Squash: low latency multi-path video streaming using multi-bitrate encoding

by

Joohan Lee

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The demand for low latency video streaming has dramatically increased as live video streaming applications, such as Twitch and Youtube Live, are becoming more popular. According to the 2021 Bitmovin video developer report, the biggest challenge that video developers are experiencing today is providing low latency video streaming. One of the most common on-site live streaming methods is using a wireless LTE network. There have been many approaches for characterizing wireless links and accurately measuring available bandwidth to provide low latency streaming over a wireless LTE network link. However, even with fine-grained bandwidth estimation, video streaming on a single LTE link is still susceptible to unexpected network delay from a sudden drop in available bandwidth or temporal disconnection.

People can utilize multiple wireless LTE links to overcome the limitations of using a single LTE link for low latency video streaming. Using multiple links can enhance video quality through increased bandwidth and resilience. However, multi-homed low latency video streaming protocols may achieve lower video quality than single-homed protocols when a frame is split and sent over more than one link. Suppose one of the links becomes congested or gets disconnected. In that case, the part of the frame sent on stable links must wait until the packets sent on the problematic link are re-transmitted through another link. Re-transmission requires at least one extra round trip time. A video player may skip the late frame or serve only the received part of the frame due to the re-transmission delay. Ferlin et al. suggest using Forward Error Correction (FEC) on Multipath TCP (MPTCP) to reduce re-transmission delay. However, FEC is not helpful in the event of a significant bandwidth drop. If the sender does not use sufficient redundancy to handle a significant bandwidth drop, the receiver will not receive enough blocks to decode the video data. FEC requires using a large portion of the network bandwidth for redundancy to handle significant bandwidth drops even when the links are stable.

In this thesis, I present Squash, a low latency video transport protocol that encodes each frame at multiple bitrates and sends them across different links to minimize video stream disruption in the event of unexpected bandwidth drops. The encoder encodes a frame into multiple different bitrates, which are high-bitrate and low-bitrate. When a high-bitrate frame cannot arrive on time due to congestion from an unexpected drop in available bandwidth, the low-bitrate frame is used to replace the missing frame. This is because the low-bitrate frame is smaller and is sent on the links that are disjoint from those used by the high-bitrate frame. To the best of my knowledge, Squash is the first architecture that uses multi-bitrate frames to increase resilience against unexpected bandwidth drops in low latency video streaming over multiple wireless LTE links. In emulated wireless

LTE network environment using Mahimahi network traces, the average SSIM of the video streamed on Squash is 13 – 58% higher than that streamed on the baseline protocol, which is designed in the same manner as Squash except that it employs single-frame encoding.

Acknowledgements

I want to thank Professors Bernard Wong and Khuzaima Daudjee for their support and guidance over the last two years at the University of Waterloo. My understanding of research could be improved by the experience with you. I also want to thank Sharon Choy for helping me throughout my entire graduate research.

Dedication

This thesis is dedicated to my family in Korea and to my father, who watches over me from heaven.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ABR Adaptive BitRate 1, 5–7, 34

ARQ Automatic Repeat reQuest 2, 5, 32

AVC Advanced Video Coding 6

B-frame bi-directional predicted frames 31

BBR Bottleneck Bandwidth and Round-trip propagation time 12

FEC Forward Error Correction 2, 3, 5, 8, 36

FHD Full High Definition 25, 41

FPS Frames Per Second 14, 16, 25

HEVC High Efficiency Video Coding 6

I-frame keyframes 31

LTE Long-term evolution 1, 2, 4, 5, 7, 9, 10, 12, 22, 25–34, 37

MPTCP Multipath TCP 31, 32, 36

MTU Maximum Transmission Unit 15, 17

P-frame predictive frames 31

QoE Quality of Experience 1–3, 6

RTT Round Trip Time 2, 12, 13, 19, 26, 32

SSIM Structural Similarity Index 25–34, 36, 37

WebRTC Web Real-Time Communication 1, 2, 38

Chapter 1

Introduction

In recent years, there has been a rise in the popularity of live video streaming applications, such as Twitch, YouTube Live, and Instagram Live, that allow users to communicate with each other in real time. These interactive live streaming applications require a low end-to-end delay, the time it takes for an event to be captured and displayed on a user's screen. On-site streaming has become prevalent in live video streaming with the proliferation of wireless LTE networks and mobile devices. However, wireless LTE networks experience high variability in available bandwidth and latency. The variability in LTE networks presents challenges for low latency streaming because it requires a short frame deadline. An unexpected bandwidth drop or temporary disconnection in a wireless network increases network delay. To avoid long end-to-end delay, the video player skips the frames that arrive late due to the increased network delay [57]. Skipping a frame significantly decreases the Quality of Experience (QoE) from lagging or freezing.

