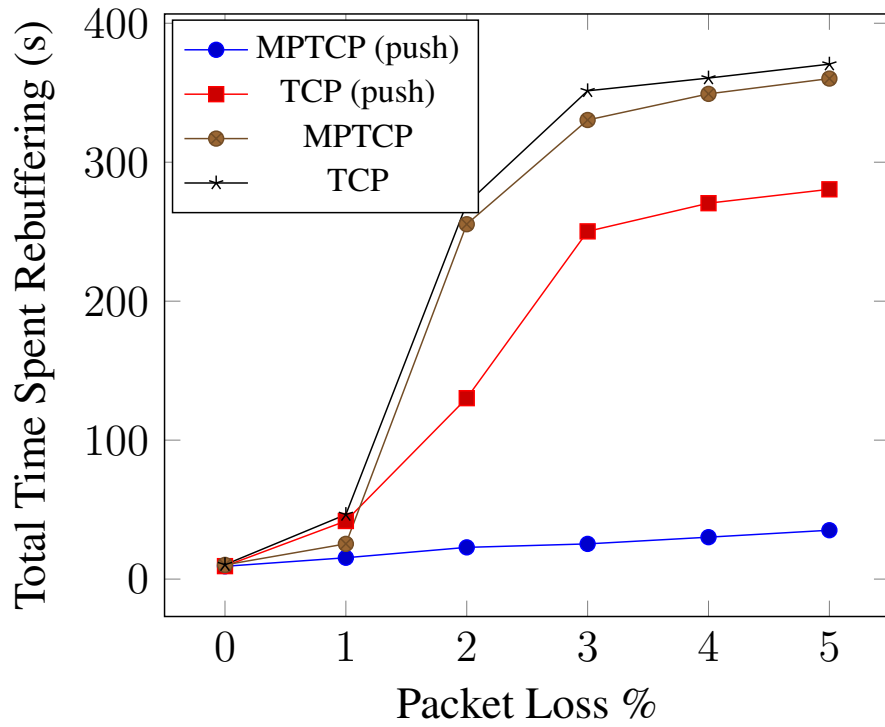


(a) Average buffer size

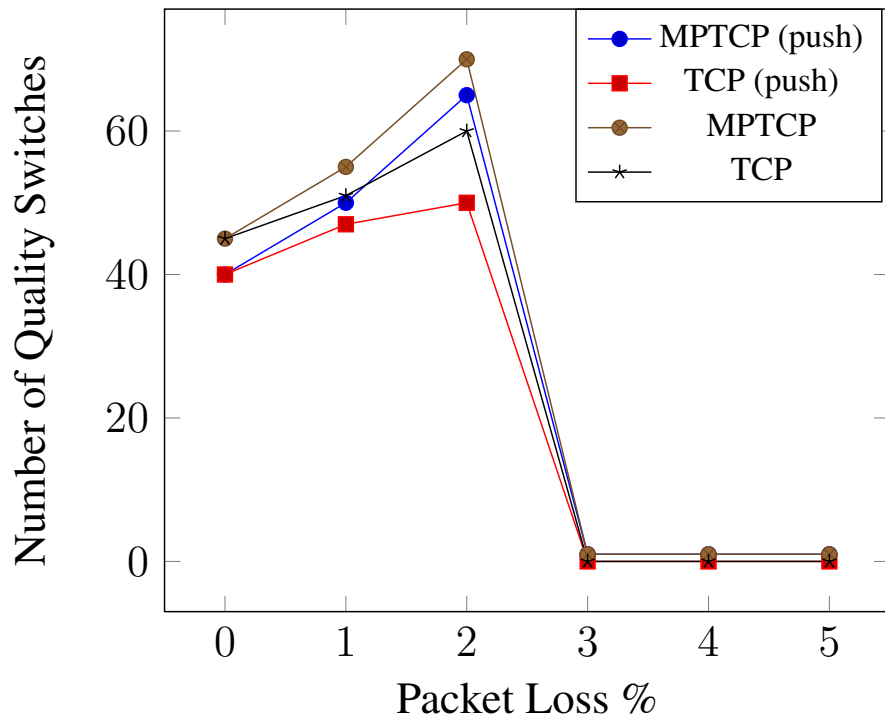


(b) Average connection throughput

Figure 3.4: Various performance metrics for a given packet loss rate percentage.



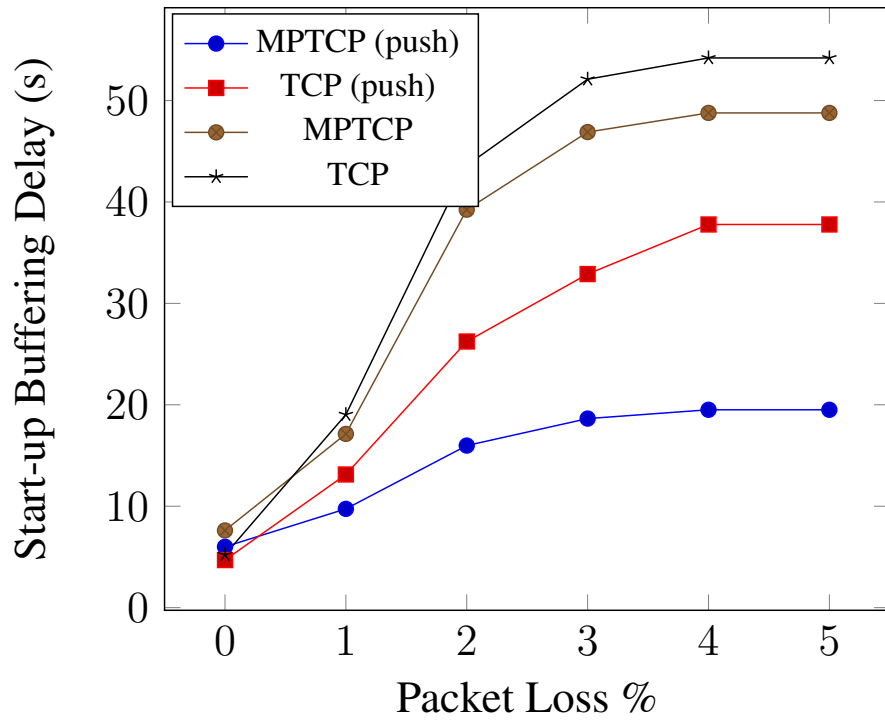
(a) Average time video playback stalled



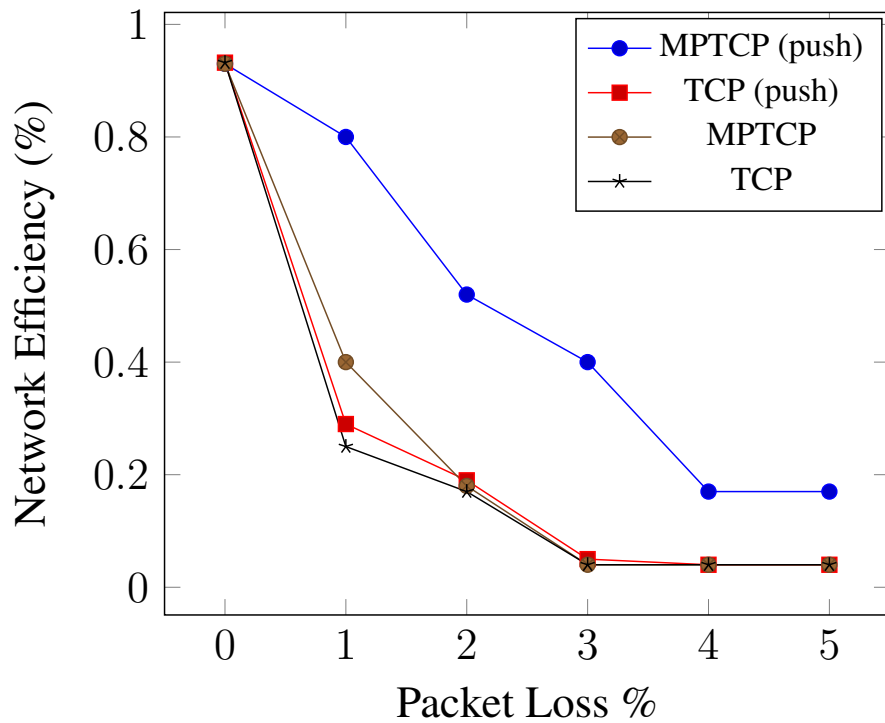
(b) Average number of quality switches

Figure 3.5: Various performance metrics for a given packet loss rate percentage.





(a) Average start-up buffering delay



(b) Network usage efficiency

Figure 3.6: Various performance metrics for a given packet loss rate percentage.

the buffer vs. packet loss rate with and without HTTP/2 server push for TCP and MPTCP was made. This plot does not show the quality of the buffered seconds of video. To get a complete picture, one needs to evaluate Fig. 3.4a with Fig. 3.4b. When the packet loss rate is zero, MPTCP and TCP perform comparably. This is because MPTCP is designed to perform exactly like TCP at bottleneck links to promote fairness with contending TCP flows. These results are consistent with previous studies [7, 15, 16]. Interestingly, as the packet loss rate increases, MPTCP(push) performs better when compared to standard MPTCP or TCP with or without push enabled. This is due to the fact that TCP(push) does not have an additional path to leverage this strategy to recover with increasing loss rates. In addition, MPTCP natively does not take advantage of HTTP/2 push to quickly overcome losses by switching to lower bitrates asynchronously. Further, MPTCP relies on timeouts when switching paths, unlike this modified version. Except for MPTCP(push), when packet loss rates reached 5%, Fig. 3.4a and Fig. 3.4b illustrate that standard MPTCP and TCP were unable to maintain usable bandwidth over the lifetime of the connection.

As shown in Fig. 3.5a, MPTCP(push) spends significantly less time stalled with an empty buffer than standard MPTCP and TCP with or without push. This is attributed to two-way communication with the server and client as they are able to transmit accurate per path network characteristics. Additionally, the start-up accelerator made a significant difference when experiencing high packet loss rates if a stall was imminent. TCP(push) improved TCP's performance significantly with the presented no stall algorithm, switching to the lowest bitrate more quickly than unmodified MPTCP.

As mentioned in the introduction, one factor affecting QoE is the number of bitrate quality switches during a streaming session. Users desire high-quality video but want to avoid too many oscillations as that degrades the overall QoE. In Fig. 3.5b, at 4% and 5% loss rates, MPTCP and TCP 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%, TCP has fewer quality switches than MPTCP. This is concluded for a few

reasons. One reason is that TCP is not able to successfully transmit as much data through the single network and, as a result, tends to select the lowest bitrate consistently; whereas MPTCP tries to ramp up quality more often due to a higher bandwidth. As a result, the bitrate selection oscillates more frequently.

Shown in Fig. 3.6a, TCP(push) with 0% packet loss is faster than MPTCP(push) while in the start-up phase of a video connection. Concluding from the analysis, this is a result of the overhead of MPTCP 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 at a 0% packet loss starts up slightly slower than TCP, but with a higher bitrate. At high packet loss rates, MPTCP(push) began playback more quickly and with a higher bitrate than traditional TCP and unmodified MPTCP. Finally, Fig. 3.6b revealed that except for a 0% packet loss rate, MPTCP(push) utilized more of the available bandwidth than the other configurations. At 0%, TCP has a slightly higher efficiency due to the lower connection overhead.

### **3.7 Discussion**

Demonstrated in this work is a technique for improving ABR video performance using HTTP/2 with MPTCP. It focused on improving the overhead of the initialization period and the overall streaming experience by implementing three novel modular HTTP/2 push strategies over MPTCP. HTTP/2 permitted some of the streaming complexity logic from the client to be assisted by the server. Using MPTCP allowed for the leveraging of path diversity for increased performance. As a result, the start-up buffering delay decreased by 40% or more when using MPTCP(push) as compared to TCP or MPTCP when packet loss rates were above 1%. Playback stall time also decreased by more than 5% when packet loss rates were above 5%. Furthermore, MPTCP(push) utilized the network 5% more efficiently with packet loss rates above 0%. The results demonstrate that this technique outperformed standard MPTCP and TCP in increasingly lossy networks by 5% or more in many key

streaming metrics. These preliminary results were for a controlled proof of concept and may vary with multiple users.

As described in later sections of this research, an investigation of various MPTCP path management algorithms for pushing data across individual subflows was assessed. Furthermore, research into investigating the impact of TLS on HTTP/2 in DASH over multiple paths was explored. Finally, obtaining results under conditions where the two links ending at the client device are asymmetric was further studied.

## CHAPTER 4

### FLEXIBLE TRANSPORT LAYER FOR HIGH BANDWIDTH CONTENT

Detailed in the previous chapter was a preliminary investigation into the practical usage of multipath techniques in multimedia streaming when users had access to multiple networks to exploit unrealized potential in existing networks. That was accomplished in a small real-world testbed that used MPTCP along with modifications to the server and client that were unique to multimedia streaming. The study concluded there were a number of results that challenged some initial assumptions, one of which was the idea that TCP-based solutions were generally more effective in lossless multipath networks. Compared to UDP-based solutions, TCP incurs a larger overhead penalty. This penalty is exasperated in various commonly occurring situations, which include highly fluctuating networks. In circumstances where high packet loss rates are common or in large RTT networks, such as cellular networks, a UDP-based solution may provide better performance. With this in-sight and continuing to build upon lessons learned from previous chapters, a hybrid TCP/UDP solution was designed. Fig. 4.1 displays the representative structure of the MPTCP/QUIC-enabled ABR network leveraged for this study.

In this chapter, two ideas are presented to improve omnidirectional video streaming in lossy environments by leveraging MPTCP, QUIC, and SDN into a single solution. The first strategy is called a connection decision engine, in which an SDN application is designed and linked with a 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, the algorithm dynamically modifies the most appropriate transport protocol. The second strategy called playback network monitoring which uses MPTCP's *MPTCP\_INFO* option and SDN's network path statistics. The algorithm monitors the number of retransmits each subflow experiences along with con-

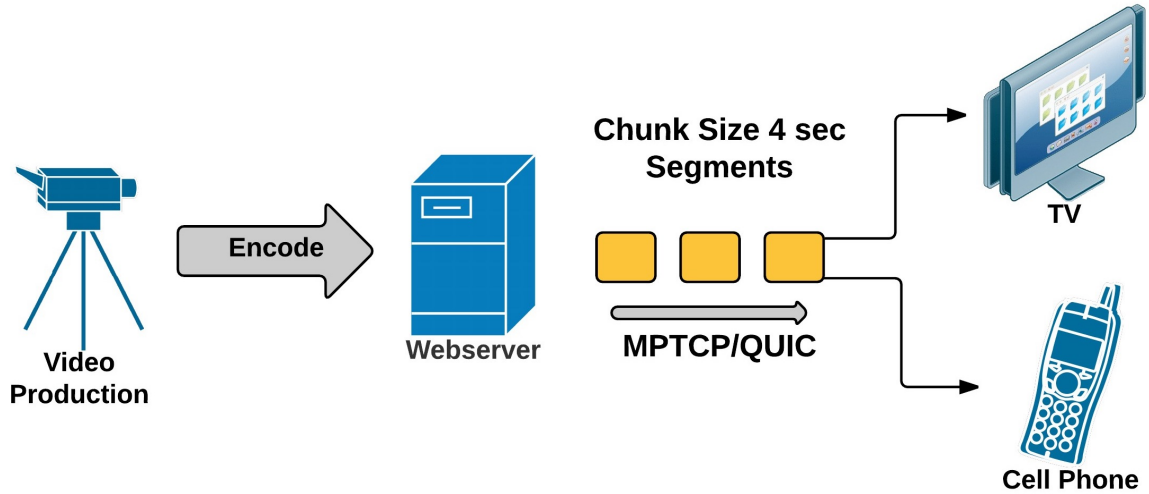


Figure 4.1: MPTCP/QUIC-enabled ABR network.

tinually 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, each strategy was integrated into one larger protocol and its overall performance was evaluated.

#### 4.1 Connection Decision Engine

The driving goal behind the connection decision engine is to reduce the streaming initialization overhead. Studies show that systems using unbalanced multipath networks often experience worse initialization performance than simply using a single network [7, 15, 26, 31]. MPTCP, in particular, can suffer from bloated reordering buffers causing data to arrive in bursts instead of a steady stream that can create problems for high-bandwidth 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 [36]. Taking a closer look, Fig. 4.2 illustrates QUIC's potential for quick connection establishment with comparable security to TCP with TLS.

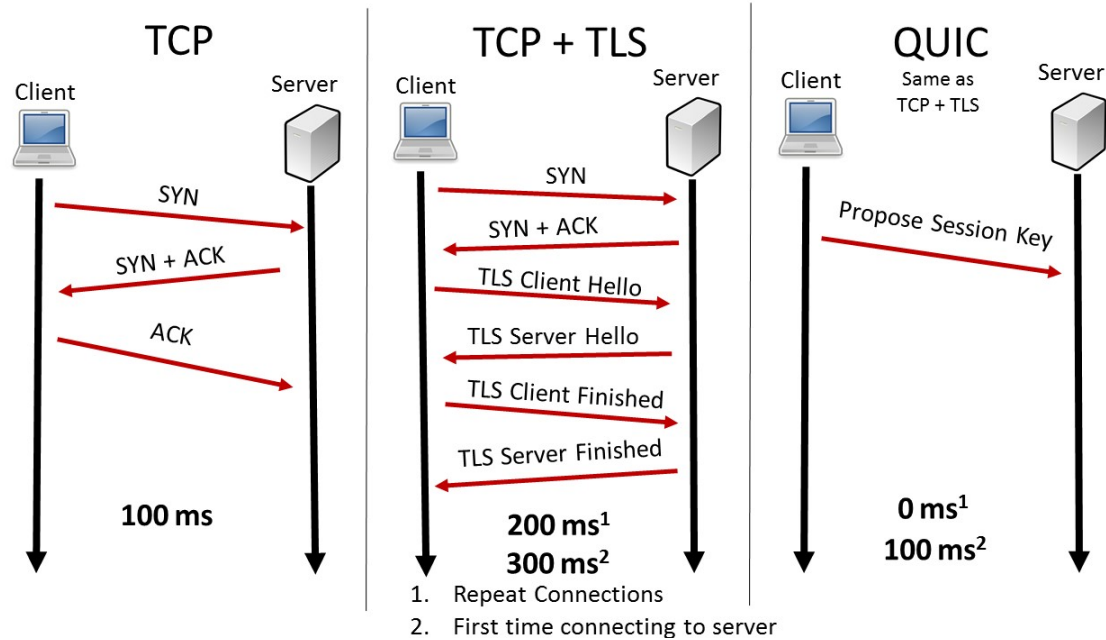


Figure 4.2: QUIC vs. TCP with TLS connection establishment.

Efforts in reducing initialization connection delays for clients led to the design of the following detailed algorithm (reference connection setup of Algorithm 2). If only one path is available to the server, meaning there is only one active network available to the client, the application defaults to using the QUIC protocol. This is because it performs better than TCP in many likely scenarios [36]. If two networks are available to the client, the SDN application compares differences in the average RTT for each network path. If the difference in the average RTTs are larger than 9:1 [58] (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 because of the aggregate throughput [15]. 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 connection 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 these experiments, this situation is not considered as it is primarily concerned with delivering video to networks that do support these options. However, this effect can be mitigated in real-world use by falling back to traditional TCP which MPTCP supports.

---

**Algorithm 2** MPTCP/QUIC SDN Decision Engine

---

```

1: procedure ROUTE-SELECTION
2:    $retransThres \leftarrow$  retransmission threshold
3:    $rttThres \leftarrow$  threshold for the change in RTT
4:    $\Delta rttList \leftarrow$  RTT compared to lowest available
5:    $rttList \leftarrow$  live avg RTT for each available path
6:    $numSubFlows \leftarrow$  number of subFlows available
7:
8:   Connection Setup:
9:   if  $numSubFlows \equiv 1$  then use QUIC protocol
10:  if ( $numSubFlows \equiv 2$ ) and
    ( $\Delta rttList < rttThres$ ) then use MPTCP
11:  else
12:    if  $numSubFlows \equiv 2$  then use QUIC on
13:    lowest RTT path
14:  if  $numSubFlows > 2$  then
    using  $rttList$  if any path  $\Delta rttList > rttThres$ 
    remove path using SDN if two or more paths
    remain use MPTCP otherwise use QUIC on
    lowest RTT
15:
16:   Connection in Progress:
17:   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

```

---

## 4.2 Playback Network Monitoring

For the second part of this contribution, SDN and MPTCP's *MPTCP\_INFO* option was used 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.).