Many buffer-based Adaptive BitRate (ABR) protocols [33, 46, 62, 64, 77] use a playback buffer to mitigate the lagging or freezing of video from sudden bandwidth drops or temporal disconnection. The video player preloads future frames in the playback buffer, and these protocols adapt the video bitrate according to playback buffer occupancy. High playback buffer occupancy enables the video stream to withstand network congestion. In live streaming, however, video frames are encoded in real time, and the video player cannot preload the future frames to increase the playback buffer occupancy. Low latency video streaming protocols usually use a very small or no playback buffer because using a large playback buffer increases the end-to-end delay in live streaming.

An alternative approach is using Web Real-Time Communication (WebRTC) [2], which is one of the most widely used real-time video streaming protocols. It quickly adapts

to changing network capacity without using a large playback buffer. However, frequent video bitrate changes resulting from fluctuating wireless LTE network capacity causes WebRTC's QoE to suffer. This is because there is additional delay between when the encoder is notified of the new video bitrate and when the generated frames reflect this updated bitrate [79]. WebRTC [2] does not consider this delay. In a wireless LTE network environment, the estimated bandwidth significantly changes on every RTT, which has millisecond granularity. On the other hand, the updated video bitrate is applied over the next 1 to 2 seconds [26]. If the network bandwidth drop is detected after the frame encoding has already started, the updated encoding rate is averaged over the next several frames instead of the current frame. As a result, the link can become congested due to the larger frame.

Salsify [26] reduces the delay in applying the changed video encoding rate to the new frames by integrating video codec and transport protocol. The video encoder combined with transport protocol can quickly adapt to fluctuating network bandwidth. However, Salsify [26] requires a custom video codec to keep the internal encoding state because most existing hardware codecs do not support it [79]. Moreover, Salsify [26] uses a single link. If a single link experiences an unexpected bandwidth drop or temporal disconnection, which often happens in wireless LTE networks [31, 72], the frame is likely to be skipped because of deadline violation from the increased network delay.

One solution to overcome the variability of wireless LTE networks is to utilize multiple LTE network interfaces. Using multiple links provides many advantages, including increased available bandwidth for the application through aggregation and sustaining some link failures. When one link is congested, the application supporting Automatic Repeat reQuest (ARQ) [21] can re-transmit the packets sent on the congested link to the stable link. However, using ARQ incurs additional delays in detecting and resending a lost packet, which requires at least one extra round trip [24]. This may cause a frame deadline violation. For example, if part of a frame is lost, the receiver must wait for the lost data to be re-transmitted. The re-transmission delay increases the end-to-end delay and reduces the on-time frame arrival rate.

An alternative to ARQ is Forward Error Correction (FEC) [48], which preemptively sends recovery data. However, FEC risks using excess bandwidth and can only tolerate a bandwidth drop that is proportional to the amount of recovery data. Wireless LTE networks are susceptible to a large bandwidth drop. Supporting them would require significant replication. If the available bandwidth is predominantly used for recovery data to handle significant bandwidth drop, then the video encoding rate will be low. When the link quality is good, it should not use excessive network capacity.

In this thesis, I present Squash: a multi-homed, multi-bitrate, low latency video streaming protocol that provides high quality video with minimal recovery bandwidth usage. Squash encodes and transmits multiple versions of the same video frame: the high-bitrate frame and the low-bitrate frame. A high-bitrate frame is used as a primary video stream, and a low-bitrate frame is used as a backup delivered to the user when a high-bitrate frame does not arrive on time. Compared to FEC [48] and replication of data on all links, Squash uses only a small portion of the available bandwidth to send a low-bitrate frame because low-bitrate frame is used to avoid QoE drop from skipping a frame when the high-bitrate frame is discarded from a deadline violation. A small-sized frame also has a higher chance of being transmitted under a significant bandwidth drop.

Squash determines the following parameters to provide high-quality video and resilience for low latency video streaming over multiple links: video encoding rate, number of links to replicate low-quality frame, and on which links to send the low-bitrate frame. To determine these parameters, Squash characterizes each link by using the knowledge of the link's condition, which includes packet latency and delivery rate.

Firstly, Squash needs to determine the video encoding rate of the low-bitrate frame. There is an inherent trade-off between the quality of the low-bitrate frame and the chance that the frame arrives on time under a significant bandwidth drop. A large low-bitrate frame limits the video quality drop from replacing the high-bitrate frame when the high-bitrate frame cannot arrive on time. However, if the network capacity is severely decreased and becomes smaller than the low-bitrate frame size, the frame cannot be delivered within the deadline. Squash allows the user to set the expected bandwidth drop, which is the maximum encoding rate of a low-bitrate frame. Squash ensures that the low-bitrate frame will arrive when the bandwidth drop is less severe than expected. Moreover, Squash considers the minimum bandwidth of each link in encoding the low-bitrate frames so that they can be sent on any link without aggregation. This is because aggregating links to send a frame has a higher risk of deadline violation than sending a frame on a single link. When a frame is sent over the aggregated links, a lost or late packet delivery on any of the links obstructs decoding the frame [11]. Consequently, encoding rates for low-bitrate frames are determined by calculating the expected bandwidth drop and the bandwidth of each link.

Secondly, Squash decides the link(s) to send the low-bitrate frame on because it needs to ensure that the low-bitrate frame can be provided to the decoder in case the high-bitrate frame does not arrive on time. To accomplish this, Squash estimates each link's stability by calculating the ratio of expected acknowledged video packets to acknowledged video packets. This ratio is referred to as the stability score. Squash prioritizes links to send a low-bitrate frame by stability score. When more than one link has the same stability

score, Squash compares the estimated bandwidth of the links. Squash sends through the link with higher network bandwidth because the link will have higher bandwidth when all links drop the bandwidth in the same ratio.

Lastly, Squash must decide on how many links it should replicate the low-bitrate frame to. Squash has to ensure that the low-bitrate frame arrives on time during an unexpected link disconnection. If the low-bitrate frame is replicated on many links, the resilience of the video stream increases; however, the video encoding rate of both types of frames decreases. Chapter 4.3, explore different replication strategies for sending a low-bitrate frame. Squash supports both dynamic and static numbers of replications. The static replication strategy sends a low-bitrate frame either on a single or all available links. On the other hand, the dynamic replication strategy decides the number of replications based on link stability (Chapter 3.4.2). I could not determine a single strategy that is appropriate for all workloads. During the experiments, however, replicating the low-bitrate frame on multiple links improves the video quality when the links have variable bandwidth.

This thesis makes the following contributions:

- Squash uses multi-bitrate frames to address the challenges of low latency video streaming over heterogeneous wireless LTE networks.
- Squash characterizes each heterogeneous link and schedules video packets to send.
- Squash determines the video bitrates on the multi-bitrate encoder, the number of links to replicate a backup frame, and which link(s) to send the backup frame.
- The experiments (Chapter 4.2) show that the impact of sudden bandwidth drops is severe. Utilizing a small portion of the overall network bandwidth, Squash efficiently handles unexpected network bandwidth drops.

Chapter 2

Related Work & Background

This chapter explains Squash's video streaming environment (Chapter 2.1) and presents related work on ABR protocols (Chapter 2.2) and multi-homed transport protocols (Chapter 2.3). There are many multi-homed ABR protocols for aggregating links and handling link failure and congestion by utilizing FEC or ARQ. However, to the best of my knowledge, there does not exist a low latency video protocol that uses multiple video bitrate encodings to mitigate the impact of congestion in multi-homing environment.

2.1 System Environment

Squash provides low latency video streaming over mutliple wireless LTE network. The following chapters describe the network environment (Chapter 2.1.1), other applications that use multi-bitrate encoding (Chapter 2.1.2), and the possible video codecs (Chapter 2.1.3).

2.1.1 Wireless LTE Network

Squash is designed to support low latency video streaming over a wireless LTE network. A cellular network has a highly variable condition, which is affected by various factors, such as signal strength, the user's location, and competing traffic from other users [32]. Most base stations of wireless LTE networks have exceptionally large buffers to cope with a burst of traffic and channel variability [36]. In addition to a large buffer, link layer re-transmission in wireless LTE networks significantly reduces the probability of packet loss [36]. Even though large buffers and link layer re-transmission conceal packet loss, they

introduce congestive delays in response to poor signal or temporal disconnection. This makes low latency video streaming challenging because network congestion increases the delay in transporting a frame. The increased transport delay causes a frame skip that deteriorates the user's QoE.