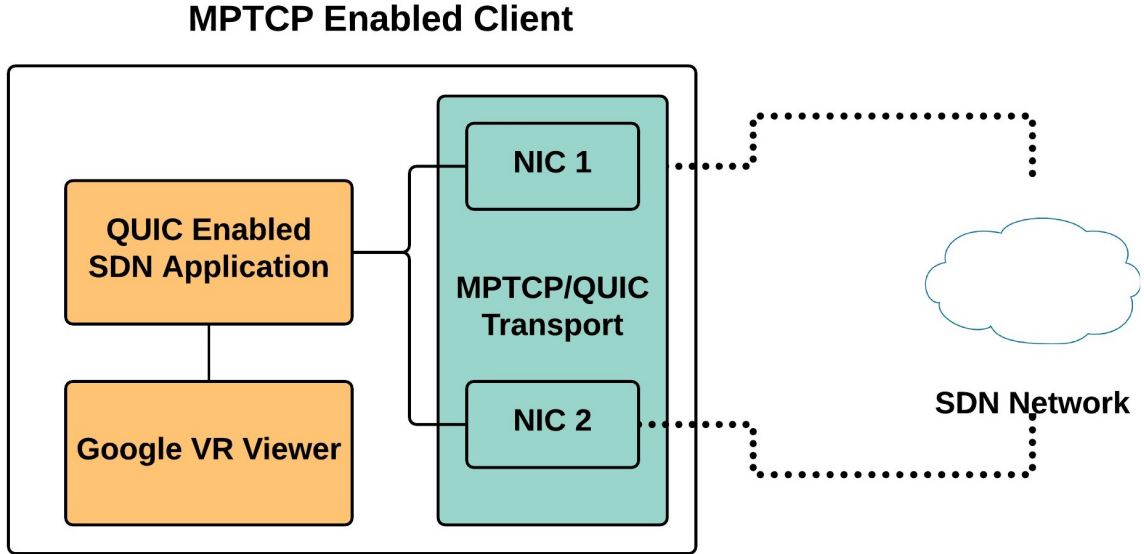


Figure 4.3: MPTCP/QUIC-enabled client used with an SDN.

One problem MPTCP experiences is the potential for a high number of retransmissions and duplicate ACKs in unbalanced networks [31]. This occurs when packets are buffered too long due to being delivered out of order, in which they arrive at the application late. Fig. 4.3 details the logical design of the MPTCP/QUIC-enabled client.

The ‘connection in progress’ section of Algorithm 2 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 the 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 a connection which is used for path elimination or addition. 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.

The flow elimination threshold of the algorithm was set to 10% above the number of retransmits of the highest RTT network. This was chosen as prior studies [7] showed that when retransmissions were above 10%, they 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, a *RST\_STREAM* frame is sent. The server immediately upon receiving this signal ends the connection and the client starts a 0-RTT QUIC connection with the server. Providing an opportunity to switch transport options mid-connection provides network flexibility for adapting quickly to network changes. 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.

### 4.3 Testbed

As a proof of concept, this section explains the testbed used in the implementation of the two presented strategies integrated into one since each focuses on a different portion of the streaming experience. See Fig. 4.4 for a closer look at the testbed setup. All of the experiments were conducted using encoded ABR VR videos captured with Google's Cardboard App [59]. The testbed setup parameters used for this particular study are similar to Chapter 3, comparing this new flexible transport system to the algorithms developed in the previous Chapter. Accordingly, the content was encoded with x264 [56] at 30 different well-spaced bitrates between 45 Kbps and 15 Mbps. To compare the performance of MPTCP with HTTP/2 push, SPTCP, TCP, and MPTCP with this implementation, the following experiments were conducted while varying the network conditions. For all experiments, the aggregate bandwidth available on the MPTCP network matched the bandwidth of a sin-

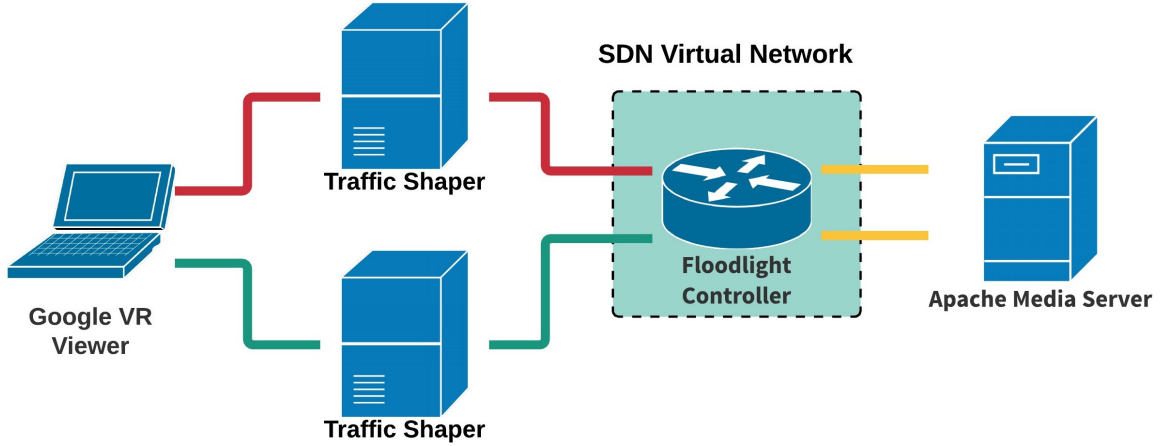


Figure 4.4: MPTCP/QUIC-enabled client setup used with an SDN network.

gle TCP flow. For example, if the conventional TCP network has 4.6 Mbps available, the MPTCP equivalent will have two networks available of 2.3 Mbps for each network. From the start of the experiment to  $t = 150s$ , the available bandwidth is 5 Mbps. At  $t = 150s$  until  $t = 300s$ , the available bandwidth is reduced to 2.5 Mbps. At  $t = 300s$ , the available bandwidth is increased to 5 Mbps until the end at  $t = 500s$ . Based on measurements from past the previous chapter, the client to server RTT was set to 120ms and the 4s segmentation was chosen. The packet loss rate were varied at discrete points of 1% increments from 0% to 5% based on [57]. Using lossy networks for experimentation are meant to reflect real-world network conditions.

Resulting experimentation data was collected and logged the via the JavaScript console incorporated into the Google Chrome web browser and network captures were captured at intermediate network nodes for evaluation. Comparing algorithmic performance, the following metrics were investigated similar to Chapter 3:

1. *Rebuffering time*; time spent waiting as the result of an empty playout buffer during a streaming session.
2. *Network Efficiency*; what percentage of the available network bandwidth is used.
3. *Start-up buffering delay*; the time from when the first video segment is requested to

when playback begins.

4. *Playback buffer size*; the number of seconds of video stored in the client buffer ready for playback.
5. *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.
6. *Average connection throughput*; the average throughput of the connection over the last 1s of the connection.

Fig. 4.3, 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 a Floodlight controller in the Mininet SDN network used during experimentation.

The architecture testbed shown in Fig. 4.4 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, 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 and the hierarchical token bucket, which is a classful queuing discipline.

## 4.4 Evaluation

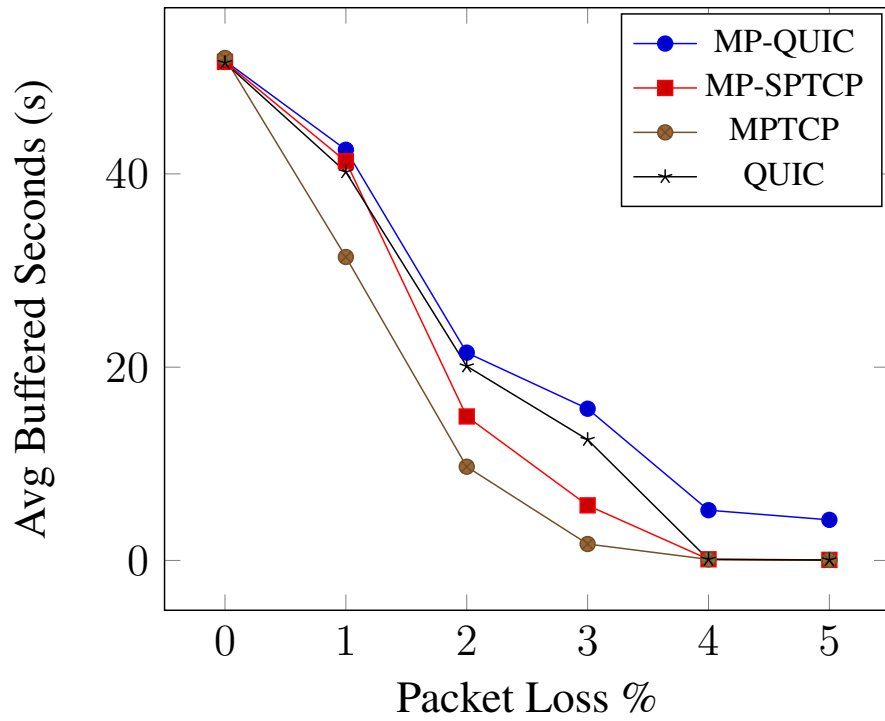
The plots below are the result of experimentation in the physical testbed. For each plot, the packet loss rate was set at the discrete points of 0%, 1%, 2%, 3%, 4% and 5%. Each data

point represents the average over a series of 20 runs at each configuration. MP-QUIC refers to this studies' solution presented in the figures. MP-SPTCP refers to an implementation using bandwidth thresholds presented by Nam et. al [31] instead of RTT and retransmission thresholds. Lastly, in the figures, MPTCP and QUIC refer to results from each transport protocol's default implementation.

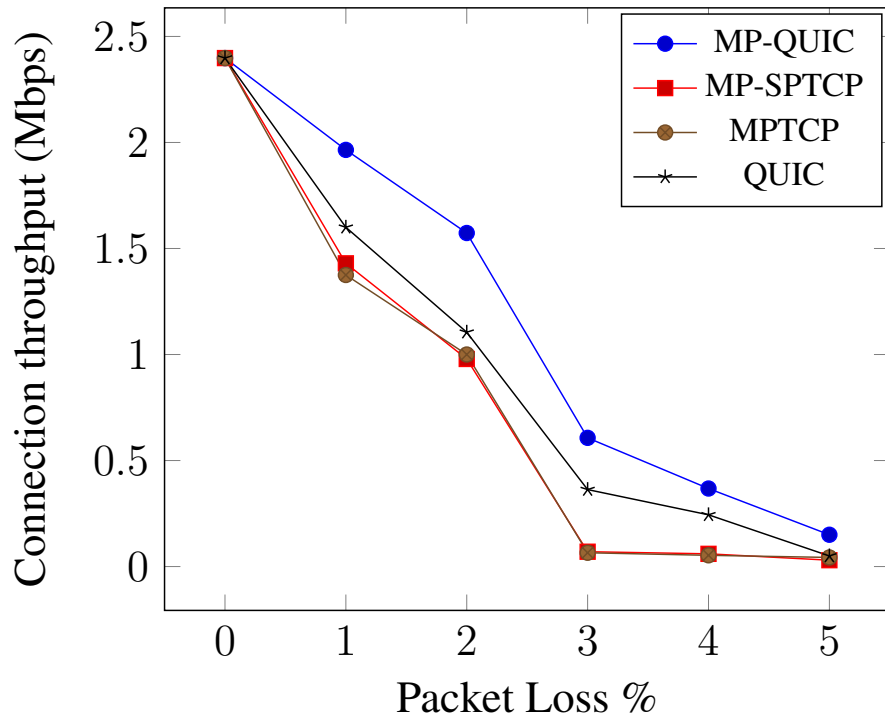
Shown in Fig. 4.5a is a comparison between the average number of seconds of video in the buffer vs. packet loss rate for each algorithm. This plot does not show the quality of the buffered seconds of video. Evaluating Fig. 4.5a with Fig. 4.5b when the packet loss rate is zero, each algorithm performs comparably. This is expected since each algorithm is able to operate with no packet loss. These results are consistent with previous studies [15, 7, 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. Furthermore, MPTCP relies on timeouts for switching paths, unlike this modified version. Except for MP-QUIC and QUIC, when packet loss rates reached 5%, Fig. 4.5a and Fig. 4.5b illustrate that standard MPTCP and MP-SPTCP were unable to maintain usable bandwidth over the lifetime of a connection.

Illustrated in Fig. 4.6a, MP-QUIC and QUIC spend significantly less time stalled with an empty buffer than standard MPTCP and MP-SPTCP. This protocol even outperforms canonical QUIC. This is attributed to the low latency of the protocol being leveraged quickly and effectively with the SDN controller providing timely network statistics. Additionally, the start-up decision engine made a significant difference when experiencing high packet loss rates when stalls were imminent, Fig. 4.6a.

As mentioned in the beginning, one factor affecting QoE is the size of MPTCP's reorder buffer and the number of retransmissions MPTCP has to make during a connection for unbalanced connections. In Fig. 4.7a, at 4% and 5% loss rates, MPTCP and MP-SPTCP

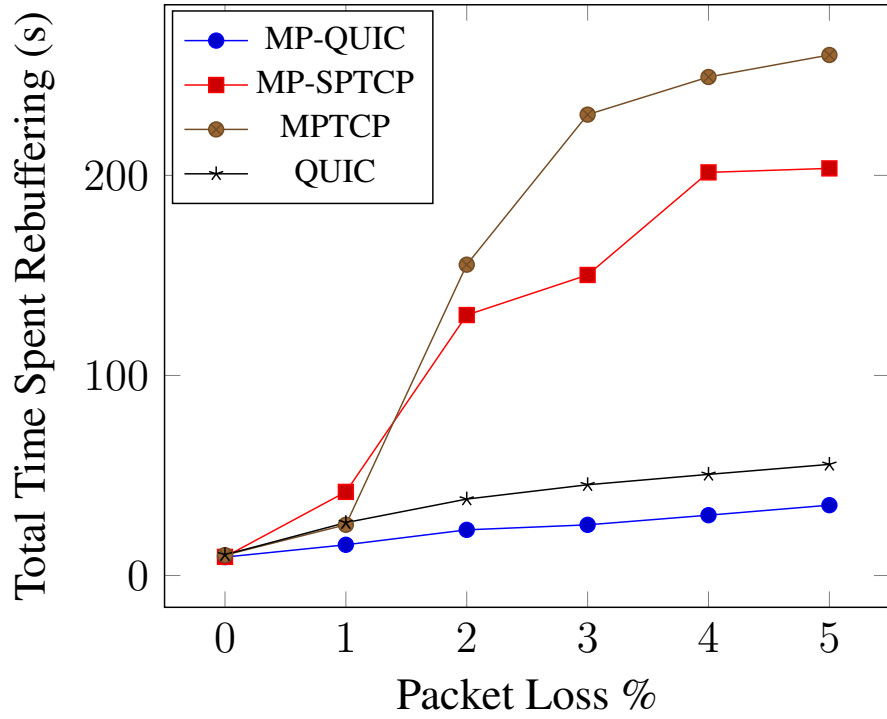


(a) Average buffer size

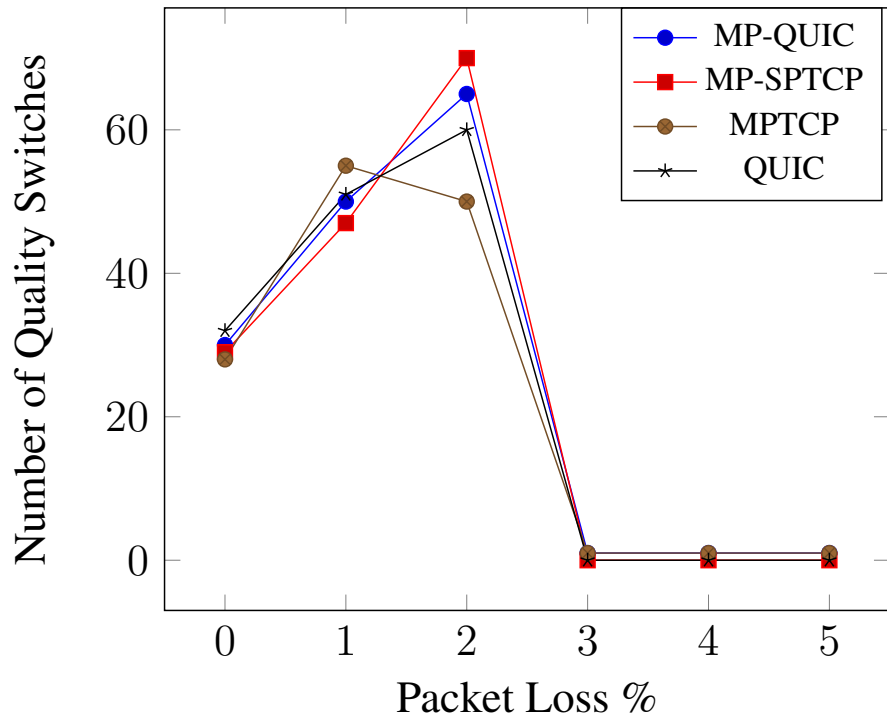


(b) Average connection throughput

Figure 4.5: Various performance metrics for a given packet loss rate percentage.

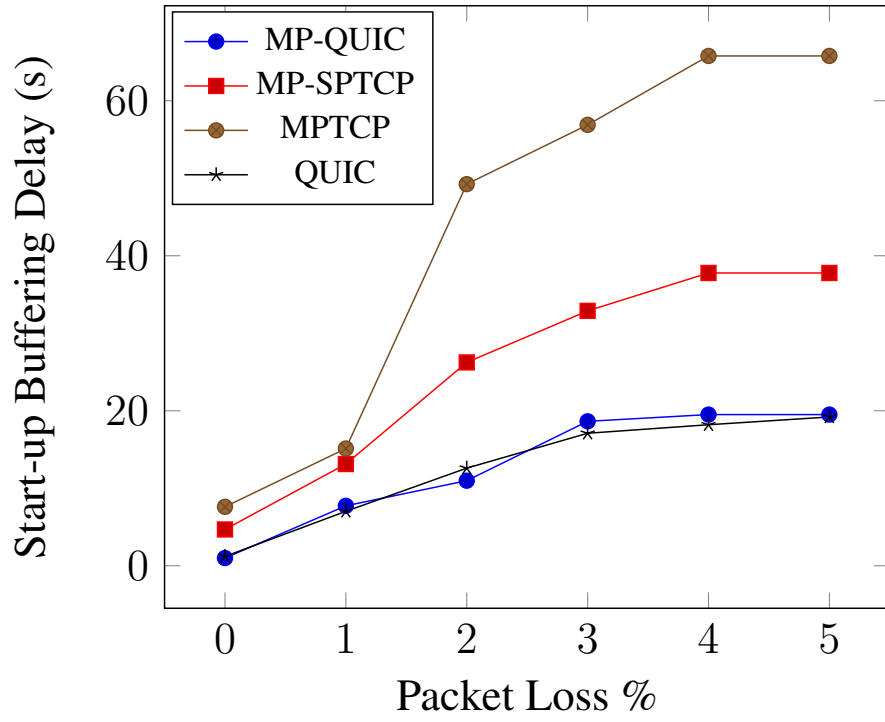


(a) Average time video playback stalled

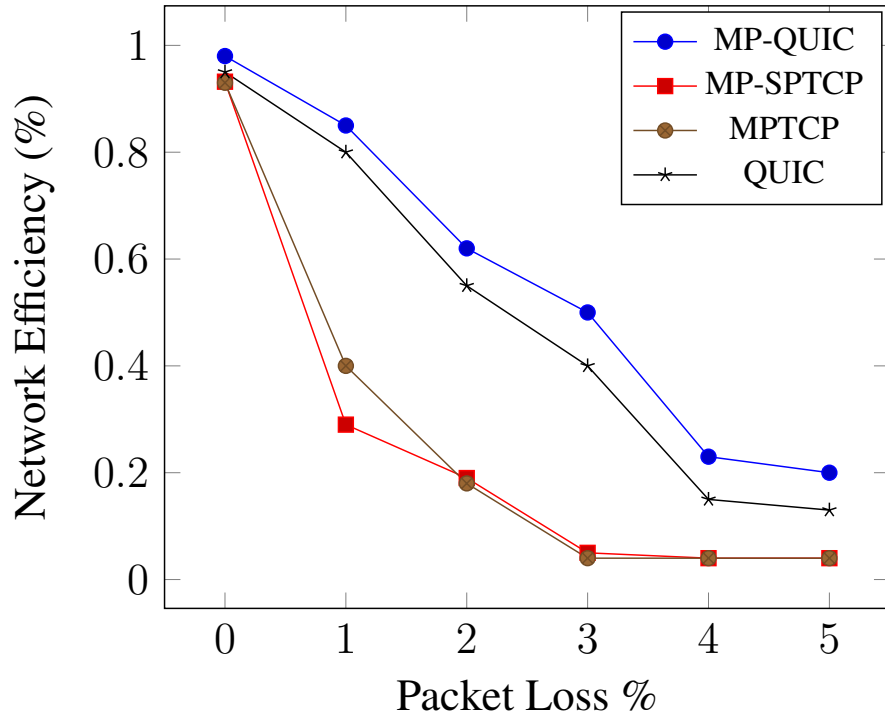


(b) Average number of quality switches

Figure 4.6: Various performance metrics for a given packet loss rate percentage.



(a) Average start-up buffering delay



(b) Network usage efficiency

Figure 4.7: Various performance metrics for a given packet loss rate percentage.



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. A conclusion is reached that this is the case for a few reasons. One reason is that MPTCP was 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. 4.7a, 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 video quality right after start-up. MPTCP and MP-SCTCP at 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. 4.7b 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.

## 4.5 Discussion

Proposed in this study are techniques for improving omnidirectional ABR video performance using MPTCP/QUIC over an SDN architecture. The focus is on improving the start-up experience and overall streaming by designing two strategies to leverage SDN. The use of SDN enabled the design of an algorithm to actively switch between MPTCP and QUIC networks. By using MPTCP/QUIC over SDN, the experiments were able to leverage path diversity for increased performance in low-loss networks and performed 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 the algorithm outperforms standard QUIC, MPTCP and MP-SPTCP in increasingly lossy networks by 15% or more in many key streaming metrics. These results were for a controlled proof of concept and may vary when multiple clients compete for bandwidth capacity.

Limitations to this study include the scale of the experimentation. Future chapters look to expand this study by using a larger testbed with Planet Lab, through examining other MPTCP path management algorithms to further compare this work, and by obtaining results under wireless conditions where there more than two networks available.

## **CHAPTER 5**

### **DYNAMIC TRANSPORT LAYER SELECTION AT SCALE USING SDN**

In the previous chapter, an extensive investigation was conducted of dynamic transport selection when streaming video content. However, it remains to be seen if the idea is as effective at scale. SDN in particular is further utilized in this chapter to address problems of traditional static network architecture. SDN decouples decisions about where data is sent from the hardware, thereby enabling flexible network configurations. SDN was leveraged in this study to dynamically control QUIC and MPTCP available paths in a way that is invisible to transport and application layers in a campus wide network. Presented in this chapter is a method to improve video streaming at scale leveraging MPTCP, QUIC, and SDN.

The research described in this chapter aims to expand on the preliminary work of the previous chapters paired with the development of a learning adaptive transport selection framework for use in multipath networks. Specifically, the next steps will entail using simulation-based networking and applications to design network-assisted learning adaptive transport protocols based on real-world network performance. Fig. 5.1 illustrates a high-level flow diagram for ABR streaming within this context.

Presented here is the design methodology to be used to generate multipath enabled neural network-based adaptive bitrate algorithms. The RL-based neural network will leverage MPTCP, and QUIC to improve video streaming in multipath networks. The first step outlines the elemental procedures needed to train the neural network and incorporate the improvements necessary to the underlying training algorithms to support multipath enabled clients.

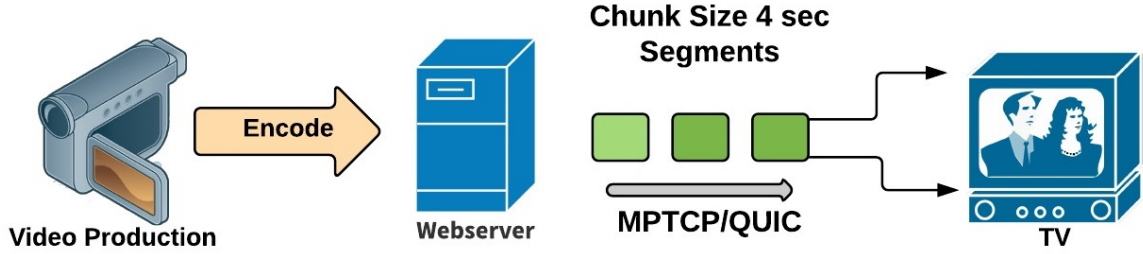


Figure 5.1: Example adaptive transport flow diagram

## 5.1 System Model

Described in this section is a method to improve 360-degree video streaming at scale leveraging MPTCP, QUIC, and SDN. The first part of the design called differential delay minimization, was developed using an SDN application that manages delay constraints of each network path. This was found to reduce the average multipath differential delay experienced at the client by up to 40% in large campus networks. The second part of the design, called flow allocation, was used in conjunction with differential delay minimization. Furthermore, it uses MPTCP's receive buffer size, current bitrate, and the number of retransmits each subflow experiences to predict overdue video given network constraints and current demand.

### 5.1.1 Differential Delay Minimization

The primary goal of differential delay minimization is to reduce the delta in RTT between subflows. The importance of this idea was synthesized from research findings [7] studying the effects of varying network bandwidth between flows. These researchers concluded that non-matching networks often experience poor performance. Single path networks can often outperform multipath networks if they are not constructed intelligently. MPTCP, in particular, can experience wildly varying delays when trying to manage different flow buffers sizes, causing data to arrive in bursts. This effect is problematic for 360-degree video streaming because of its low delay tolerances. In high error, low bandwidth networks

multipath techniques can be reluctant to react thereby causing video buffer starvation. Insights from previous studies on QUIC demonstrate that the responsive nature afforded in UDP-based protocols can be exploited to provide increased performance for video streaming sessions [27].

---

**Algorithm 3** Connection Controller

---

```

1: procedure DIFFERENTIAL DELAY CALCULATION
2:    $appThres \leftarrow$  threshold for the change in RTT
3:    $\Delta rttList \leftarrow$  RTT compared to lowest available
4:    $rttList \leftarrow$  live avg RTT for each available path
5:    $numSubFlows \leftarrow$  number of subFlows available
6:
7:   Connection Selection:
8:   if  $numSubFlows \equiv 1$  then use QUIC protocol
9:   if ( $numSubFlows \equiv 2$ ) and
      ( $\Delta rttList < appThres$ ) then use MPTCP
10:  else
11:    if  $numSubFlows \equiv 2$  then use QUIC on
12:      lowest RTT path
13:  if  $numSubFlows > 2$  then
      using  $rttList$  if any path  $\Delta rttList > appThres$ 
      remove path using SDN if two or more paths
      remain use MPTCP otherwise use QUIC on
      lowest RTT

```

---

In an effort to mitigate the issues described above, a unique model aimed at reducing the differential delays for clients is proposed (reference Algorithm 3). The main SDN controller of the campus network keeps an up-to-date list of the available paths throughout the network. When a new node joins the SDN network, as the controller has a global view, it will assign appropriate flow table entries at the local switches. The controller will rank flows based on the similarity of their path RTT. The switches are laid out in a hierarchical fashion where each local network switch will run a average calculation and transmit a resulting synopsis to the controller above itself within the network layout. From the SDN application perspective, it will request a connection based on the the application requirements (e.g., error rates, latency, etc). The differential delay minimization calculation formulated

in this section is used and leveraged with the flow allocation unit described in the next subsection to initiate a multipath connection that has optimal matching characteristics.

As mentioned in the previous chapter, QUIC outperforms standard TCP in many low bandwidth scenarios [36]. When the SDN controller presents two or more networks to the client, the SDN application compares the differences in the average RTT for each network path. Using that difference, combined with the estimated channel throughput over the last second, the application optimizes its decision of selecting which path to use. If the differential delay is larger than 9:1 [58] (e.g. 90 Mbps and 1 Mbps) the algorithm ignores the slower path and uses only the QUIC protocol on the lowest RTT network. This implementation of SDN, MPTCP, and QUIC has implications for short flows - defined as flows that terminate before MPTCP exits the slow start phase and large flow connections - defined as flows that enter MPTCP's congestion avoidance phase. Drawbacks of this method, however, include networks that do not allow UDP-based streaming nor do they support SDN networks. For experimentation purposes, this situation is not considered as this research is focused on delivering video to networks that do support these options. This effect can be mitigated however in real-world use by falling back to traditional TCP-based streaming techniques.

### 5.1.2 Flow Allocation

The second part of the design called flow allocation can be summarized as follows. A calculation of the subflows needed to satisfy the video quality constraints of the application and multipath data allocation were determined by minimizing a global linear approximation for a stable network. Using SDN and MPTCP's *MPTCP\_INFO* provided the controller and applications with statistics at the MPTCP-level (e.g., state, retransmits, tcp\_info for all other subflows, etc.). Specifically, each flow is given a priority in terms of most likely to meet the bitrate requirements. To eliminate the potential for a high number of retransmissions and duplicate ACKs in unbalanced networks, the system selectively drops QUIC packets that

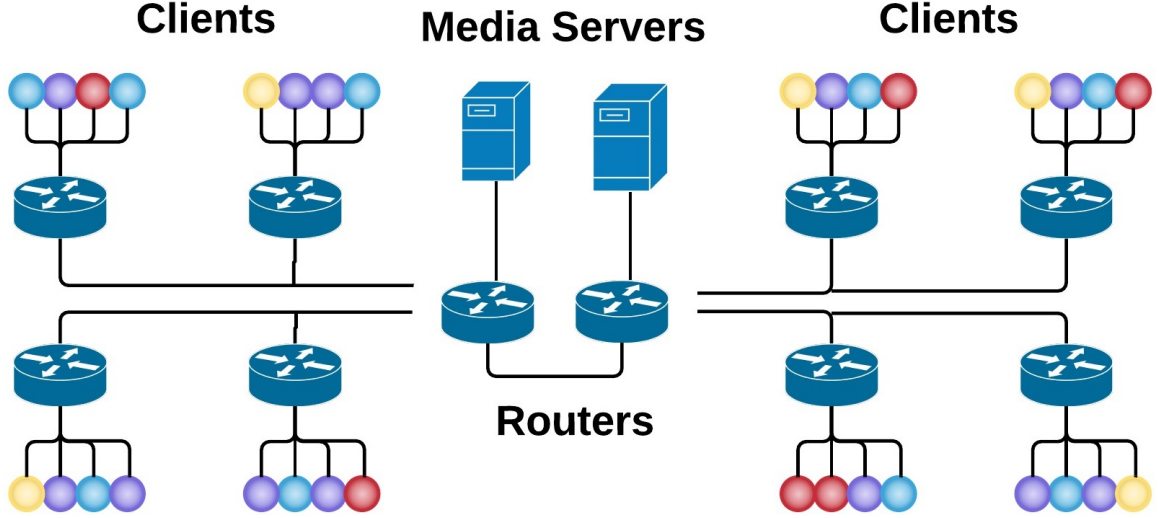


Figure 5.2: Network topology

are overdue. Historically, packets could be buffered too long due to being delivered out of order, in turn arriving at the application late.

When the client’s playback buffer was critically low, there was only one path available, and the connection was started with MPTCP, the system then sends 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. This technique employs path diversity techniques which aid in dealing with network unpredictability. The flow allocator is also able to avoid the three RTTs usually required in connection establishments. This was especially useful in networks with a consistently high RTT. A drawback of this method, however, is the potential of frequent oscillations between MPTCP and QUIC. This situation can be mitigated by dropping the connection instead of keeping it alive.

## 5.2 Experiment Setup

Experiments for this chapter were conducted over a campus style network. Encoded ABR 360-degree videos were captured with a Samsung Gear 360. The content was encoded with x264 [56] at 40 different bitrates between 5 kbps and 50 Mbps. To compare the performance of MP-QUIC, MP-SPTCP, and MPTCP with this implementation, the following

experiments were performed while varying the network size from 100 to 1,300 nodes. Each and every client and server nodes were multipath enabled. The overall network bandwidth was adjusted as follows. From the start of the experiment to  $t = 240s$ , the available bandwidth was 10 Mbps. At  $t = 240s$  until  $t = 480s$ , the available bandwidth was reduced to 5 Mbps. At  $t = 960s$ , the available bandwidth was increased to 10 Mbps until the end at  $t = 1920s$ . The experimental setup in this chapter are consistent with previous chapters to allow for the comparison of results on a similar platform, but with the adjustment of parameters to fit the application performance requirements of various networks. Measurement studies of prior work inform the client to server RTT was determined to be realistic when set to approximately 150ms and a 4s segmentation and was therefore chosen for the DASH algorithm [22]. The link-loss rates were not artificially changed for this chapter, however, if buffers overflowed, packets may be dropped.

Comparing network performance, the following parameters were selected as explained in the previous chapter, however with additional domain specific parameters which are unique to this design:

1. *Differential buffering delay*; the difference in the delay between individual multipath links.
2. *Playback buffer level*; the number of seconds of video stored in the client buffer ready for playback.
3. *Average connection throughput*; the average throughput over the last second of the connection.
4. *Number of quality switches*; the number of times during the connections lifetime that the quality of the video was 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 was used.

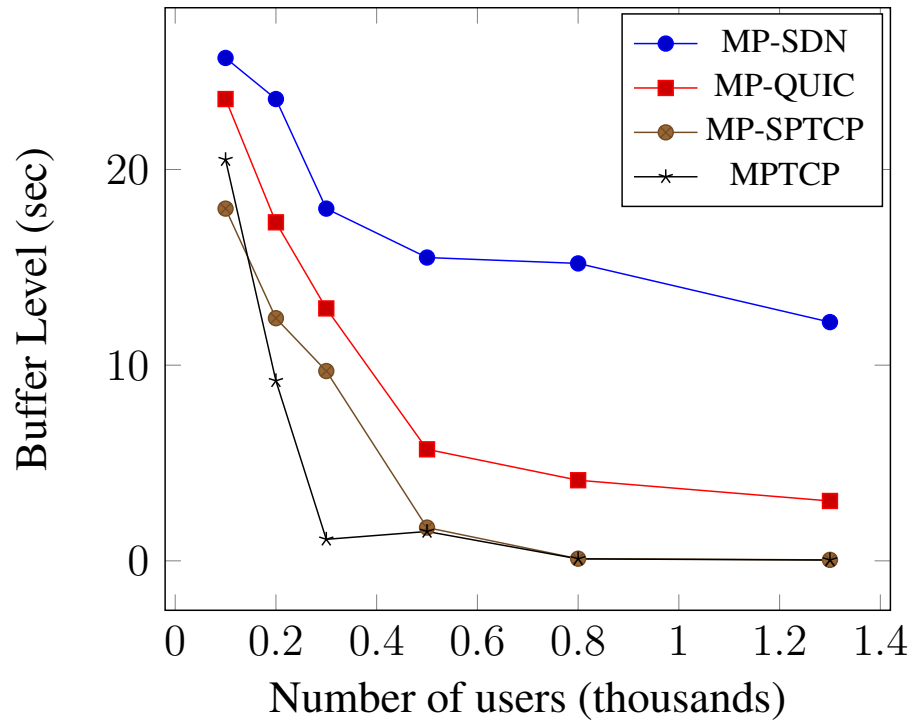
The SDN application queries the SDN controller in the network to retrieve the average RTT delay along each path available. The application then interfaces with the Floodlight controller in the Mininet SDN network.

The layout of the testbed shown in Fig. 5.2, supports QUIC, MPTCP, MP-QUIC, and MP-SPTCP. The testbed consists of a range of client/server nodes, routers, and a Floodlight Mininet controller. The MPTCP congestion control algorithm used was wVegas. The network simulator used was Network Simulator 3 (NS-3).

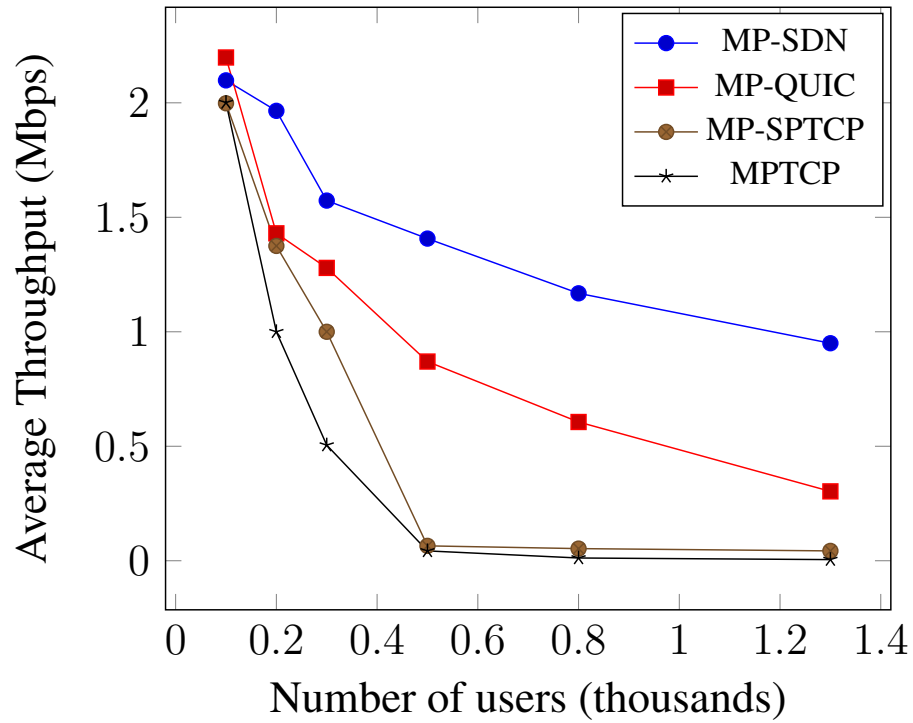
### 5.3 Evaluation

The diagrams below illustrate the results of experimentation which display aggregate averages across all users. For each plot, the number of users were 100, 200, 300, 500, 800, and 1,300, respectively. MP-SDN refers to this chapter's solution presented in the figures. MP-QUIC [27] refers to prior chapter's implementation which leverages QUIC and SDN in the figures. MP-SPTCP [31] refers to described implementation of using bandwidth thresholds instead of RTT and retransmission thresholds. Lastly, MPTCP refers to its standard implementation without modification.

Shown in Fig. 5.3a, is a comparison to the average number of seconds of video in the buffer vs. the number of users streaming video on the network. This figure does not reflect the quality of the video stored in the buffer at that time. To facilitate that comparison one should assess Fig. 5.3a with Fig. 5.3b; when there are a low number of users on the campus network each technique performs comparably. This was expected since each algorithm was able to operate with little to no network congestion. These results are consistent with previous studies [27, 28]. As the number of users increase, MP-SDN and MP-QUIC outperform MP-SPTCP and MPTCP. This was due to the fact that MP-SDN provides symmetric flows for users instead of simply choosing any available flow when the network reaches capacity. MP-SPTCP relies only on TCP and was not aided by QUIC and SDN

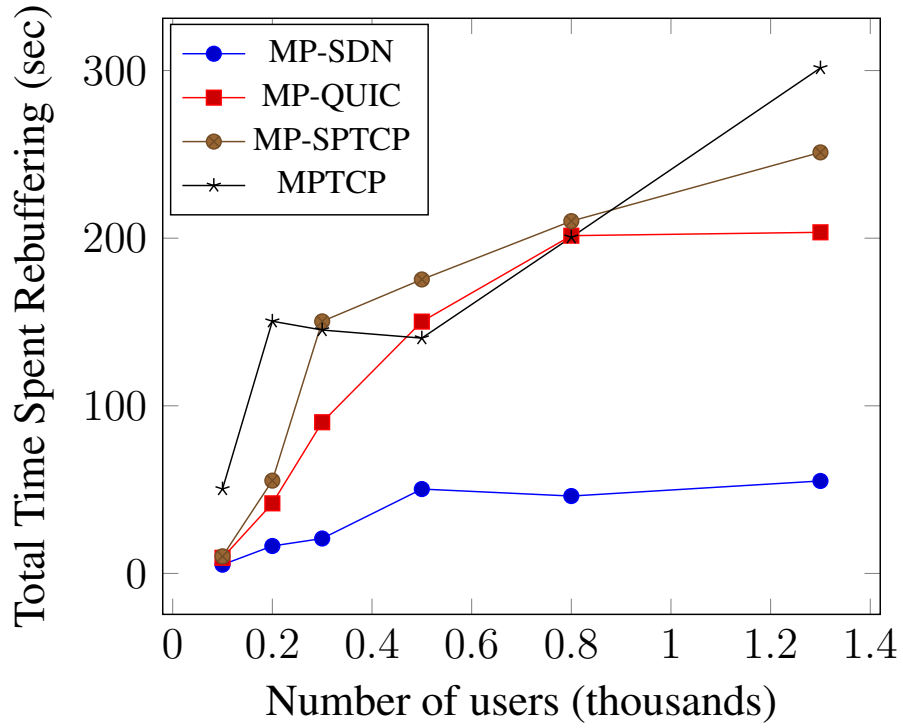


(a) Average buffer level

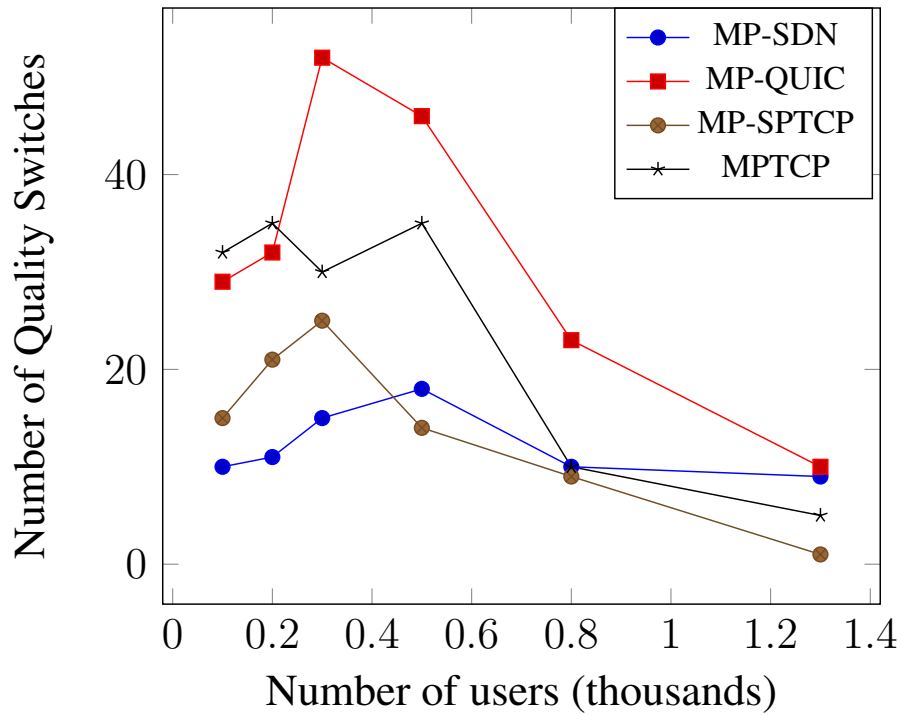


(b) Average user throughput

Figure 5.3: Various performance metrics for a given number of users.