2.1.2 Multi-bitrate Encoding

The advancement in video encoding technology and hardware allowed multi-bitrate encoding for popular video conferencing applications, such as Skype and Google Hangouts. These applications utilize multi-bitrate encoding to provide video frames that fit into each receiver's network capacity [75]. The use of multi-bitrate encoding in these applications demonstrates that the additional overhead for video encoding is sustainable on current commodity devices.

2.1.3 Video Codec

Squash uses H.264 codec to encode a video frame captured by a camera. According to the bitmovin video developer report 2021 [7], the two most commonly used video codecs are Advanced Video Coding (AVC) and High Efficiency Video Coding (HEVC). In live streaming, the speed of encoding and the amount of compression have a large impact on the user's quality of experience. The encoding speed affects the glass-to-glass latency, and the compression rate affects the video's image quality as the available network bandwidth is limited. HEVC has a higher compression rate, but its higher complexity slows down the encoding speed. In the testbed, outlined in Chapter 4.1, the x265 encoder was not able to finish encoding a video frame before the next frame's arrival. I decided to use AVC in order to reduce the encoding delay.

VP8 [71] is a widely used codec that provides a similar compression rate to H.264 and it is under a royalty-free public license. However, Squash uses H.264 instead of VP8 because other works [22, 58, 59] have shown that H.264 outperforms VP8 in both compression quality and speed of encoding.

2.2 Adaptive Bitrate Streaming

Streaming video over a wireless network requires ABR because the available bandwidth fluctuates over time. ABR algorithms change the video encoding rate according to link

status information in order to avoid over-sending, which causes link congestion and video stalls. However, many proposed ABR protocols are designed for on-demand video streaming over a single link with high latency requirements. These protocols can be categorized as buffer-based [33, 46, 62, 77], rate-based [37, 66], or a hybrid of the aforementioned approaches.

In a buffer-based protocol, the playback buffer's occupancy on the receiver is used to select the desired video bitrate. The receiver requests higher-quality video if the number of preloaded frames in the playback buffer exceeds the threshold. Using high buffer occupancy can withstand congestion because a buffer having high occupancy can be used to mitigate fluctuating network conditions. However, the buffer having high occupancy increases the end-to-end delay, the time it takes for an event to be captured and displayed on a user's screen. The video player does not start playing until it has enough preloaded frames in the playback buffer. Buffer-based protocols are more appropriate for on-demand video streaming than for low latency, live video streaming.

Rate-based protocols estimate the currently available bandwidth. Examples include FESTIVE [37] and CS2P [66], which use the harmonic mean of throughput information collected from the sample video chunk downloads. However, these methods require collecting samples from several video chunk downloads for accurate bandwidth estimation. Doing so takes time; thus, this method is inappropriate for low latency streaming over wireless LTE networks with highly fluctuating bandwidth.

Squash aims to provide low latency video streaming over wireless LTE links having highly variable bandwidth. Squash quickly adapts the video bitrate to the variable bandwidth by updating the link's sending rate on each acknowledgement message instead of a video chunk download. Squash also supports stable video streaming under unexpected bandwidth drops with a small playback buffer by utilizing multi-bitrate frames.

2.3 Existing Approaches Using Multiple Links

Using multiple wireless LTE network links can provide higher throughput through aggregation. However, congestion on one link can affect all of the aggregated links. When the sender sends a video frame over the aggregated links, the received part of the frame has to wait in the receiver buffer until the rest of the frame sent on the congested link arrives. This increases the delay to transmit a frame and causes a frame to skip due to a deadline violation in low latency streaming.

Several multi-path transport protocols [51, 63, 68] provide resilience against link congestion by re-transmitting the lost packets through an alternative link. However, it causes additional delays in detecting the congestion and resending the packets. The increased transport delay may result in a frame skip due to a deadline violation in low latency streaming.

Another way to deal with link congestion in a multi-homing environment is to add redundancy to video data, such as replicating a frame on multiple links or using FEC [48]. Preemptively sending redundant video data provides resilience without requiring additional delay to re-transmit the packets sent on congested links. Replicating a frame on all links ensures the frame arrival if at least one link is available, but it is not efficient because it wastes a large portion of the network bandwidth. The degree of replication increases as more links are available. When there are more than two links, it uses more than half of the available links to send the replicated video data. In addition, each link may have different network bandwidth. Some links can experience congestion from over-sending, while other links are underutilized.