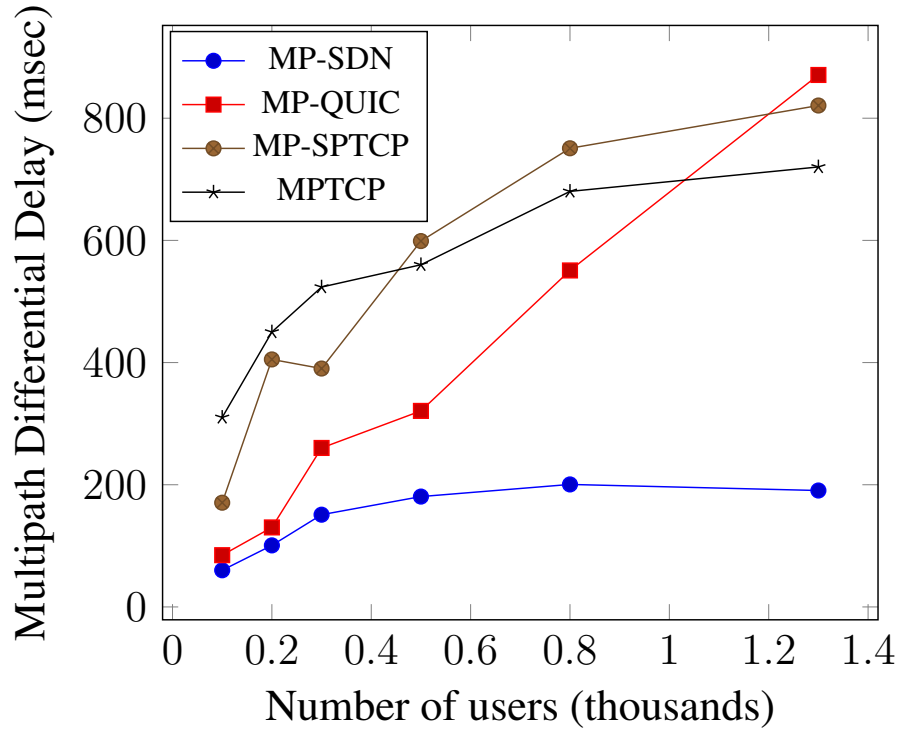


(a) Average time video playback stalled

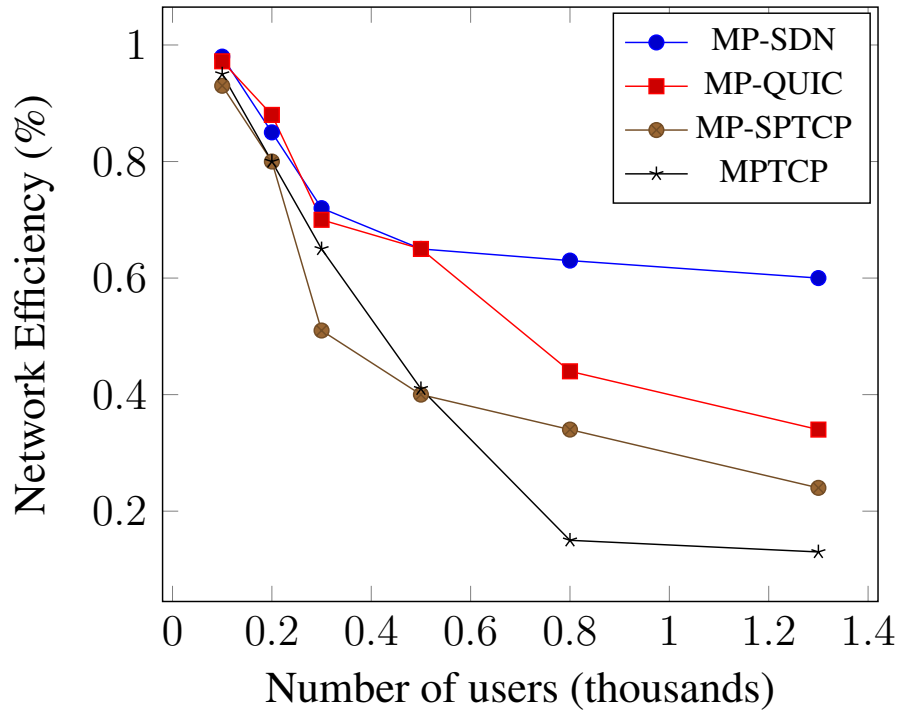


(b) Average number of quality switches

Figure 5.4: Various performance metrics for a given number of users.



(a) Average multipath differential delay



(b) Network usage efficiency

Figure 5.5: Various performance metrics for a given number of users.

flow control, which was a limitation in highly congested networks. Further, MPTCP and MP-SPTCP rely on timeouts when switching paths, unlike MP-SDN and MP-QUIC. In the testbed, when the number of users surpassed 300, MP-SPTCP and MPTCP (see Fig. 5.3a and Fig. 5.3b), illustrate that the standard MPTCP and MP-SPTCP were unable to maintain usable bandwidth over the lifetime of the connection.

In Fig. 5.4a, as the number of users increased, each transport protocol, except for MP-SDN, spent an increasing percentage of time with a stalled empty playback buffer. MP-SDN even outperformed MP-QUIC, which reverts to canonical QUIC in highly congested networks. This was a result of design decisions for the SDN applications and controllers to take into account the jitter experienced at each user's multipath flow from a system wide standpoint.

As mentioned in the introduction, a few issues affecting MPTCP's scalability and performance include the size of MPTCP's reorder buffer and a large number of retransmissions that occur during a connection in unbalanced networks. In Fig. 5.4b, each algorithm followed a similar trend in which they each began with a low number of switches. After increasing the number to approximately 500 users, a peak was reached and then the number of quality switches began to decline. These results are in-line with the previous studies [46, 27], which indicate there are a low number of switches when the network is not congested and the highest bitrate stream is used. However, after congestion builds from the increased number of users, the number of quality switches increased as it approaches a saturation point. In many cases, this point is dictated by the physical layout of the network itself. After the saturation point, the network is unable to support a high bitrate for all users and the network stabilizes at a lower quality bitrate for each connection. As a result, the number of quality switches for each user declined. MP-SPTCP was unable to make any quality switches and remained at the lowest quality level during the entire duration of the 1,300 user experiment.

As shown in Fig. 5.5a, MP-QUIC and MP-SDN have a markedly better multipath dif-

ferential delay with a low number of users. MP-SDN continued to produce favorable differential delays within a large network, whereas MP-QUIC diverged as congestion increased. Concluding from the analysis, this was a result of the overhead of MPTCP, MP-QUIC, and MP-SPTCP when establishing a full mesh of connections without considering their differential delay. Not shown in figures due to brevity, MP-QUIC started up slightly faster than MP-SDN and MP-SPTCP, but with a lower overall bitrate. In larger networks, MP-QUIC and MP-SDN began playback more quickly and at a higher bitrate than traditional MPTCP and MP-SPTCP. Finally, Fig. 5.5b reveals that MP-SDN and MP-QUIC utilized more of the available bandwidth than the other configurations, however, MP-SDN outperformed MP-QUIC's efficiency in dense congested networks.

## 5.4 Discussion

This chapter demonstrated techniques for developing, evaluating, and scaling a 360-degree ABR video testbed using MPTCP and QUIC over an SDN architecture. Performing differential delay minimization for multipath capable devices was a primary focus for the study. The use of SDN enabled an algorithm design that not only actively switches between MPTCP networks and QUIC, but also modified MPTCP's paths real time using this studies' flow allocation scheme. By using MPTCP and QUIC over SDN made for increased performance in both small sparse networks and in large congested networks. As a result, the average differential delay decreased by 40% or more when using MP-SDN as compared with MP-QUIC, MPTCP or MP-SPTCP in campus networks larger than 500 users. Playback stall time also decreased by more than 50% in networks larger than 300 users. Furthermore, MP-SDN utilized the network 30% more efficiently in networks with 800 or more users. These results demonstrate that the presented algorithm outperforms MP-QUIC, MPTCP, and MP-SPTCP when scaling the number of users that have access to the network by 20% or more in many key streaming metrics.

A primary limitation to this study includes the type of network chosen, as the results

may differ when the physical network topology dramatically changes. Efforts in expanding this study are explored in later chapters using various network model types. Finally, a plan has been made to obtain results under wireless conditions in which there are additional network paths.

## **CHAPTER 6**

### **REINFORCEMENT LEARNING ADAPTIVE TRANSPORT IN MULTIPATH NETWORKS**

In the previous chapter, an analysis was administered comparing the performance of various dynamic transport selection protocols when operating at the scale of a large managed campus network. Comparing relevant parameters of streaming video content was of particular interest and was explored. The chapter focuses on the innovation needed to create an implementation that will not only operate at scale efficiently, but also operate effectively in varying types of networks. The previous concept started with a small scale testbed described in Chapter 4. The idea was further extended in Chapter 5 with the caveat of requiring a managed SDN to operate with optimal efficiency. Presented here are methods addressing fundamental limitations of previous chapters; in particular when scaling video delivery as there is a tendency to have an over-reliance on centrally managed networking techniques. This chapter will focus on improving video streaming at the client using creative scaling techniques that do not rely on centrally managed networks (i.e., SDN), by utilizing MPTCP, and QUIC with the addition of SAND, and RL.

Within the ABR space, efforts to improve its performance have reached new areas of research. In particular, RL is now being used and its performance is being evaluated for its effectiveness. Despite this fact, many studies have limited their work to simulated synthetic network traces [47, 48], and as a result, rely on state spaces small enough to store a value per state in an allocated computer memory array. Unfortunately, this decision limits deployment scalability. When using RL to train ABR algorithms for use in single path networks, high-bandwidth video requirements and multipath networks are oftentimes not considered [49]. RL can aid in the design and implementation of algorithms that are able to adjust more accurately and efficiently than simple heuristic based designs of prior efforts.



As mentioned in previous chapters, leveraging SAND provides a unique mechanism for providing video streaming advancements within existing networks [14]. SAND presents unique mechanisms for providing context-aware quality of service and experience signaling which are utilized to operate in multipath networks. Using SAND-based networks to more intelligently encode and transmit multimedia data in real time is essential [60], however the volume of data generated in 360 VR environments within multipath networks need to be considered. However, like other studies, they do not consider the volume of data generated in 360 VR environments nor do the authors address multipath considerations.

An emerging standard to improve VR interoperability, referred to as Omnidirectional Media Application Format (OMAF) (see Fig. 6.1), has been leveraged for the system design [61]. Current research efforts in this area have not focused on the performance of 360 VR media in multipath networks with competing traffic. A unique approach to remove limitations of prior ABR efforts in multipath networks are addressed by designing learning algorithms without having to rely on traditional networking models or suboptimal client heuristics. Through a testbed of multihomed devices leveraging SAND, MPTCP, and QUIC; extensive testing illustrate this technique is able to support 360 VR media streams effectively.

## **6.1 System Design**

In this section, the design methodology used to generate multipath enabled neural network-based adaptive bitrate algorithms are discussed. The RL-based neural network leverages MPTCP, QUIC, and SAND to improve 360 VR video streaming in multipath networks. This chapter begins with outlining the elemental procedures incorporated to train the neural network used for experimentation and then describes improvements to be made to the underlying training algorithms to support multipath enabled clients.

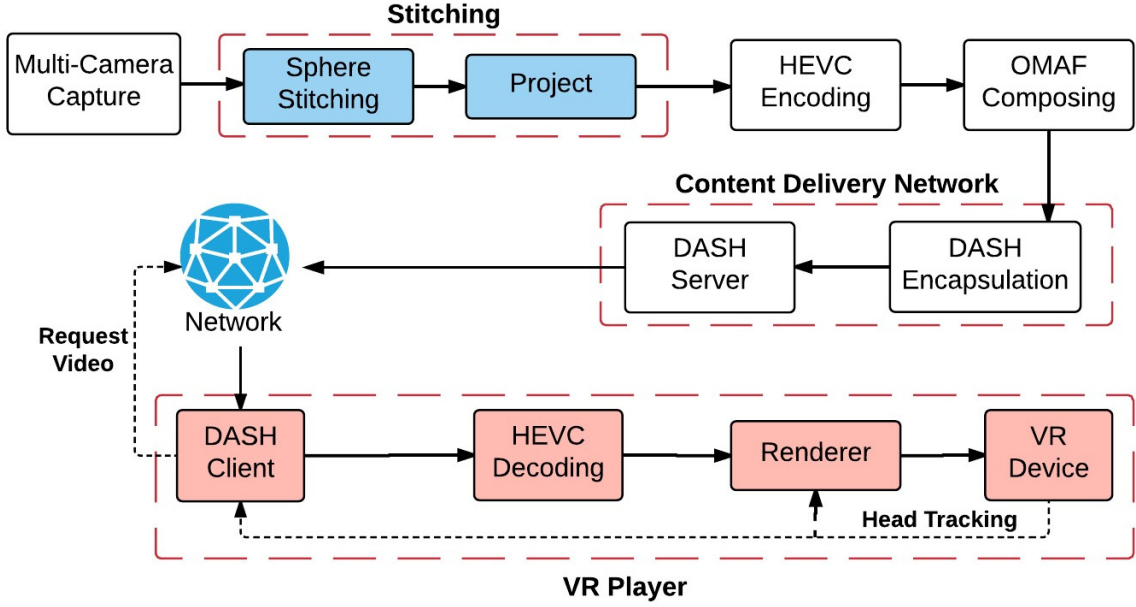


Figure 6.1: DASH OMAF architecture

### 6.1.1 Training Procedure

Developing flexible and adaptive ABR algorithms using RL requires clever exploration of mechanisms for training. In an ideal setting, the learning agent would use a variety of streaming clients and actual streaming sessions to capture every nuance of the video streaming experience. This process of training however, is slow and would not be practical in real-world use. As a result, the algorithms are trained using a simulator that models the critical aspects of the streaming experience.

The chunk level simulator [62] outlined and described in Chapter 5 that models network effects while pairing with actual streaming clients was expanded upon in this study. The use of this simulator allowed for a shortened training duration for the ABR algorithms to learn from multiple real-world network traces. The simulator models playback buffer fill/depletion rates, download times, and rebuffering events based on the actual network traces provided.

Recalling differences detailed in Chapter 2, recent breakthroughs in training deep neural networks demonstrate that novel end-to-end RL techniques, including DQN and A3C

models, learn very differently when utilizing large scale neural networks. Each technique has a domain in which it operates effectively. Generally comparing each technique only makes sense once there is a particular problem to be solved. There are techniques to combine DQN-like features within an AC3 model providing performance benefits. Models with a limited number of actions have been shown to be more efficient using DQN. A3C incorporates Q-learning and policy iteration methods into a single technique. This allows for efficiency gains given the complexity of video streaming and the multitude of adjustable features. AC3 has been shown in the literature by Google’s DeepMind project to thrive in this problem space. Consequently, the RL training procedure begins with a learning agent at each decision point. This is determined by a system designer to make an assessment about the status of the system in an effort estimate how its choices have affected the streaming experience. Consequences of each decision is reflected in a score from the user or objectionable streaming measurements which are used to encourage the highest score the model can achieve in the long term. The long term score is formed based on an optimization problem. The training algorithm state parameters are as follows: the number of chunks left for download, previous chunk bitrate, previous chunk download duration, previous chunk throughput, encoded chunk size and buffer status. The score is primarily a function of three metrics: bitrate utility, smoothness playback penalty, and rebuffering penalty.

While using a chunk level simulator captures a majority of dynamics of the streaming experience, there are underlying effects of lower layers that may interfere with the performance of real-world networks that are not readily addressed thus far. Exploring one example in particular, when downloading a chunk after long pauses, TCP may revert to TCP slow-start restart [63] thereby under-utilizing available bandwidth. Due to results found in literature and described in Chapter 5, when TCP slow-start is disabled, the clients are able to utilize the full bandwidth available. Because of these findings, slow-start restart was disabled for all of these experiments. The purpose of this decision was to eliminate the

effects of lower network layers as much as possible for a fair comparison of individual algorithms without influences from the operating system or platform they are being tested on. Recognizing that this may cause bursty download behavior, this process is necessary as it normalizes the testing environment for algorithm comparison. As discussed in the previous chapter, this type of modeling is more accurate, however it can induce unneeded training complexity into the system for marginal improvements. In reality, every simulation model abstracts away minor features of systems, however the RL training process is generalizable given the large set of varied training data and proves to perform well.

Formalizing the reward function in Eq. 6.1.  $R$  at step  $i$ , customizable weights were used  $x, y, z$  that can be modified based on application importance. *Smoothness Penalty* ( $S$ ) is the magnitude of the bitrate change from one chunk to the previous chunk. *Rebuffering Penalty* ( $Re$ ) is the stall duration. *Bitrate Utility* ( $B$ ) reflects a higher utility with a higher bitrate.

$$R_i = x * B_i - y * S_i - z * Re_i \quad (6.1)$$

### 6.1.2 SAND Assisted Controlled Unfairness

Outlined here is the inclusion of SAND into the algorithm along with a description of how it was used to assist in driving the streaming experience. As discussed in the introduction, SAND provides real-time operational communication between servers, network components, clients, and content delivery networks with regards to DASH performance. SAND supports context-aware quality of service signaling in which is extended to operate in multipath networks. SAND architecture is explained in Fig. 6.2. The following list explains the figure's components and messages:

- DASH-Aware Network Elements (DANE): have intelligence about DASH formatted objects and may prioritize, parse or even modify them.
- Regular Network Element (RNE): unaware of DASH content and treat video delivery as any other content.

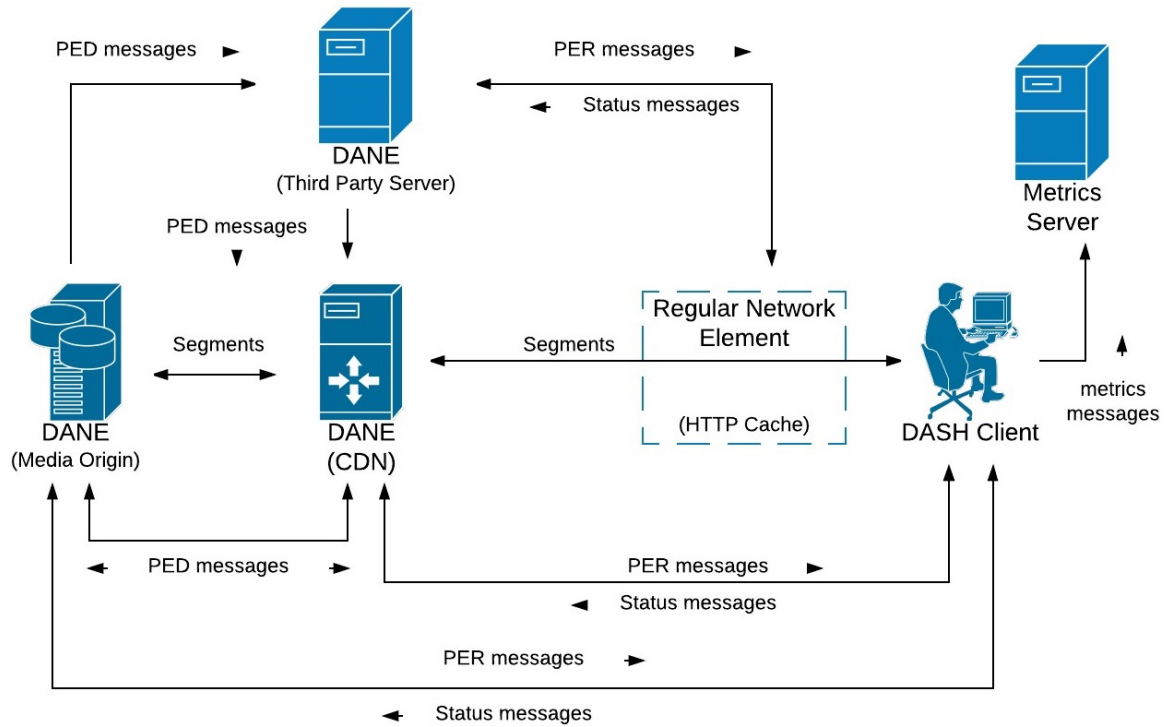


Figure 6.2: Example SAND architecture

- Metrics Server: gather metrics from DASH clients.
- Dash Client: is able to stream DASH content.
- Status Messages: messages from clients to DANEs.
- Parameters Enhancing Reception (PER): messages that are sent from DANEs to clients.
- Parameters Enhancing Delivery (PED): messages that are exchanged between DANEs.
- Segments: DASH media content.

Implemented here is only an extension of portions needed from the SAND architecture as it is still in the standardization process; see Fig. 6.3. SAND provides a mechanism for providing controlled unfair delivery of content, as not all content or network paths are handled equally. Network capacity withstanding, if network paths are content-aware, they are able to make more intelligent routing and/or request decisions. For experimentation

purposes, DANEs were used to send SAND signals to clients providing network fluctuation warnings before clients experience them. Anticipation messages were attached to client requests so intermediate DANEs avoid evicting future useful video for improved cache hit rates. By extending the neural training to receive a second set of inputs based on multipath data delivery statistics from MPTCP’s *MPTCP\_INFO*, the neural network is provided with statistics at the MPTCP-level (e.g., state, retransmits, tcp.info for all subflows, etc.). This makes the neural network multipath aware while taking into account bitrate decisions.

Leveraging the trained neural network indirectly reduces the differential delays for clients (reference Algorithm 4). The trained neural network receives PER messages from the SAND network, *MPTCP\_INFO* statistics, and client state statistics discussed in Section 6.2. Optimizing around the tunable reward function, this algorithm is able to make informed decisions with more information than previously available to clients. MPTCP options keep an up-to-date list of the available paths along with their statistics. The PER messages from the DANE device provide anticipated requested information to the clients so that it may push data to the client prior to a request. Based on client status messages, DANE devices can cancel or generate higher quality bitrate push decisions.

---

**Algorithm 4** Neural ABR Controller

---

```

1: procedure PATH AND BITRATE SELECTOR
2:   perMess  $\leftarrow$  PER message information
3:   mptcpInfo  $\leftarrow$  MPTCP-Info statistics
4:   clientState  $\leftarrow$  client state statistics
5:   numSubFlows  $\leftarrow$  number of subFlows available
6:
7:   Connection/Bitrate Selection:
8:   if numSubFlows  $\equiv$  1 then use QUIC protocol
9:   if (numSubFlows  $\equiv$  2) and
      (neuralAlgoDecision) then use MPTCP
10:  else
11:    if numSubFlows  $\equiv$  2 then use QUIC on
12:      lowest RTT path
13:  if numSubFlows > 2 then
      use output to neuralAlgoDecision

```

---

SAND natively encourages without explicitly training to model balanced RTT network paths. Prior testing has shown that balanced RTT networks produce better multi-path streaming results. If there is only a single active network available to the client when the controller presents the available paths, then the application defaults to using the QUIC protocol. QUIC outperforms standard TCP in commonly video streaming scenarios [36]. When there are two or more networks available, the neural network balances its trained priorities for each path and optimizes its decision of selecting which to use.

## 6.2 Experiment Setup

TensorFlow’s [64] software library is utilized in support of the neural network implementation. The network simulator fed traces to a one-dimensional convolution layer with 256 filters each with a size of four and with a stride of one. Training the neural network took approximately six hours however, this offline training should be an infrequent occurrence depending on problem domain stability.

Experiments were conducted in the testbed shown in Fig. 6.3. The encoded adaptive 360 VR videos were three minutes in length and were captured with a Samsung Gear 360. The content was encoded with x265 at 20 different well-spaced bitrates between 1 Mbps and 10 Mbps. The performance of MP-SDN, MP-SPTCP, and MPTCP was compared with a replicated MP-DANE implementation. Details of the comparison algorithms are discussed in Section 6.3. Experiments were performed while varying the number of concurrent streams from one to six. All network nodes were SAND, MPTCP and QUIC enabled. The overall network bandwidth and link-loss rates were not adjusted artificially, however, if intermediate buffers become full, packets may be dropped. Experiments were conducted over Gigabit Ethernet while using WebSockets for transmitting status and PER messages. Based on measurements from past studies, the client to server RTT was set to 120ms and 3s segmentation was selected [16, 22].

For performance comparison with prior studies, the following metrics were also used

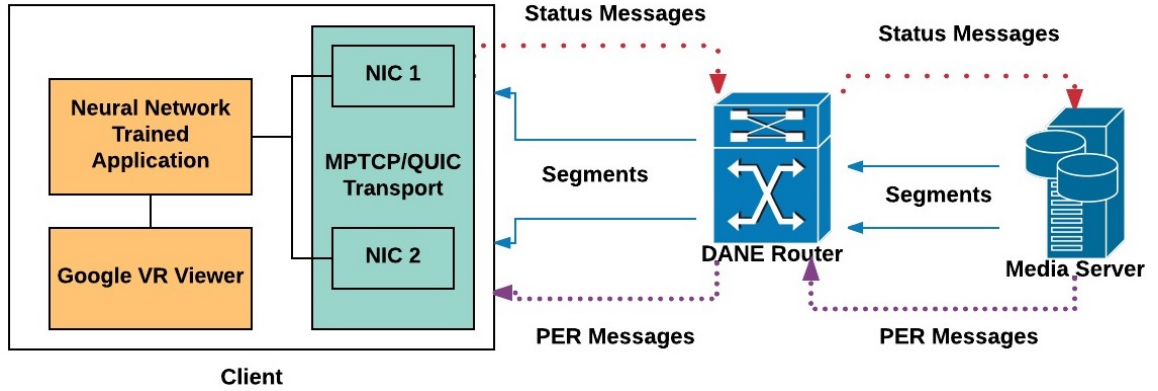


Figure 6.3: Implemented multipath SAND testbed

for this study:

1. *Differential buffering delay*; the difference in the delay between individual multipath links.
2. *Playback buffer level*; the number of seconds of video stored in the client buffer ready for playback.
3. *Average connection throughput*; the average throughput over the last second of the connection.
4. *Number of quality switches*; the number of times during the connections lifetime that the quality of the video was 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 was used.

The algorithm may experience unfamiliar priorities or encounter networks it has never been exposed to. To test the algorithm performance in this situation, a second set of experiments were conducted using various traces from actual LTE, WiFi, and LAN traffic that the neural network was not trained on. Measuring the bitrate utility, playback smoothness,



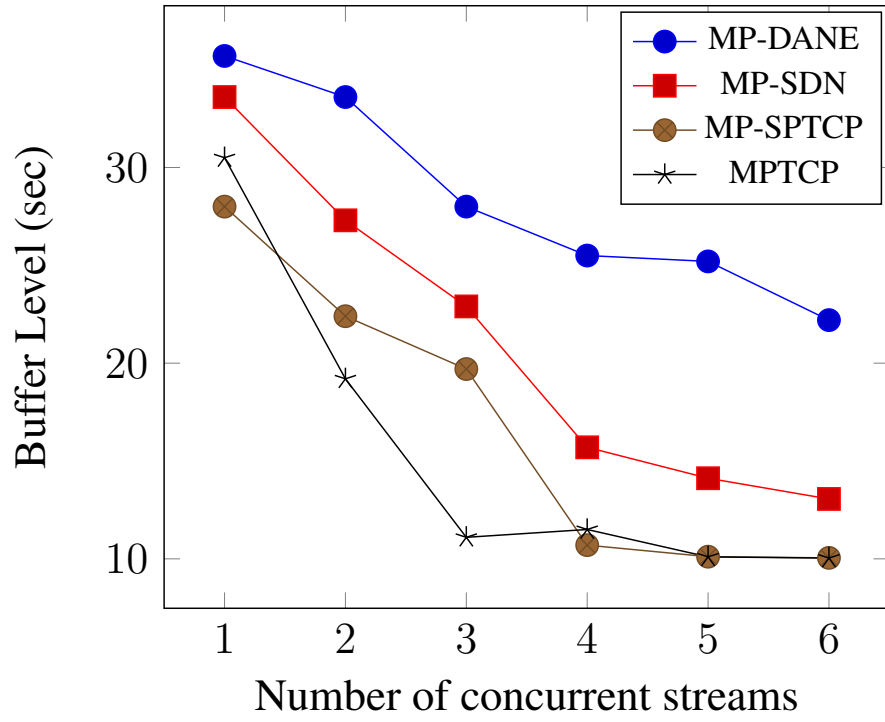
and rebuffering penalties during an aggregate average of streaming sessions, these metrics were used as a crude QoE test given essential QoE metrics remain unknown.

The neural network trained application relies on SAND PER messages from the DANE network to anticipate network fluctuations. In parallel, a DANE router prefetches chunks based on status messages from the client to a DANE network. The testbed layout, as shown in Fig. 6.3, supports SAND, QUIC, MPTCP, MP-SDN, and MP-SPTCP.

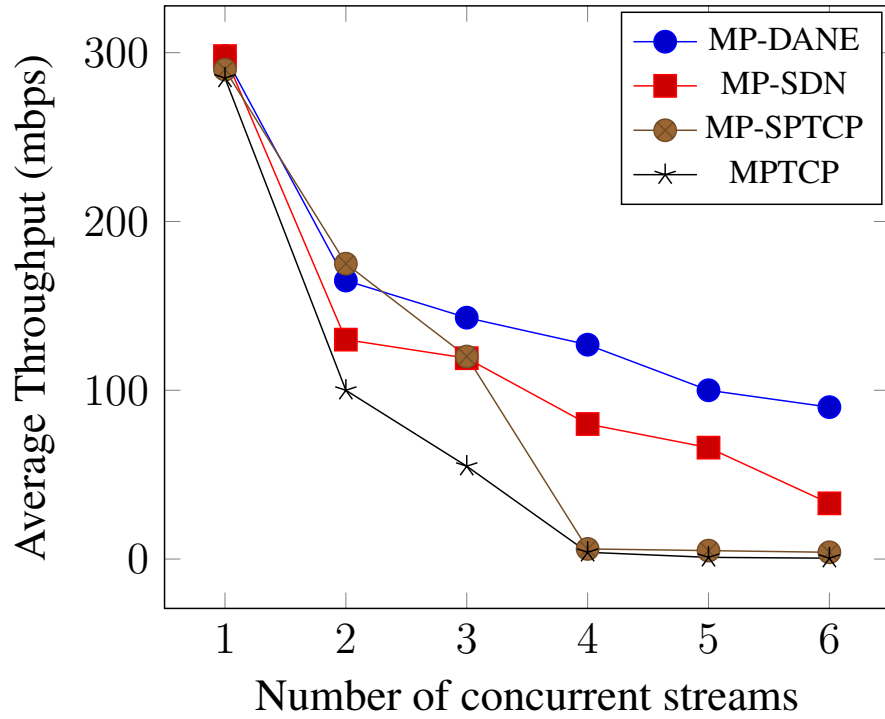
### 6.3 Evaluation

The figures below highlight experimentation outcomes. Fig. 6.4b - Fig. 6.6b display an aggregate of averages across concurrent streams for each algorithm in the testbed LAN network. Fig. 6.7a - Fig. 6.8a reflect the average reward and penalty values accumulated when testing these algorithms in WiFi, LTE, and LAN networks. MP-DANE/DANE refers to this studies' solution presented in the figures. MP-SDN refers to an implementation, which control traffic flows in multipath networks with the aid of SDN [62]. MP-SPTCP refers to an implementation of a technique that relies on active queue management and use of bandwidth thresholds instead of RTT and retransmission thresholds [31]. Lastly, MPTCP refers to a standard implementation of the protocol without modification.

Fig. 6.4b compares the average number of seconds of video in the buffer versus a number of competing video streams on the network. Acknowledging this figure alone does not reflect the bitrate of the video stored in the buffer, therefore Fig. 6.4b can be referred to for context. With a single 360 VR video stream, each algorithm successfully maximizes the network's capacity. As the number of concurrent streams begin to increase, MP-DANE outperforms with higher quality buffered video. The neural network has learned to fill its buffer quickly; primarily in unreliable environments leveraging the use of symmetric flows from SAND messaging. In this testbed, when the number of concurrent streams surpassed three, only MP-SDN and MP-DANE (see Fig. 6.4b and Fig. 6.5a) performed well illustrating that standard MPTCP and MP-SPTCP were unable to maintain usable

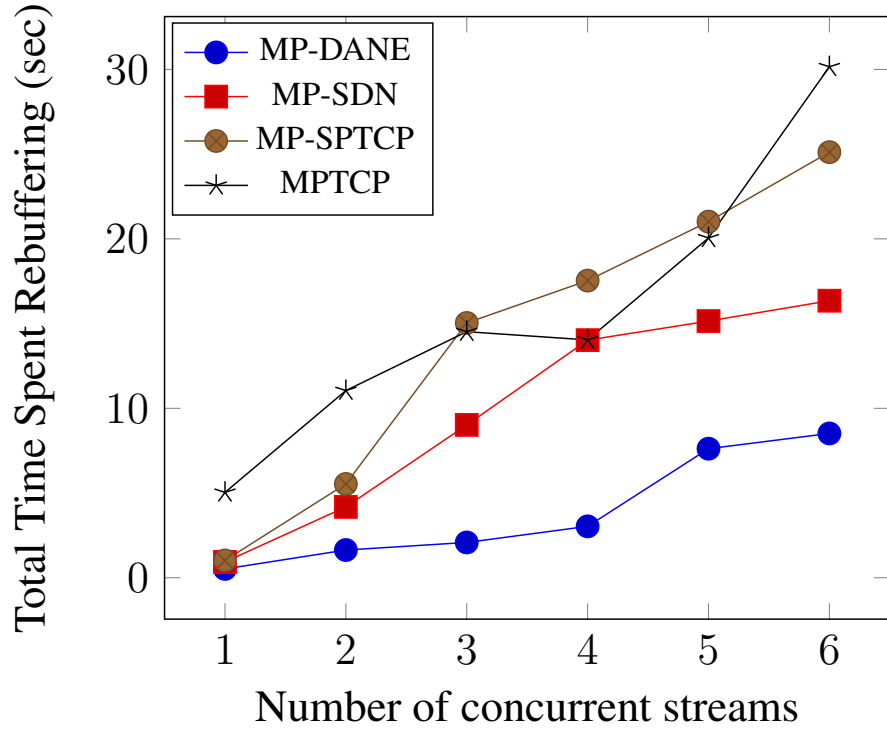


(a) Average buffer level

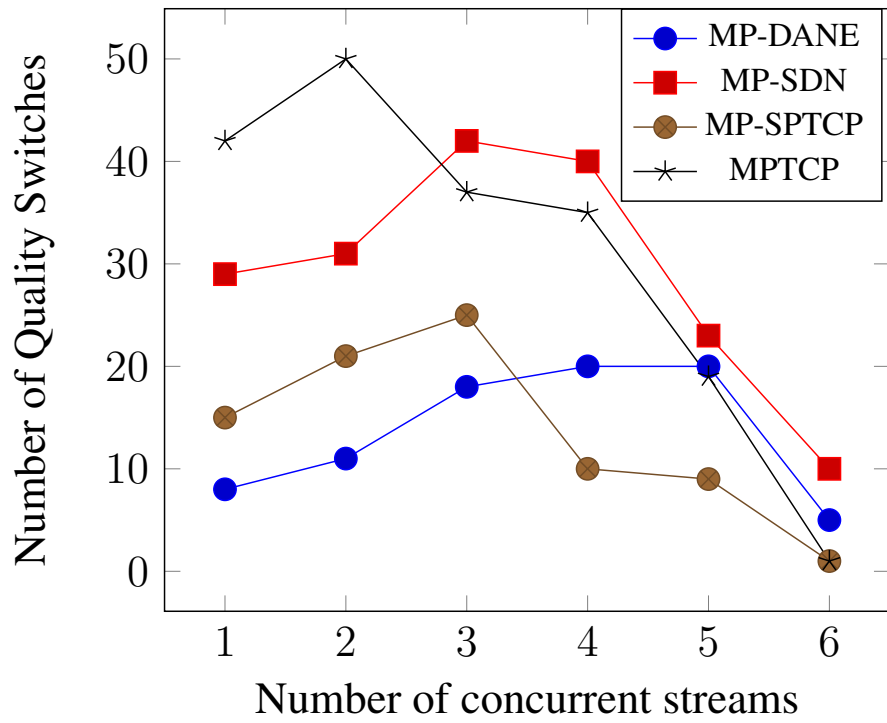


(b) Average user throughput

Figure 6.4: Various performance metrics for a given number of users.

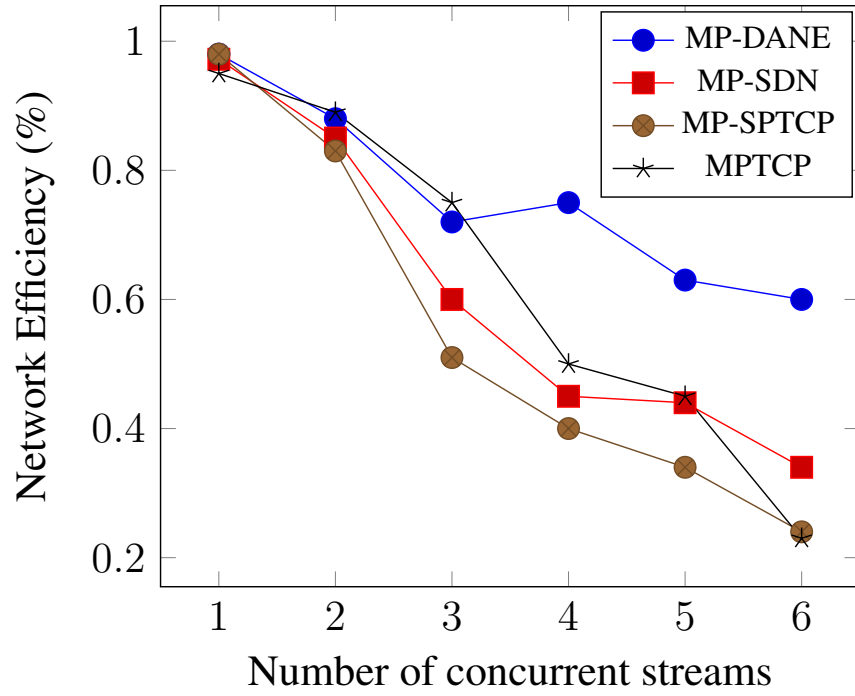


(a) Average time video playback stalled

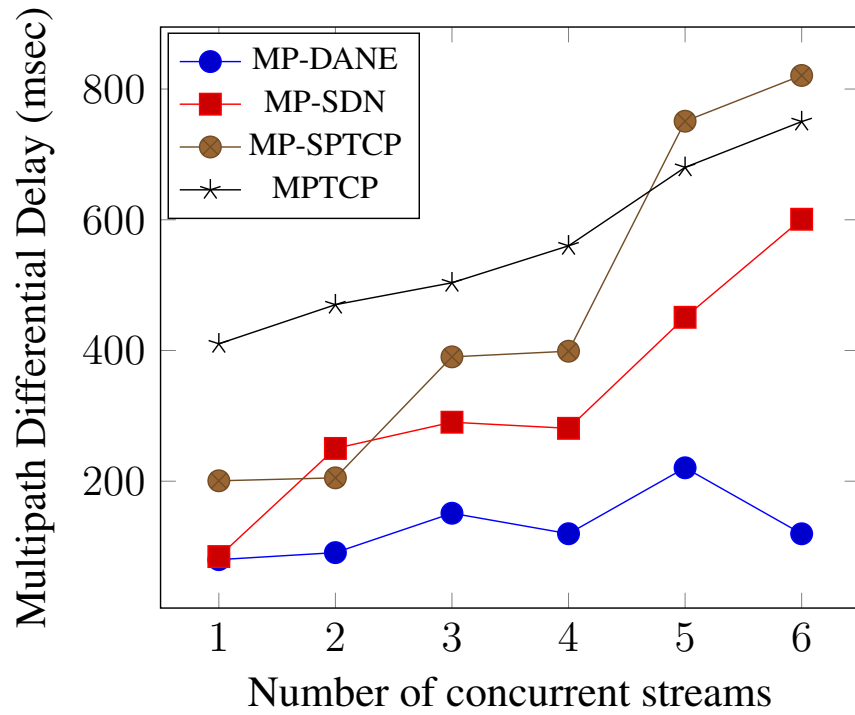


(b) Average number of quality switches

Figure 6.5: Various performance metrics for a given number of users.



(a) Network usage efficiency



(b) Average multipath differential delay

Figure 6.6: Various performance metrics for a given number of users.

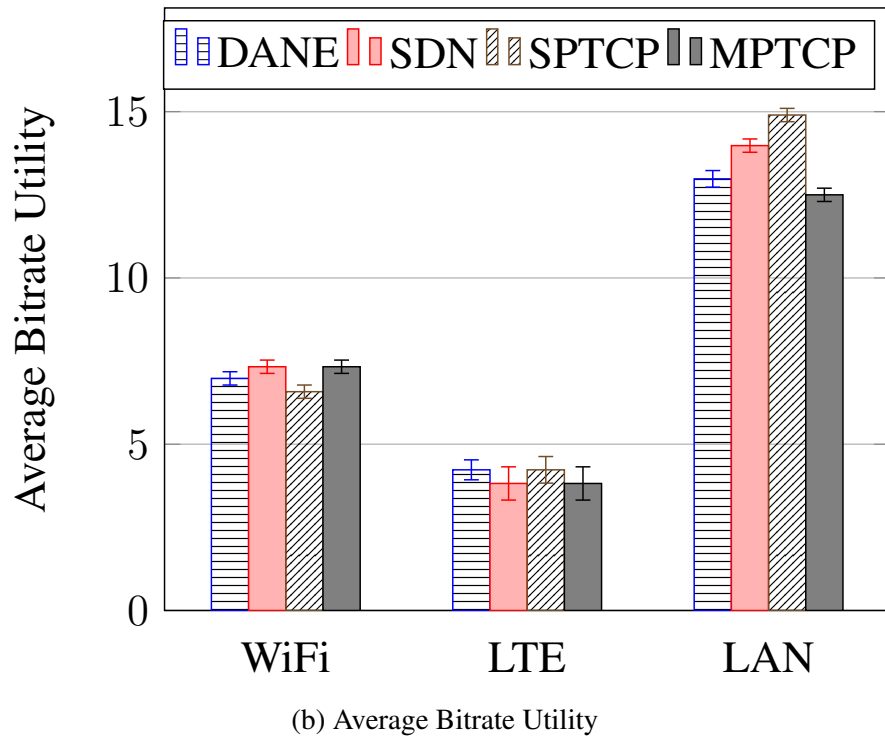
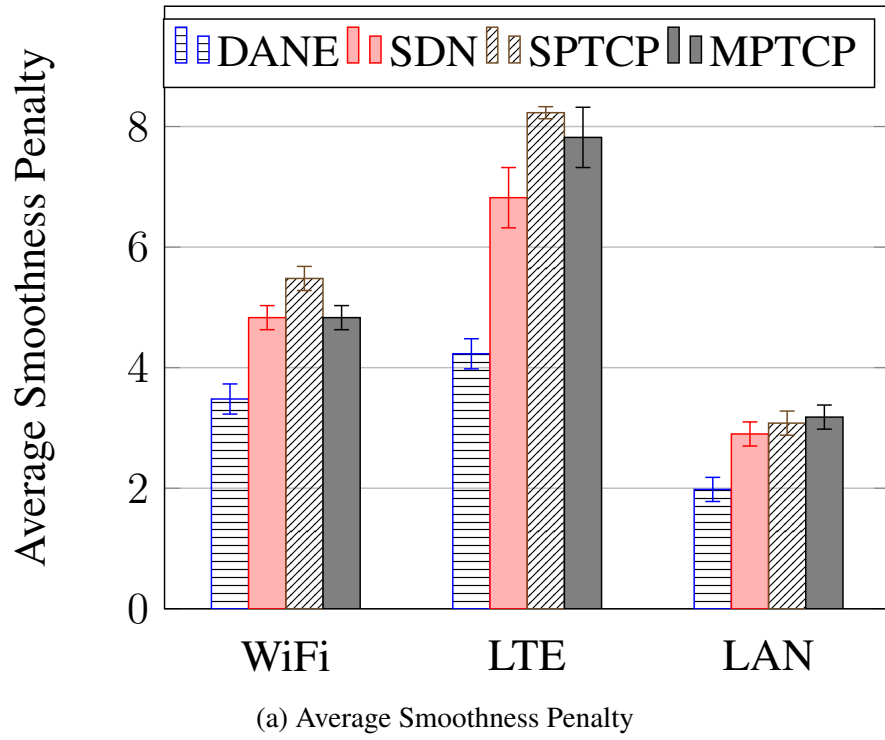


Figure 6.7: Various performance metrics for a given number of users.