Compared to replicating a frame on all available links, FEC is a more efficient way to provide resilience against link congestion on aggregated links. Several protocols [11, 24, 73] send recovery data using some portion of the aggregated links' bandwidth. The receiver can use the recovery data when part of the frame does not arrive on time due to congestion. However, the amount of recovery data determines how much bandwidth drop can be tolerated. It risks using excess bandwidth when the links are stable to prepare for a significant bandwidth drop.

Squash utilizes multi-bitrate frames to provide resilience against a significant bandwidth drop without using excess bandwidth when the links are stable. Squash increase the availability of video frames in more efficient way than FEC by encoding a frame at low and high bitrates. As high bitrate frames are the main video stream, low bitrate frames use only a small portion of the overall bandwidth to mitigate the impact of a significant bandwidth drop on video quality.

Chapter 3

Architecture and Design

This chapter describes Squash's design to support low latency streaming over multiple wireless LTE networks. Chapter 3.1 describes Squash's protocol design and its process of video streaming. Chapters 3.2 to 3.6 describe each component of Squash and its role.

3.1 Squash Protocol Design

Squash is a low latency video transport protocol that delivers multimedia content across multiple wireless LTE network links. Its design is motivated by the challenges of providing on-site live video streaming. These applications send time-critical content and must handle the latency and bandwidth variability of public shared wireless LTE networks.

Squash provides stable on-time delivery of video frames by leveraging hardware support for using multi-bitrate frame encoding that encodes the same stream at different rates. The high-bitrate frame (primary frame) is the main video stream that is delivered to the user when links are stable. Squash delivers the low-bitrate frame (backup frame) to the user when the primary frame does not arrive on time. In the event of link congestion from significant bandwidth drops, sending a backup frame increases the likelihood of avoiding frame skips due to a deadline violation without consuming excessive bandwidth to add redundancy. Although the backup frame does not offer the same level of video quality as the primary frame, it reduces the number of skipped frames that significantly decrease video quality.

Additionally, Squash improves the quality of video encoding by aggregating the available links when the links are stable. To support link aggregation, Squash divides a frame

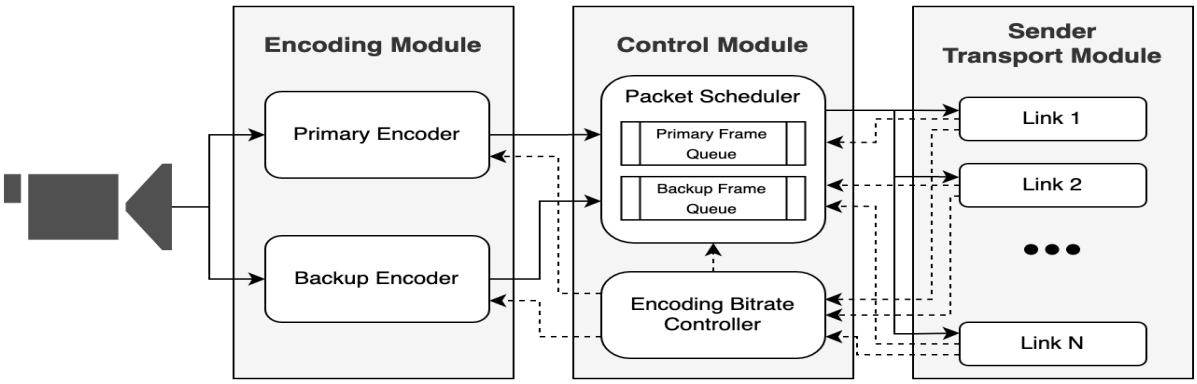

Figure 3.1: Squash Sender. Each link manages its status, which includes its estimated bandwidth(Chapter 3.4.1) and stability score (Chapter 3.4.2). The link status is updated on every acknowledgement. The Encoding Bitrate Controller collects each link's status information and decides the encoding rates of the primary frame and the backup frame. The encoded video frame is partitioned into multiple packets and distributed over the multiple links. The number of packets sent on a link depends on the link's available bandwidth.

into multiple packets and sends the packets using different links. This increases the video encoding rate at the cost of potentially increasing the probability that Squash falls back to using a backup frame because part of the primary frame does not arrive before the deadline. Squash's main challenge is determining the video stream to send on each link and the encoding rate of each video stream. Squash decides which links to send video data and the video bitrates of the encoders based on each link's bandwidth estimation (Chapter 3.4.1) and quantified stability (Chapter 3.4.2). In the next chapter, I describe the different components of Squash.

3.2 Modules of Squash Sender

A Squash deployment consists of a sender and a receiver that communicates using the Squash modules over multiple wireless LTE network interfaces. Figure 3.1 illustrates Squash's sender components. The sender consists of the **Camera**, the **Encoding Module**, the **Control Module**, and the **Sender Transport Module**.

3.3 Encoding Module

The Encoding Module receives a raw video frame from the Camera that captures the picture for video streaming. The Encoding Module then encodes the raw video frame at different bitrates to send over the network. The Encoding Module contains the Primary Encoder and the Backup Encoder, which are used to generate two separate video streams. The Primary Encoder encodes the raw video frames at a high bitrate, which forms the main video stream. This stream is delivered to the user when the links are stable. The Backup Encoder encodes the raw video frames at a low bitrate to create a secondary stream. This stream is used when the primary frames of the main stream do not arrive at the receiver before their deadline. The Encoding Module passes the encoded video frames to the Control Module. The video bitrate of each encoder is assigned by the Encoding Bitrate Controller in the Control Module before it starts encoding the next frame. The next chapter explains about the Sender Transport Module because the Control Module requires information about each link to make a decision.

3.4 Sender Transport Module

The Sender Transport Module manages each link component that sends the assigned packets through the network interface. The link component updates its estimated bandwidth (Chapter 3.4.1) and stability score (Chapter 3.4.2) every time it receives an acknowledgement. After updating the information, the link component notifies the Encoding Bitrate Controller in the Control Module to determine the new video bitrates for subsequent video frames from the Camera. The link component also shares its estimated bandwidth and stability score with the Packet Scheduler in the Control Module when the Packet Scheduler assigns packets to send. The following chapters explain how the link measures bandwidth and quantifies the link' stability.

3.4.1 Sending Rate Control

Inaccurate bandwidth estimation can cause under-utilization of the network or traffic congestion due to over-sending. Avoiding network congestion is critical in low latency video streaming because frames have short deadlines. Even though Squash mitigates significant video quality loss resulting from skipped frames by using backup frames from the secondary stream, the video quality is still lower when compared to using a primary frame

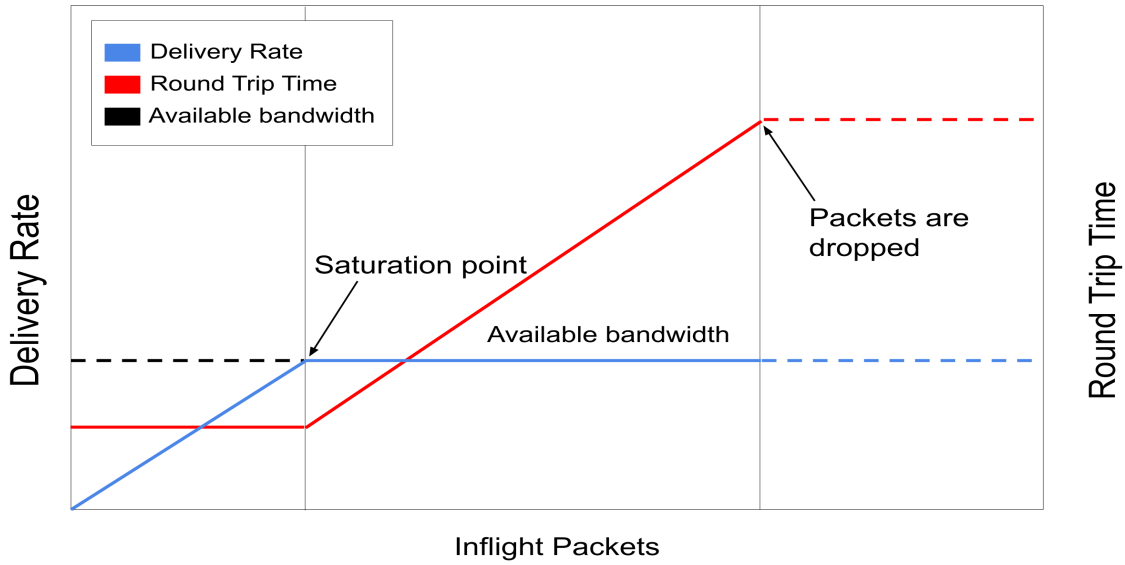

Figure 3.2: The delivery rate and RTT versus the number of in-flight packets. When the sender continues to increase the sending rate, the delivery rate does not exceed beyond the available bandwidth. The packet latency increases until the buffer becomes full and starts dropping packets.