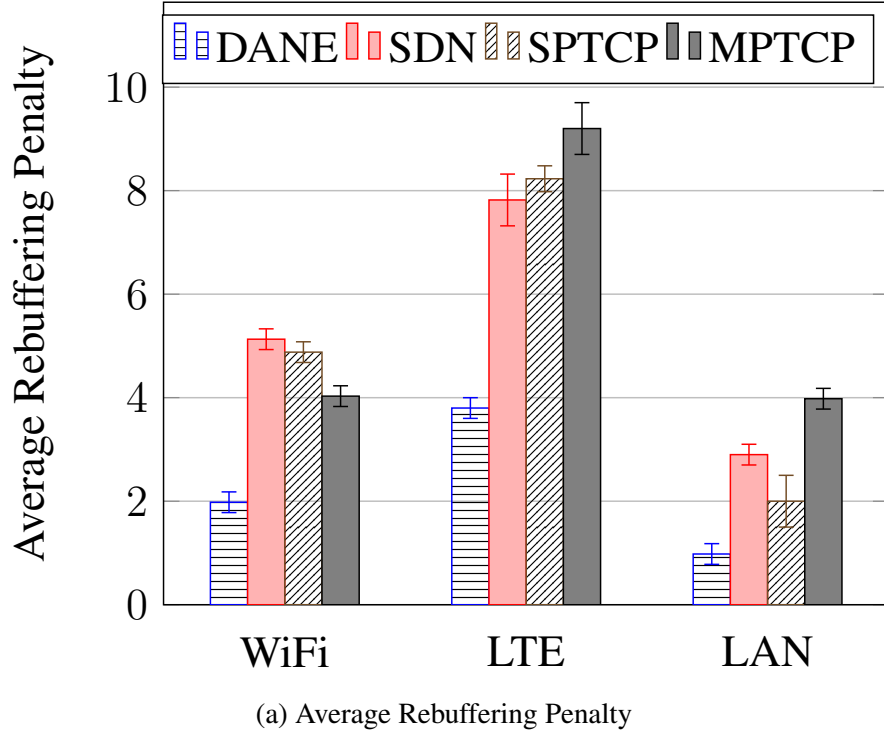


Figure 6.8: Various performance metrics for a given number of users.

bandwidth over the lifetime of the connection.

Fig. 6.5b highlights a pattern in which each algorithm begins with few quality switches, reaches a peak, then declines. There is little bitrate variance in non-congested or highly congested networks as the network is likely requesting the highest or lowest bitrate respectively. However, in-between those limits, the number of quality switches increases as it approaches saturation. Based on experimentation, this point is dictated by the physical layout of the network.

Fig. 6.6a illustrates that MP-DANE utilizes more of the available bandwidth than the other configurations in highly congested networks. However, MPTCP occasionally outperforms MP-DANE's efficiency when network paths are closely balanced. As shown in Fig. 6.6b, MP-DANE has consistently better multipath differential delay across all experiments, whereas MP-SDN diverges as congestion increases. Concluding from analyses, this was a result of the overhead of MPTCP, MP-SDN, and MP-SPTCP when building connections without considering the impacts of differential delay.

Important observations to note from Fig. 6.7b is that MP-DANE performs comparably, but does not generate the highest quality bitrate utility across all networks. MP-DANE does however outperform all other techniques when considering rebuffering and smooth playback (see Fig. 6.7a and Fig. 6.8a). As explained in Section 6.1, the reward function weighs playback smoothness and rebuffering penalties more than the bitrate utility to avoid stalls. However, when training the neural network, the QoE reward weights can be changed to match the desired balance.

## 6.4 Discussion

This paper proposes an adaptive 360 VR video testbed to develop and evaluate neural-based ABR algorithms to control unfair delivery using SAND in multipath networks leveraging MPTCP and QUIC. The development of multipath aware and teachable ABR algorithms begin to address limitations commonly seen in modern ABR algorithms including removing reliances on inaccurate throughput models and rigid control heuristics that allow the formation of optimized algorithms. Extending SAND for use in multipath networks allowed for algorithm development based on IETF standards to not only actively switch between MPTCP networks and QUIC, but to modify MPTCP paths in real time. As a result, the average differential delay decreased by 30% or more when using MP-DANE as compared to MP-SDN, MPTCP or MP-SPTCP. Playback stall time also decreased by more than 25%. Furthermore, MP-DANE utilized the network 30% more efficiently in networks with competing traffic. These results demonstrate that this algorithm outperforms MP-SPTCP, MP-SDN, and MPTCP in many key streaming metrics. However, one limitation of this work is the offline training period. Further research expanding upon the investigation of solutions to decrease offline training time of this algorithm was explored in the following chapters.

## **CHAPTER 7**

### **REINFORCEMENT LEARNING ADAPTIVE CONTENT DELIVERY NETWORKS (CDN)**

The preceding chapter illustrated a new branch of research in the field of dynamic transport selection in multimedia streaming. The incorporation of SAND and RL into the ABR streaming decision process has resulted in a renewed research interest into ABR streaming algorithms. This chapter continues that research from a different approach. Experiments described in Chapter 6 incorporated RL into ABR algorithms that allowed clients to make data driven bitrate decisions. Removing the limitation of a SDN network, SAND was used to replace it and provide many of the same SDN features to clients without a managed network. Combining massive scale simulation environments with a trained neural network was useful for the general video streaming experience use case. Despite this, clients increasingly demand streaming personalization and are moving towards 4K, and 360 Virtual Reality (VR) along with other bandwidth prohibitive streaming technologies. Delivering a personalized experience without increases in network and data centers costs has and will continue to become increasingly challenging.

This chapter presents a unique approach to address these specific personalization requirements. First, through the implementation of a novel CDN that distributes a lightweight video specific neural network along with video stream manifests during initialization. Secondly, through optimization of ABR algorithms using reinforcement learning at the CDN, and finally, by active management of multipath transport networks. This system continues to support dynamic transport selection when streaming video content at scale, but focuses on delivering more customizable experiences that would scale efficiently starting with the design that was generated in Chapter 6. As mentioned in the previous chapter, when scaling video delivery there is a tendency to have an over-reliance on centrally managed network-



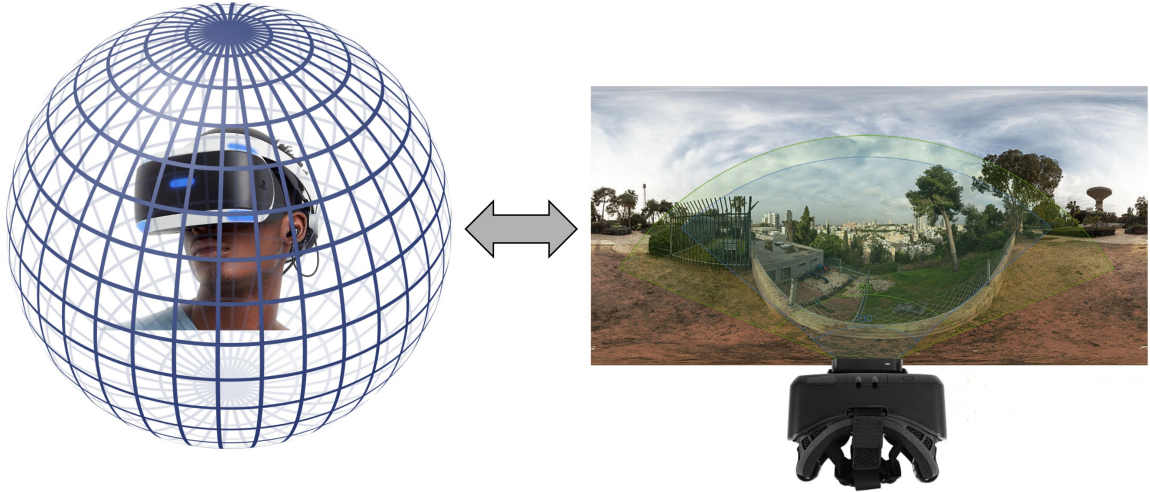


Figure 7.1: Example 360 VR video field of view mapping

ing. This chapter prioritizes tailoring unique video streaming experiences at the client using CDNs that do not rely on centrally managed networks (i.e., SDN), while leveraging RL, MPTCP, and QUIC.

The last mile of modern video streaming between a CDN and end-users take advantage of ABR. As discussed in preceding chapters, traditional ABR techniques lack personalization, rely on imprecise network models, and use fixed adaption rules leading to poor streaming experiences. These drawbacks stem from their lack of network-wide knowledge and control. Additionally, there is no consensus with media players about how they make bitrate control selections. Oftentimes those decisions are in conflict with one another causing network inefficiencies. This work looks to fill that gap by focusing on customizing 360 VR media delivery in multipath networks (see Figure 7.1). This chapter demonstrates an exclusive method modifying CDNs to train and distribute at scale, lightweight neural networks along with video manifests. The goal of this system is to remove limitations of client-side ABR in multipath networks by designing learning algorithms. Through a simulation of multihomed devices leveraging a CDN network, MPTCP and QUIC, extensive testing illustrates the ability of learning algorithms to more efficiently support 360 VR media streams than previously realized.

The application of RL in ABR streaming is a recent development, however the effects

of RL in multipath networks have yet to be taken into consideration. This limits the flexibility of many designs. RL has been used to train ABR algorithms on a large scale, but unfortunately different video types including VR requirements are commonly overlooked [49, 52, 53]. By taking these limitations into consideration in addition to using realistic parameters that are not reliant on managed networks is a primary focus of this chapter.

Detailed in this chapter is the development of a framework that delivers a unique approach modifying CDNs to train and distribute at scale, lightweight neural networks along with video manifests while actively managing transport selection of MPTCP or QUIC for streaming 360 VR video in varying network conditions. Previous research has not considered or explored leveraging neural network based CDNs with MPTCP and QUIC for the delivery of 360 VR content in multipath networks.

The chapter is organized as follows: Section 7.1 discusses the proposed framework strategy in detail. Section 7.2 describes the simulation setup and experimental parameters. Section 7.3 presents this chapter’s figures and results. Concluding, section 7.4 details strengths, limitations, and proposed future work.

## **7.1 Architecture Design**

This section presents the methodology reinforcing an architectural design which support evolving CDN-based neural networks for 360 VR media delivery in multipath networks. This begins with an intuition discussion followed by high-level design characteristics. Next, an in-depth CDN-to-client interface analysis to support, train and distribute lightweight neural networks along with video manifests is described. Finally, considerations made for multipath users are detailed.

### 7.1.1 System Design

In traditional networks, CDNs distribute manifest files describing the video formats supported and clients download the corresponding videos in *chunks*. Having the CDN act as a

cache media server is a critical juncture in the network and serves many different videos to various device types. The CDN consequently is able to keep track of useful performance metrics (i.e. device type, bandwidth performance, video selection, latency, etc.). Under typical circumstances, clients make bitrate decisions locally in isolation. The CDN is in a unique position to “advise” clients on the best course of action based on thousands of streaming hours from similar devices on comparable networks. Using this insight, a CDN was developed that trains a lightweight neural network in the background specific to the video being downloaded. When a new client begins a stream, a neural network model is incorporated into the metadata allowing CDNs to take an instructive role for clients streaming a particular content. This design encourages improved QoE with a customized adaption experience that scales. The client can then leverage this light network to make bitrate decisions.

The high-level, CDN-based neural network architecture displayed in Fig. 7.2 consists of three main modules. The first, a video identifier module, classifies content based on playback requirements. The second module, a neural network training block, collects statistics from the CDN’s video identifier and is fed into a network training session. Individual neural networks are created for each new content type. Lastly, the stored model module caches completed representations to be disseminated in the metadata associated with manifest files when a client begins playback using existing content models.

Once a video is stored in the CDN, it is tagged and identified based on the type of content and the playback requirements (i.e. latency, RTT, etc.). The following training step required the development of an evolving generative training model using RL. To accurately design such a model in an ideal setting, the learning agent would need access to thousands of actual streaming sessions to capture every subtlety of the video streaming experience. The limitations of this training technique are numerous, requiring many days of training time and access to thousands of users. Because of this, themes proposed in previous RL studies are used by exploiting a simulator that models the streaming experience. A simula-

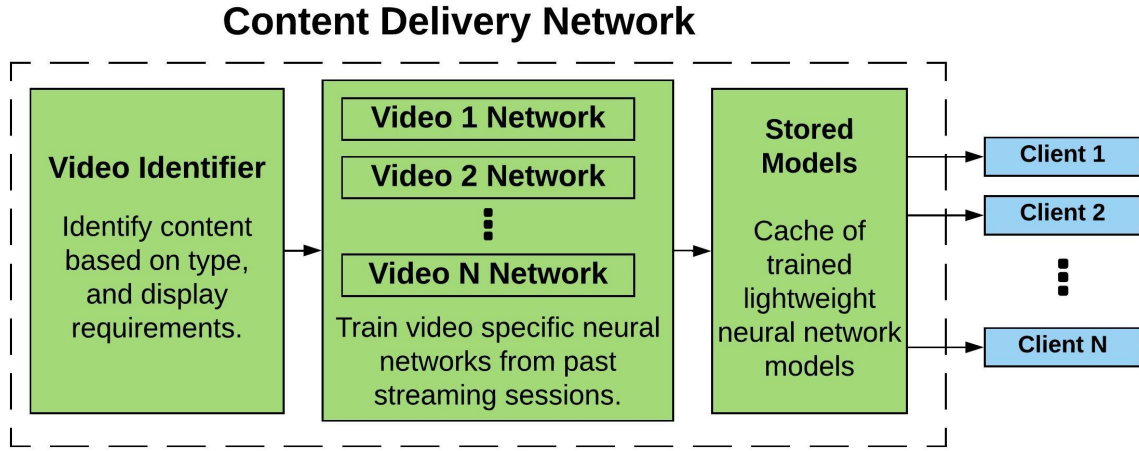


Figure 7.2: Flow diagram for neural network-based CDN architecture

tor was used that models network effects while being paired with actual streaming clients. In this study, the traditional client-side adaption algorithms were removed and the neural network model was substituted. This simulator allows for shortened training durations by allowing these small models to learn from multiple real-world network traces. These traces represent different instances of the exact same content but streamed from various devices in varying network conditions. The simulator models playback buffer characteristics, RTT, rebuffering events, and download times.

As mentioned in Chapter 6, the RL process starts with a learning agent at each decision point observing the state of the model to infer how its decisions have changed the system environment. After observing the environment, the agent provides some external input into the system to receive and receives a resulting reward. The agents intention ultimately is to drive the cumulative discounted reward towards an optimal policy set forth prior to training. The reward function consists of three metrics commonly used for this type of research: bi-rate utility, smoothness playback penalty, and rebuffering penalty. For the offline training phase, the training algorithm parameters are as follows: previous chunk bitrate, previous chunk download duration, previous chunk throughput, the number of chunks left for download, encoded chunk size, and buffer status.

Discrete-event network simulators for Internet-based systems are routinely used to ab-

stract away nonessential lower-layer physical elements. The rational for this technique is that often many of these low level features are non-repeatable for testing purposes [65]. Simulation allows for the rapid deployment and testing of critical protocol aspects run in the user space of operating systems. Having concepts that are simplified but provide visibility into actual interfaces between stack layers are critical. Emulation provides a bridge between simulation and actual hardware allowing protocols to be run on real networks while still removing real-world problems such as unexpected link failures. To repeatedly test protocols, there needs to be a reliance on fast and simple solutions. As a result of these implications, emulation was used for experimentation to produce meaningful results that are not only theory based. Relying on complete networking models would be ideal, however as shown above it would significantly increase training complexity. In production, every simulator and networking model makes some set of assumptions, however this technique has been used by numerous researchers producing accurate results.

The formalized reward function in Eq. 7.1.  $R$  at step  $i$  was developed in a prior work [53], but was modified using weights  $x, y, z$  that were based on content requirements. *Smoothness Penalty* ( $S$ ) is the magnitude of the bitrate change from one chunk to the previous chunk. *Rebuffering Penalty* ( $Re$ ) is the stall duration. *Bitrate Utility* ( $B$ ) reflects a higher utility with a higher bitrate.

$$R_i = x * B_i - y * S_i - z * Re_i \quad (7.1)$$

In addition to developing novel evolving generative personalized neural networks for the CDN, this algorithm was designed to be applicable not only in single path networks, but in networks with access to multiple paths. Therefore included are transport protocol ideas verified in the literature [27]; particularly the use of balanced RTT network paths as balanced RTT networks have been found to improve QoE. When there are multiple networks available, the transport decision engine does a trade-off analysis to balance streaming performance priorities deciding which path to use. This encourages path diversification

dealing with network fluctuations.

## 7.2 Experiment Setup

The TensorFlow [64] framework and software library provides a rich set of RL tools and was used for the development and testing of many prior works and as such, it was utilized in this chapter as well. The training process of a RL algorithm is fairly straight forward. The RL algorithms under test are given a one-dimensional convolution layer with 128 filters each with a size of two and with a stride of one. Training each neural network took approximately two hours for the number of clients used in this chapter. As the number of clients increases, one would expect the offline training period to increase as it usually takes longer to converge on a solution when the data set is large.

Experiments were conducted in an example simulated network layout shown in Fig. 7.3. The encoded adaptive 360 VR videos were ten minutes in length. The content was encoded with x265 at three different well-spaced bitrates between 5 Mbps and 10 Mbps.

The performance of MP-DANE, MP-SPTCP, and MPTCP was compared with a MP-CDN implementation. Details of the comparison algorithms are discussed in Section 7.3. Experiments were performed while varying the number of concurrent streams from one to six. All network nodes were MPTCP and QUIC enabled. The overall network bandwidth and link-loss rates were not adjusted artificially, however, if intermediate buffers became full, packets were potentially dropped. Based on measurements from comparable studies, the client to server RTT was set to 120ms and 3s segmentation was chosen.

For comparison, the following metrics were studied:

1. *Estimated mean opinion score (eMOS)*; rating system developed by Claeys et. al [47].
2. *Playback buffer level*; the number of seconds of video stored in the client buffer ready for playback.