from the main stream. Squash needs accurate bandwidth estimation to maximize the usage of primary frames. However, the bandwidth of a wireless network is highly variable, and providing accurate bandwidth estimation is challenging. Squash addresses this challenge by introducing its bandwidth estimation algorithm, which is inspired by Bottleneck Bandwidth and Round-trip propagation time (BBR) [8] and Copa [6]. Squash adjust these approaches for low latency video streaming over a wireless LTE network.

The link module estimates the available bandwidth by utilizing the delivery rate, which is measured by the receiver by calculating how many bytes were received over the last 500 ms. The delivery rate becomes the same as the link's available bandwidth when the link is saturated. Figure 3.2 illustrates delivery rates and packet latency when the sender increases the number of in-flight packets. The delivery rate does not increase when the sender sends more than the available bandwidth. Instead, the packets are queued in buffers within the network until a buffer becomes full, leading to increased packet latency. As a result, the sender can find the maximum delivery rate by saturating the link. However, it is challenging to determine the saturation point without causing a frame deadline violation because a saturated link can become congested from over-sending. When the link becomes

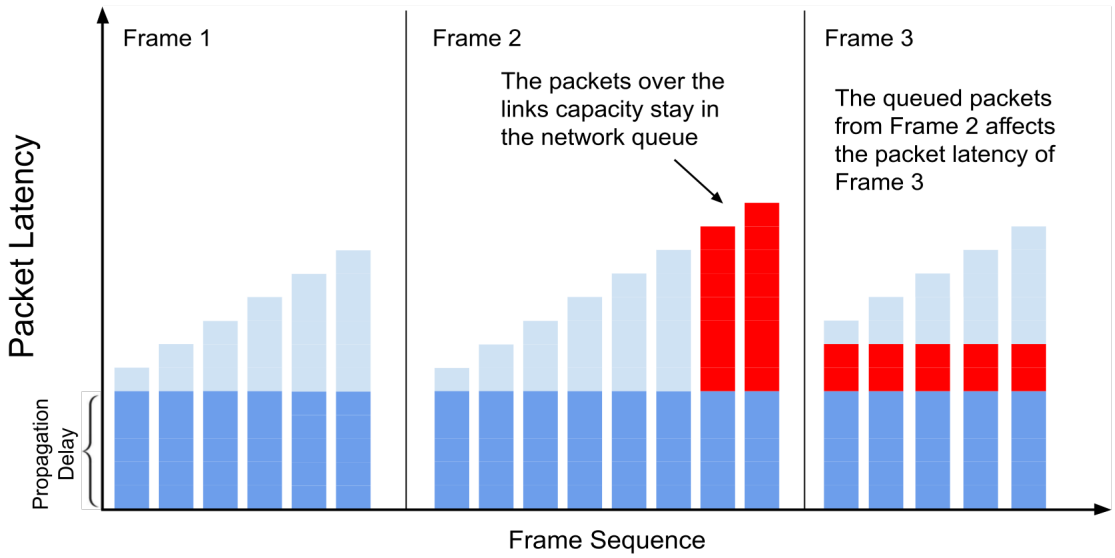

Figure 3.3: This is the latency of each packet when a frame is partitioned and sent as multiple packets. The sender bursts multiple packets when it sends a frame on the link. The later packets in the burst have higher latency because of the delay to process earlier packets in the burst. If the packets from the previous frame are still in the link, the minimum packet latency of the current frame is increased.

congested from over-sending, the packets sent on the link have to wait for the link's buffer to flush. To detect link saturation before the link becomes congested, the link module compares two groups of windowed RTT measurements.

The link module compares the true minimum RTT (RTTmin) against the standing RTT (RTTstanding) to measure the queuing delay. The queuing delay is calculated by $RTTstanding - RTTmin$. RTTmin is the minimum RTT over the last 10 seconds. RTTstanding is the minimum RTT of the packets in the most recently acknowledged frame. The link module sets the window of RTTmin to 10 seconds to handle route changes that might affect the minimum RTT of the path [6]. Considering the interval of video data availability and the variable encoding speed, the link module sets the window of RTTstanding to the most recently acknowledged frame.