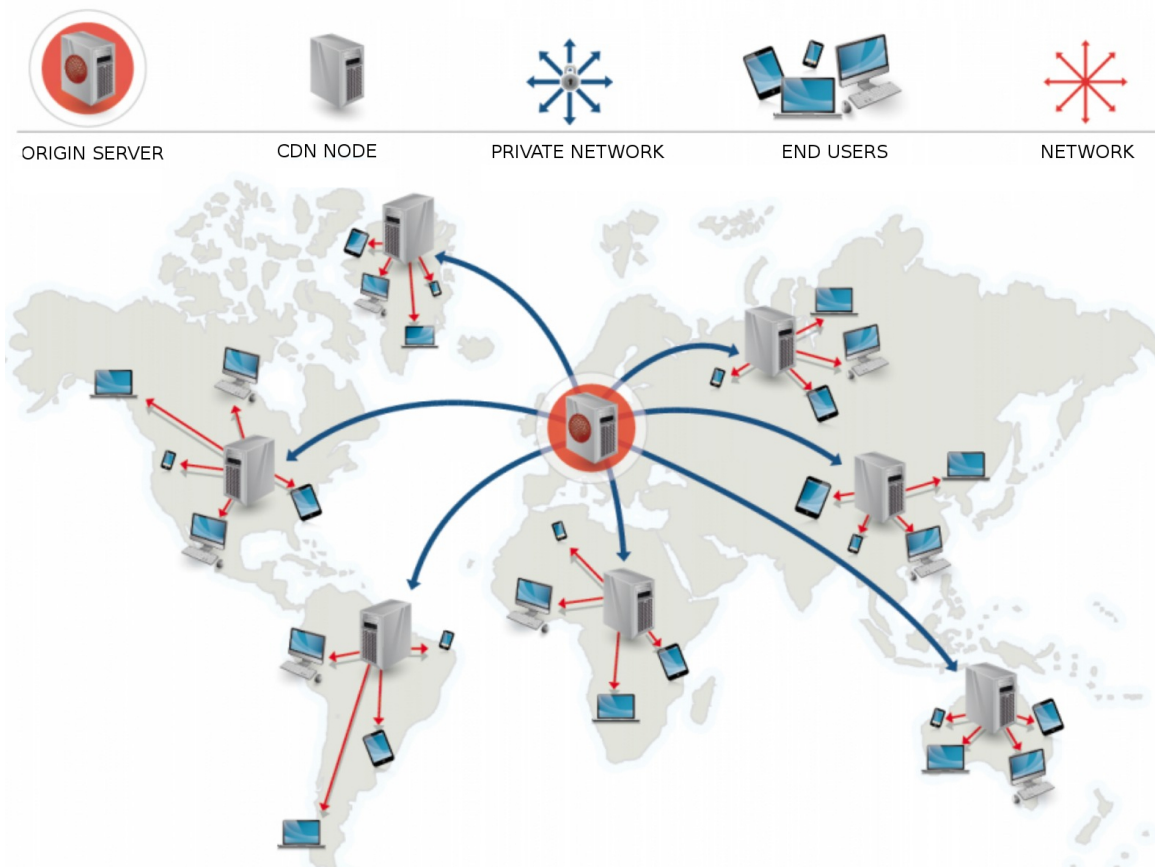


Figure 7.3: High-level simulated network topology

3. *Average connection throughput*; the average throughput over the last second of the connection.
4. *Number of quality switches*; the number of times during the connections lifetime that the quality of the video was 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 was used.

The eMOS rating system was used to compare the performance of this work's algorithm to others on a level playing field using the same evaluation system. There is a trade-off being made between the average quality of selected representations, the quantity and

magnitude of switches among various representations, and the frequency. The duration of freezes are also considered, as this has a significant impact on QoE. If the eMOS score is close to zero, then its only goal at that point is to fill its buffer with content; the system is not concerned with the quality level as there is a high chance of video stall. If the eMOS score is close to five, the QoE is the main priority and only the highest quality video is requested. Being able to fill the buffer is not a concern as the displayed content is at the highest quality.

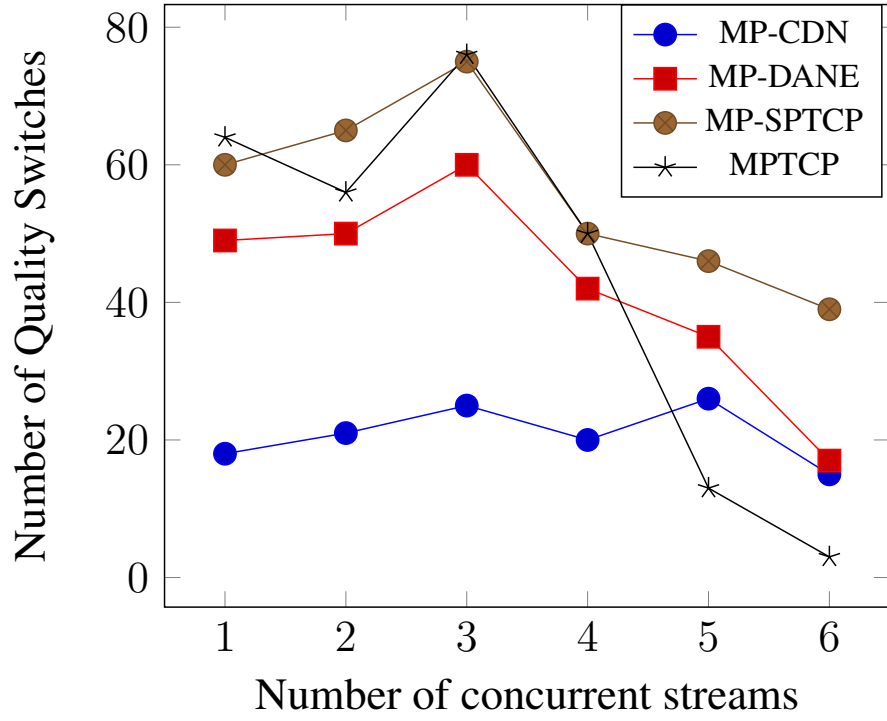
Lastly, when considering real deployments in the field, this algorithm may encounter various types of content it has never been exposed to in training. Evaluating for this scenario, a second set of experiments were conducted studying the peak signal-to-noise ratio (PSNR) of different content types as the resolution changes. PSNR metrics were used as a crude QoE test given exact QoE metrics are unknown for new content.

### 7.3 Evaluation

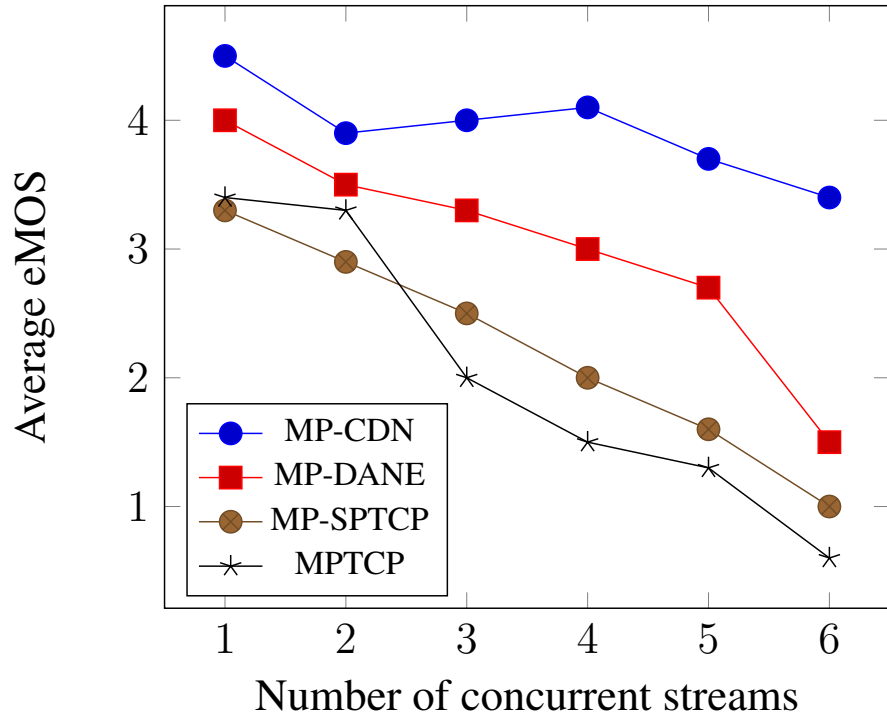
The plots below outline the findings from experimentation and are organized as follows, Fig. 7.4a - Fig. 7.6b display concurrent stream aggregates for each algorithm in the testbed network. In Fig. 7.7a - Fig. 7.8a displayed is the PSNR values at a particular resolution when testing algorithms streaming various types of content. MP-CDN/CDN refers to the solution presented in the graph figures for this chapter. MP-DANE/DANE refers to an implementation of prior work which used learning algorithms in multipath networks with the aid of an SDN [53]. MP-SPTCP refers to the implementation of a modern technique which uses active queue management and bandwidth thresholds [31]. MPTCP in the plots refer to an unmodified version of the standard plugged directly into the system.

In Fig. 7.4a, there is a pattern seen with most algorithms in which it began with few quality switches, reached a high point, then declined. MP-CDN however, differs in that it stays fairly steady. This is because the CDN has learned from previous streaming sessions how to balance multiple concurrent flows. For the other implementations, there is minor



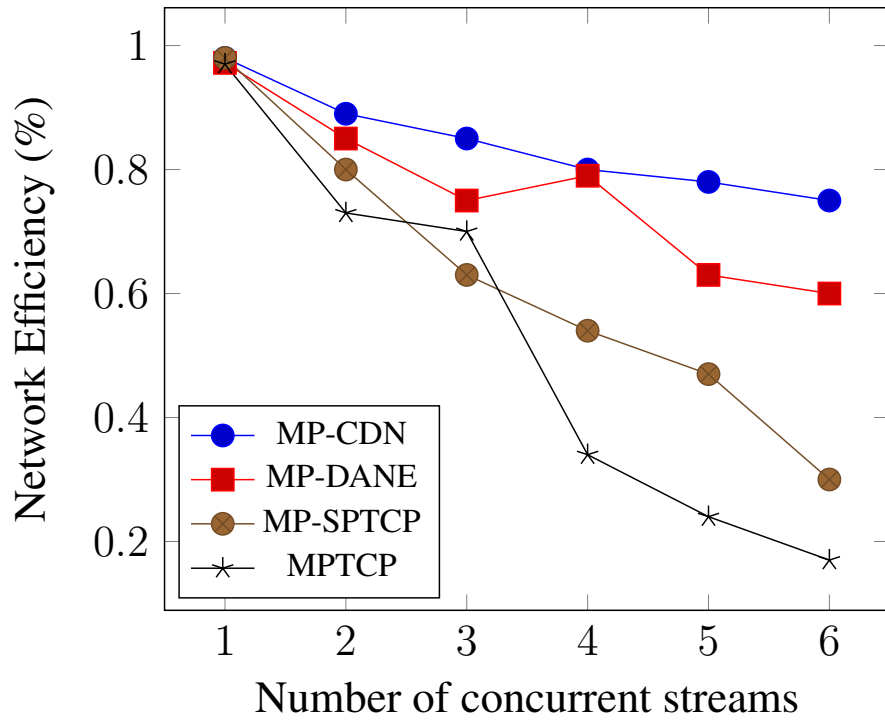


(a) Average number of quality switches

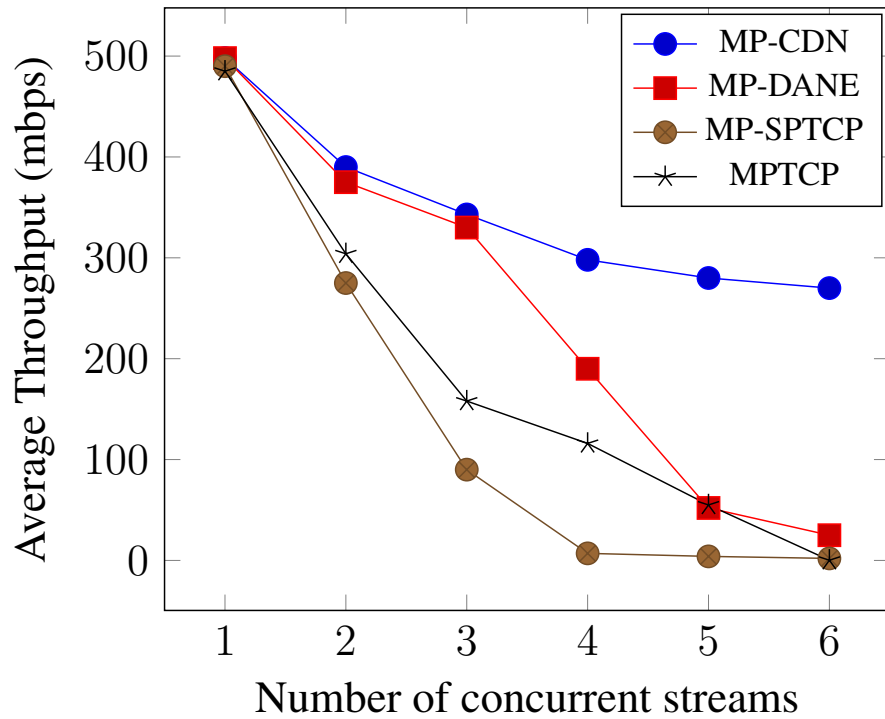


(b) Average estimated mean opinion score

Figure 7.4: Various performance metrics for a given number of streams.

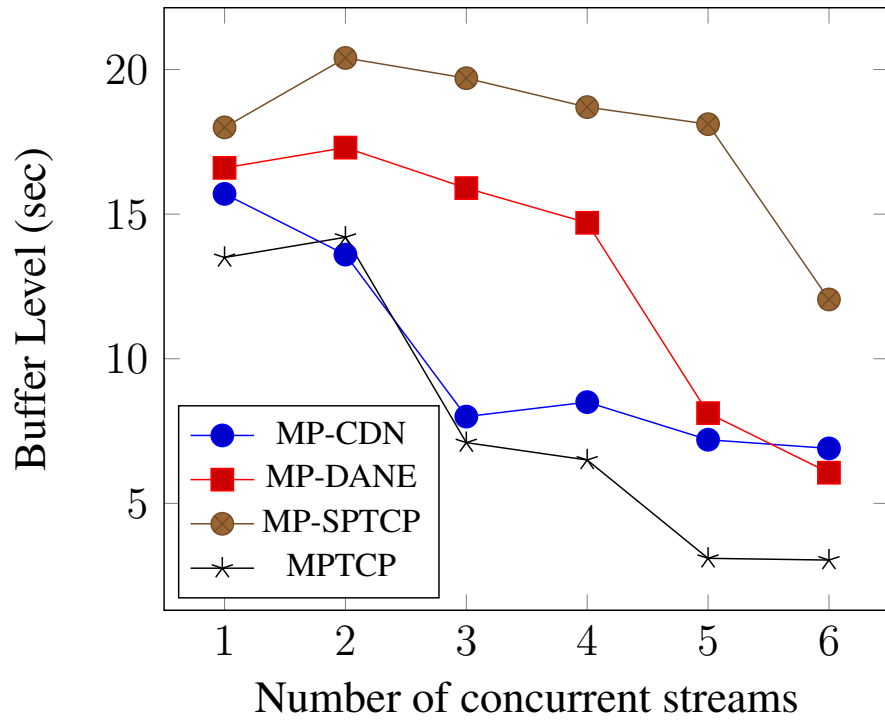


(a) Network usage efficiency

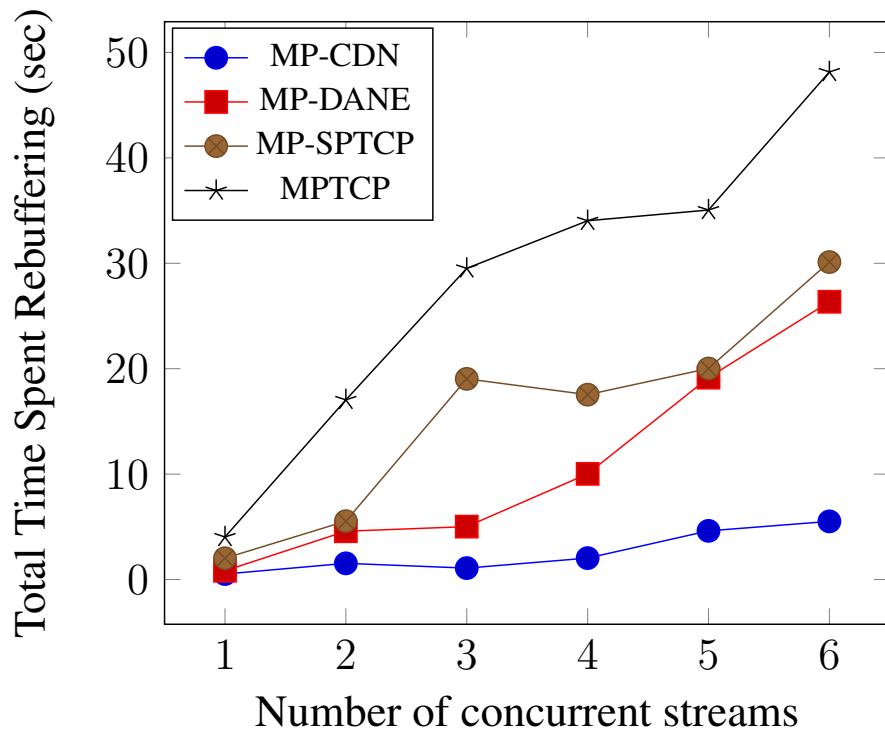


(b) Average user throughput

Figure 7.5: Various performance metrics for a given number of streams.



(a) Average buffer level



(b) Average time video playback stalled

Figure 7.6: Various performance metrics for a given number of streams.

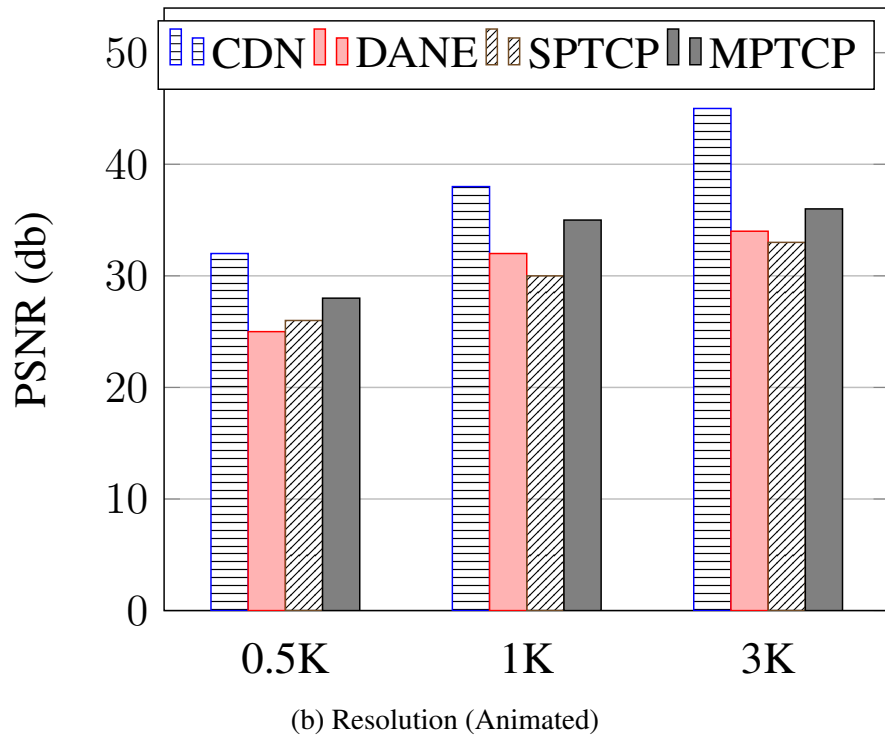
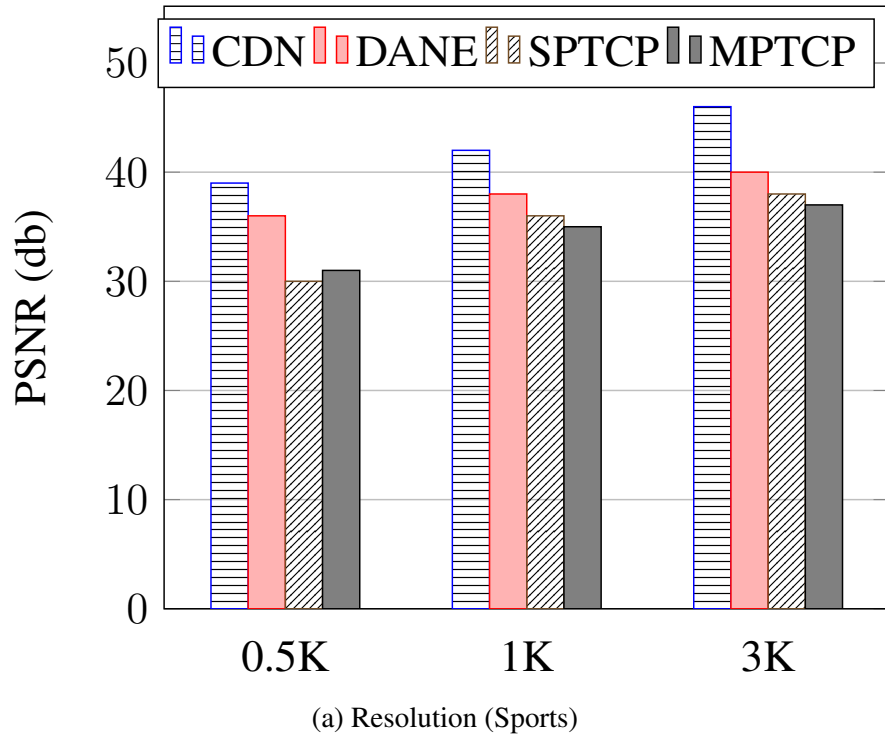


Figure 7.7: Various performance metrics for a given number of streams.

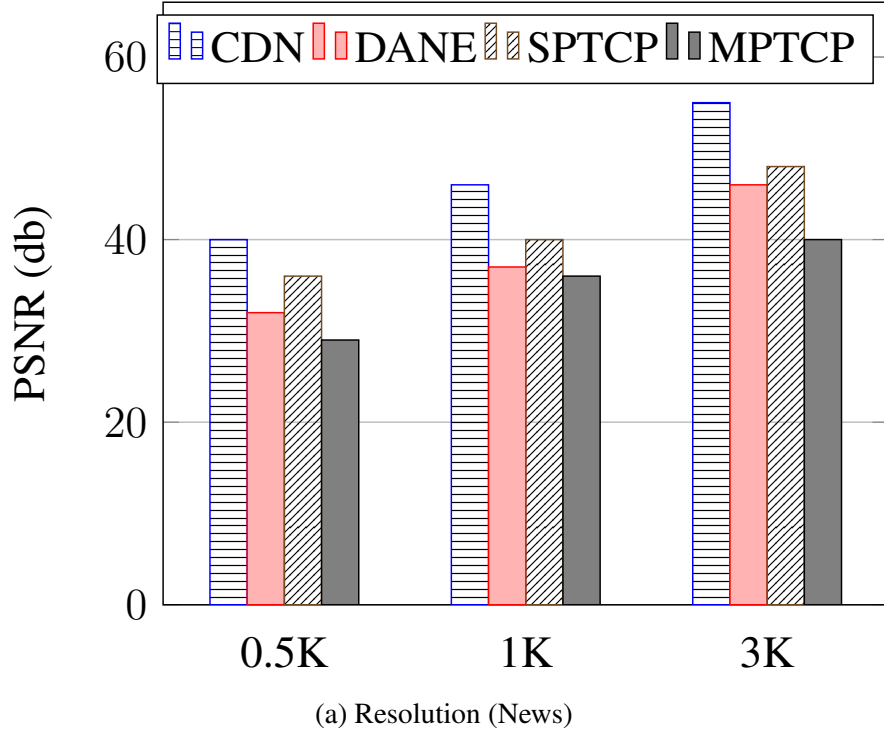


Figure 7.8: Various performance metrics for a given number of streams.

variance in non-congested or highly congested networks as the network is likely requesting the highest or lowest bitrate respectively. However, in-between those limits, the number of quality switches increased as it approached saturation. Fig. 7.4b uses the estimated mean opinion score (eMOS) algorithm [47] in which it creates a reward function that combines multiple streaming metrics for a single score for rating streaming performances. MP-CDN has shown to perform better across content types when using eMOS, whereas the rest of the techniques performance dips as the number of users increase. Concluding from analyses, this was a result of the overhead of MPTCP, MP-DANE, and MP-SPTCP when building connections without being able to rely on trained neural networks from a CDN. Fig. 7.5a illustrates that MP-CDN and MP-DANE utilize more of the available bandwidth than the other configurations in challenging network conditions. MPTCP's performance is lackluster with this network configuration.

Fig. 7.6a displays the number of seconds of video in the buffer versus a number of competing video streams on the network. Given the fact that this figure alone does not

reflect the bitrate or eMOS reflected to the users, Fig. 7.5b can be referred to as well. With a single 360 VR video stream, each algorithm successfully maximizes the network's capacity. MP-CDN outperforms the other implementations as the number of concurrent streams increases. For experimentation, when the number of concurrent streams surpassed three, only MP-CDN (see Fig. 7.5b and Fig. 7.6b) performed well illustrating that even an SDN network was unable to maintain better bandwidth over the lifetime of the connection. Observations made in Fig. 7.8a implies that MP-CDN/CDN generates a high PSNR value for News type content where most of the background is static. Improving upon the MP-DANE algorithm, MP-CDN outperforms other techniques when considering PSNR across multiple content categories (see Fig. 7.7a and Fig. 7.7b). As explained in Section 7.1, the lightweight content specific neural networks avoid pitfalls associated with having a limited view of the network. When training the neural networks, the CDN favors a stable performance strategy for optimal performance.

## 7.4 Discussion

This chapter proposes a unique multipath aware approach to designing CDNs which train and distribute at scale, lightweight neural networks in metadata along with video manifests. This development begins to address limitations commonly seen in modern ABR algorithms. Removing the reliance on throughput models and prebuilt heuristics allowed the formation of optimized algorithms.

In addition to extending recent research which use neural networks client-side with the aid of an SDN network, the SDN limitation was removed which led to improved performance. Results illustrate that the estimated mean opinion score was at least 20% higher when using MP-CDN as compared to MP-DANE, MPTCP or MP-SPTCP. Playback stall time also decreased by more than 15%. Furthermore, MP-CDN utilized the network 15% more efficiently in networks with competing video streams. These results demonstrate that this algorithm outperforms MP-DANE, MP-SPTCP, and MPTCP in many fundamen-

tal streaming metrics. One limitation not addressed in this work however is managing and distributing neural networks across CDNs. Future work needs to be conducted to expand upon the ease of interoperability within existing CDNs.

## **CHAPTER 8**

### **CONCLUSION AND FUTURE WORK**

#### **8.1 Conclusion**

Research has shown that changing user consumption habits along with improved network performance has led to an increase in multimedia streaming services (i.e., Netflix, Youtube, and Hulu) [66, 67, 68]. This increase is the result of improvements in mobile device technology as well as improved Internet connectivity. Content providers recognizing this trend are investigating techniques to improve the user experience, one of which includes personalization of content. As content providers further invest in delivering 4K and 360 VR content and other future streaming technologies, delivering these personalized experiences without infrastructure investments or increasing costs will be nearly impossible. The future of ABR streaming requires content developers and users to rethink the utilization of today's existing technologies. The introduction of ABR streaming has allowed content providers to hone in on specific parameters while concurrently optimizing its performance in real-time and on a large scale leading to the interoperable transmission standards of MPEG-DASH. Unfortunately, the capabilities of this legacy technology has lagged behind the performance requirements of up-and-coming media [5].

The objective of this research was to develop a unique approach to delivering high-quality multimedia data by leveraging concepts from various multipath networking models in conjunction with ABR protocols that augment the reliance on client-side network bandwidth estimation and/or buffer-state. Prior research into the development and evaluation of the performance of multipath-enabled multimedia streaming techniques have been presented. Machine learning based multipath-enabled techniques have been developed for selective transport protocols for streaming applications while validating their effectiveness.



In fulfilling these objectives, this work began with an investigation of techniques for improving ABR video performance using HTTP/2 with MPTCP. It focused on improving the overhead of the initialization period and the overall streaming experience by implementing three novel modular HTTP/2 push strategies over MPTCP. HTTP/2 allowed for some of the streaming complexity logic from the client to be reallocated by using server resources. The results demonstrated that MPTCP and HTTP/2 outperformed standard MPTCP and TCP in lossy networks by 5% or more in nearly every important streaming metric outlined. These results suggest that HTTP/2 can improve MPTCP's performance.

Examining this work led to the realization that TCP-based protocols have fundamental limitations as outlined throughout the literature. As a result, a solution was developed for improving ABR video performance using MPTCP/QUIC over an SDN architecture. The initial thought process was to use a UDP-based protocol to augment MPTCP in a managed network with a focus on improving the start-up experience and overall streaming. This was done by designing two strategies leveraging SDN. The use of SDN enabled the design of an algorithm to actively switch between MPTCP networks and QUIC. By using MPTCP/QUIC over SDN, the experiments were able to take advantage of path variety for increased performance in low-loss networks and perform optimally in high-loss networks.

Limitations that were acknowledged from the preceding work included the experimentation scale. Previous studies were conducted in a small local testbed however, investigating ways to use a large testbed such as Planet Lab was of interest. A plan was established and executed to examine other MPTCP path management algorithms in conjunction with this new testbed along with developing, evaluating, and scaling an ABR video delivery using MPTCP and QUIC over an SDN architecture. Performing differential delay minimization for multipath capable devices was a primary focus for this study at scale. The use of SDN enabled the design of an algorithm that not only actively switched between MPTCP networks and QUIC, but modified MPTCP's paths in real time using a flow allocation scheme. Playback stall time decreased by more than 50% in networks larger than 300 users. Further-

more, MP-SDN utilized the network 30% more efficiently in networks with 800 or more users. One primary limitation to this work includes the use of a global managed network. Efforts were made in expanding this work using an adaptive video testbed aimed to develop and evaluate neural-based ABR algorithms to control for unfair delivery using SAND in multipath networks leveraging MPTCP and QUIC. The development of multipath-aware and teachable ABR algorithms began to address limitations commonly seen in modern ABR algorithms. Eliminating reliances on unsustainable throughput modeling and simple control mechanisms allowed for the formation of optimized algorithms. These results demonstrate that this algorithm outperforms MP-SPTCP, MP-SDN, and MPTCP in many key streaming metrics. This design developed a general neural network system that is ideal for general video watching, but lacks personalization of content in truly demanding situations.

Further steps were taken in an effort to provide a personalized streaming experience in demanding settings, such as VR over mobile networks. By building upon the success of neural network-based streaming, this final contribution developed a unique multipath-aware approach to designing CDNs which train and distribute at scale, lightweight neural networks in metadata included within video manifest files. This development begins to address limitations commonly seen in modern ABR algorithms, including the lack of personalized streaming experiences. In addition to extending recent research using neural networks client-side with the aid of an SDN network, the SDN limitation was removed thus leading to improved performance and versatility. The results from this contribution illustrate that the estimated mean opinion score was at least 20% higher when using MP-CDN as compared to MP-DANE, MPTCP or MP-SPTCP. Furthermore, MP-CDN utilized the network 15% more efficiently in networks with competing video streams. These results demonstrate that this algorithm outperforms MP-DANE, MP-SPTCP, and MPTCP in many fundamental streaming metrics. One limitation not addressed in this work however, is managing and distributing neural networks across CDNs.

## 8.2 Future Work

In the future, this work could take shape in many directions. One area that could be potentially explored is an investigation of various MPTCP path management algorithms for pushing data across individual subflows. Research also assessing the impact of TLS on HTTP/2 in DASH over multiple paths is of interest. Obtaining experimental results under conditions where asymmetric links end at the client device need to be further evaluated. The examination of other MPTCP path management algorithms to further compare this work should also be explored in addition to obtaining results for multiuser wireless conditions while varying network model types in real time on a massive scale.

Continued efforts in expanding this work in the future should encompass various network model types, such as 5G, mmWave technology, and other cutting edge networking models. The offline training period of neural networks needs to be reduced in future iterations of this design. Lastly, future work should consider expanding upon the ease of interoperability with existing CDNs as this research relied solely on a single CDN to service the entire network.

This dissertation presents research which culminates in the expansion of novel contributions with the development of a learning adaptive transport selection framework used in multipath networks. Using a combination of varying real-world testbeds, simulation-based experimentation, and application development informed the design of the network-assisted learning adaptive transport protocols which were tested on collected networking traces. Underlying limitations of MPTCP-based video streaming have been addressed and as a result, new algorithms that are specifically designed to operate in multipath networks within everyday, modern devices are now available for deployment.

## REFERENCES

- [1] A. Zambelli and M. Corporation, “IIS Smooth Streaming Technical Overview,” Tech. Rep., Mar. 2009, 18 pp.
- [2] *Http dynamic streaming specification*, <https://www.adobe.com/devnet/hds.html>, Accessed: 2017-04-01.
- [3] *Apple http live streaming*, <https://developer.apple.com/resources/http-streaming>, Accessed: 2017-04-01.
- [4] 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, San Jose, CA, USA: ACM, 2011, pp. 133–144, ISBN: 978-1-4503-0518-1.
- [5] *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](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.
- [6] *Dash-264 javascript reference client*, <http://dashif.org/reference/players/javascript/index.html>, Accessed: 2016-08-09.
- [7] Y. P. G. Chowrikoppalu, “Multipath adaptive video streaming over multipath tcp,” PhD thesis, Intel, 2013.
- [8] S. Akhshabi, A. C. Begen, and C. Dovrolis, “An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http,” in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys ’11, San Jose, CA, USA: ACM, 2011, pp. 157–168, ISBN: 978-1-4503-0518-1.
- [9] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, “BOLA: near-optimal bitrate adaptation for online videos,” *CoRR*, vol. abs/1601.06748, 2016.
- [10] M. Azumi, T. Kurosaka, and M. Bandai, “A qoe-aware quality-level switching algorithm for adaptive video streaming,” in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–5.
- [11] Y. Chen, F. Zhang, K. Wu, and Q. Zhang, “Qoe-aware dynamic video rate adaptation,” in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

- [12] 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, Tokyo, Japan: ACM, 2015, 12:1–12:8, ISBN: 978-1-4503-3554-6.
- [13] F. Chiariotti, “Reinforcement learning algorithms for dash video streaming,” PhD thesis, University of Padova, 2015.
- [14] *Information technology – dynamic adaptive streaming over http (dash) – part 5: Server and network assisted dash (sand)*, ISO/IEC 23009-5, International Organization for Standardization, 2017.
- [15] 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, Barcelona, Spain: ACM, 2013, pp. 455–468, ISBN: 978-1-4503-1953-9.
- [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, 2016.
- [17] 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, 2016.
- [18] R. Fielding, J. Gettys, J. Mogul, and H. Frystyk, *Hypertext transfer protocol – http/1.1*, RFC 6824, Internet Engineering Task Force, 1999.
- [19] H. Schulzrinne, S. Casner, R. Frederick, and Jacobson, *Rtp: A transport protocol for real-time applications*, RFC 3550, Internet Engineering Task Force, 2003.
- [20] M. Belshe, R. Peon, and M. Thomson, *Hypertext transfer protocol version 2 (http/2)*, RFC 7540, Internet Engineering Task Force, 2015.
- [21] C. Mueller, S. Lederer, C. Timmerer, and H. Hellwagner, “Dynamic adaptive streaming over http/2.0,” in *Multimedia and Expo (ICME), 2013 IEEE International Conference on*, 2013, pp. 1–6.
- [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, Singapore, Singapore: ACM, 2014, 37:37–37:42, ISBN: 978-1-4503-2706-0.
- [23] R. Huysegems, J. van der Hooft, T. Bostoen, P. Rondao Alface, S. Petrangeli, T. Wauters, and F. De Turck, “Http/2-based methods to improve the live experience of adaptive streaming,” in *Proceedings of the 23rd ACM International Conference*

on *Multimedia*, ser. MM '15, Brisbane, Australia: ACM, 2015, pp. 541–550, ISBN: 978-1-4503-3459-4.

- [24] W. Cherif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori, “Dash fast start using http/2,” in *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '15, Portland, Oregon: ACM, 2015, pp. 25–30, ISBN: 978-1-4503-3352-8.
- [25] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, *Tcp extensions for multipath operation with multiple addresses*, RFC 6824, Internet Engineering Task Force, 2013.
- [26] 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)*, 2017, pp. 68–73.
- [27] —, “Omnidirectional adaptive bitrate media delivery using mptcp/quic over an sdn architecture,” in *Global Telecommunications Conference, (GLOBECOM)*, 2017.
- [28] 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 ()*, 2014, pp. 3044–3050.
- [29] 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)*, 2015, pp. 485–491.
- [30] 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*, 2015, pp. 9–16.
- [31] H. Nam, D. Calin, and H. Schulzrinne, “Towards dynamic mptcp path control using sdn,” in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 286–294.
- [32] R. Stewart, *Stream control transmission protocol*, RFC 4960, Internet Engineering Task Force, 2007.
- [33] S. Wee, J. Apostolopoulos, W. tian Tan, and S. Roy, “Research and design of a mobile streaming media content delivery network,” in *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, vol. 1, 2003, I–5–8 vol.1.
- [34] *Immersive vr and 360 video at streamable bitrates: Are you crazy?* Beamr, 2016.

- [35] M. Khlewind and B. Trammell, “Applicability of the QUIC Transport Protocol,” Internet Engineering Task Force, Internet-Draft draft-ietf-quic-applicability-00, Jul. 2017, Work in Progress, 9 pp.
- [36] *A quic update on googles experimental transport*. <http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html>, Accessed: 2017-04-01.
- [37] 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, Salamanca, Spain: ACM, 2015, pp. 609–614, ISBN: 978-1-4503-3196-8.
- [38] P. Biswal and O. Gnawali, “Does quic make the web faster?” In *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6.
- [39] A. Langley, *Transport layer security (tls) snap start*, Internet Engineering Task Force, 2010.
- [40] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, *Tcp fast open*, RFC 7413, Internet Engineering Task Force, 2014.
- [41] *Taking google’s quic for a test drive*, <http://www.connectify.me/blog/taking-google-quic-for-a-test-drive/>, Accessed: 2018-05-02.
- [42] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX, Monterey, California: ACM, 2010, 19:1–19:6, ISBN: 978-1-4503-0409-2.
- [43] *Project floodlight*, <http://www.projectfloodlight.org/floodlight/>, Accessed: 2018-05-02.
- [44] 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, Mar. 2008.
- [45] T. Uzakgider, C. Cetinkaya, and M. Sayit, “Learning-based approach for layered adaptive video streaming over sdn,” *Computer Networks*, vol. 92, pp. 357 –368, 2015, Software Defined Networks and Virtualization.
- [46] N. Kukreja, G. Maier, R. Alvizu, and A. Pattavina, “Sdn based automated testbed for evaluating multipath tcp,” in *2016 IEEE International Conference on Communications Workshops (ICC)*, 2016, pp. 718–723.

- [47] M. Claeys and S. Latr, “Design and optimisation of a (fa)q-learning-based http adaptive streaming client,” *Connection Science*, vol. 26, no. 1, pp. 25–43, 2014. eprint: <http://dx.doi.org/10.1080/09540091.2014.885273>.
- [48] F. Chiariotti, S. D’Aronco, L. Toni, and P. Frossard, “Online learning adaptation strategy for dash clients,” in *Proceedings of the 7th International Conference on Multimedia Systems*, ser. MMSys ’16, Klagenfurt, Austria: ACM, 2016, 8:1–8:12, ISBN: 978-1-4503-4297-1.
- [49] H. Mao, R. Netravali, and M. Alizadeh, “Neural adaptive video streaming with pen-sieve,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM ’17, Los Angeles, CA, USA: ACM, 2017, pp. 197–210, ISBN: 978-1-4503-4653-5.
- [50] V. Mnih and Puigdomènech Badia, “Asynchronous Methods for Deep Reinforcement Learning,” *ArXiv e-prints*, Feb. 2016. arXiv: 1602.01783 [cs.LG].
- [51] *Deepmind research*, <https://deepmind.com/research/>, Accessed: 2017-06-06.
- [52] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. D. Turck, “A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients,” in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 131–138.
- [53] B. Hayes, Y. Chang, and G. Riley, “Controlled unfair adaptive 360 vr video delivery over an mptcp/quic architecture,” in *International Conference on Communications, (ICC)*, 2018, to appear.
- [54] Y. Liu, S. Dey, F. Ulupinar, M. Luby, and Y. Mao, “Deriving and validating user experience model for dash video streaming,” *IEEE Transactions on Broadcasting*, vol. 61, no. 4, pp. 651–665, 2015.
- [55] *Itec dynamic adaptive streaming over http - dataset 2014*, <http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/>, Accessed: 2017-04-01.
- [56] *X264*, <http://www.videolan.org/developers/x264.html>, Accessed: 2017-04-01.
- [57] 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, Vienna, Austria: Springer-Verlag, 2012, pp. 32–41, ISBN: 978-3-642-28536-3.



- [58] Y. C. Chen and D. Towsley, “On bufferbloat and delay analysis of multipath tcp in wireless networks,” in *2014 IFIP Networking Conference*, 2014, pp. 1–9.
- [59] *Google cardboard camera app*, <http://storage.googleapis.com/cardboard-camera-converter/index.html>, Accessed: 2017-04-01.
- [60] M. Uitto and A. Heikkinen, “Sand-assisted encoding control for energy-aware mpeg-dash live streaming,” in *2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2016, pp. 1–5.
- [61] *Omnidirectional media application format (omaf)*, ISO/IEC 23000-20, 2016.
- [62] B. Hayes, Y. Chang, and G. Riley, “Scaling 360-degree adaptive bitrate video delivery over an sdn architecture,” in *International Conference on Computing, Networking and Communications, (ICNC)*, 2018, to appear.
- [63] G. Fairhurst, A. Sathaseelan, and R. Secchi, *Updating tcp to support rate-limited traffic*, RFC 7661, Internet Engineering Task Force, 2015.
- [64] M. Abadi and A. Agarwal, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, 2015.
- [65] J. M. Pullen, “The network workbench: Network simulation software for academic investigation of internet concepts,” *Comput. Netw.*, vol. 32, no. 3, pp. 365–378, Mar. 2000.
- [66] H. Blumenstein and B. J. Reeber, *The rise of online video on the tv screen*, think with Google, 2017.
- [67] A. Agrawal, *2018 video trends on the rise*, The Next Web, 2018.
- [68] R. Banerji, *Digital media: Rise of on-demand content*, Deloitte, 2018.

## VITA

Brian D. Hayes attended Georgia Institute of Technology for his Ph.D. and M.S. degree in Electrical and Computer Engineering. He attended the University of Maryland - College Park for his B.S. in Computer Engineering. He was born in Washington, D.C. and lived in Fort Washington, MD his entire childhood. Brian has worked for many corporations during his time in school; Northrop Grumman as a Digital Technology Test Engineer, Broadcom as a Residential Broadband Test Engineer; Intel Corporation in a few different roles, namely validator in the HIPs (Hard IP) Chipset Circuit Group, validator in the MIC (Many Integrated Core) Visual and Parallel Computing Group, and finally as a hardware validator in the Digital Home Group. Brian also worked at General Electric as a Quality Assurance Engineer in Aviation Supply Chain, Georgia Tech Research Institute as a graduate research assistant, and Lexmark International as a low-end firmware/hardware engineer. Furthermore, Brian worked with United Technologies Corporation Sikorsky Aircraft, and finally NASA's Goddard Space and Flight Center. His technical skills include computer network security, computer architecture, and high speed digital system testing among others. He is knowledgeable of many programming languages including C, C++, JAVA, Perl, Prolog, MATLAB, Verilog, Tcl. He also has numerous IEEE technical publications.