The link module detects if packets from the previous frame are queued on the link by measuring the queuing delay. If packets from the previous frame are queued, the following packets experience queuing delay which results in increased RTTstanding. The sender sends a burst of packets when a frame is produced at the video's frame rate. Frame rate is the

frequency at which a camera captures an image; it is typically represented as Frames Per Second (FPS). For example, a frame rate of 25 FPS means that a frame is captured every 40 ms. Squash aims to send the size of a frame that can be transmitted within 40 ms so that it does not cause queuing delay that increases the network delay of subsequent video frames.

Figure 3.3 illustrates each packet's latency when the Squash sender sends three video frames on a link that can transmit five packets within a frame interval. Packets from **Frame 1** do not experience queuing delay because there is no previous frame. In this example, these packets are transmitted before **Frame 2** is sent on the link. As a result, the RTTstanding from **Frame 1** and **Frame 2** is the same as the RTTmin. However, **Frame 2** is too large to send completely within one frame interval and the link cannot process all the packets from **Frame 2** before the sender transmits **Frame 3**. Due to the queued packets from **Frame 2**, packets from **Frame 3** experience additional queuing delay. The queuing delay increases the RTTstanding from **Frame 3**'s minimum packet latency. The sender has to decrease the sending rate so that the queued packets are flushed out of the link without delaying the following frames.

Algorithm 1 Sending Rate Update on a Link

Require: $DeliveryRate$ ▷ Received from acknowledgement
Require: $SaturatedDeliveryRate$ ▷ Delivery rate when the link is saturated
Require: $QueuingDelay \leftarrow RTTstanding - RTTmin$
Require: $ExpectedTransmissionDelay \leftarrow \frac{MTU}{DeliveryRate}$
Require: $\beta \leftarrow MTU\ per\ second$ ▷ How much sending rate to increase
  if $QueuingDelay <= ExpectedTransmissionDelay$ **then**
    $DeliveryRate \leftarrow MAX(DeliveryRate, SaturatedDeliveryRate)$
    if $DeliveryRate > SendingRate$ **then**
      $SendingRate \leftarrow DeliveryRate$
    else
      $SendingRate \leftarrow SendingRate + \beta$
    end if
  else
    $SendingRate \leftarrow DeliveryRate \cdot (1 - \alpha \cdot \frac{RTTstanding - RTTmin}{FrameInterval})$
    $SaturatedDeliveryRate \leftarrow DeliveryRate$
  end if

Algorithm 1 illustrates how the link component changes its sending rate. The link component calculates the queuing delay by subtracting RTTmin from RTTstanding. Every

time the link component receives an acknowledgement, it updates the sending rate based on the queuing delay.

The link component increases the sending rate when the measured queuing delay is smaller than or equal to the expected transmission delay, which is defined as how long it takes to send a Maximum Transmission Unit (MTU) packet at the current delivery rate. The link component calculates the expected transmission delay by $\frac{MTU}{DeliveryRate}$. When the queuing delay is smaller than the expected transmission delay, the queued packet is smaller than one MTU packet. The link component increases the sending rate by a constant amount (β) on each acknowledgement. We set the β to 1472 bytes per second, which is based on the MTU in our system.

When the link component increases the sending rate, it utilizes the delivery rate to quickly recover the sending rate from dropping the sending rate due to the latency increase. When the link component detects link saturation, it reduces the sending rate until all queued packets are flushed out. While the link component reduces the sending rate, it keeps track of the delivery rate. As the queued packets are flushed out and the link component starts increasing the sending rate, the link component sets the sending rate to the delivery rate if the delivery rate is larger than the current sending rate.

The link component decreases the sending rate if it detects the link saturation by measuring queuing delay. The link component divides the queuing delay by the Frame Interval to calculate how much it should decrease from the delivery rate to flush the queued packets from the link before the next frame is ready to be sent. To control how aggressively or conservatively the link component reduces the sending rate, Squash takes a parameter α when the application starts. The parameter α reflects the risk tolerance. The user has the domain knowledge of the application requirement. The user can take a risk of higher latency increase from a network bandwidth drop to increase the link utilization by setting a low α. Alternatively, the link component can preemptively reduce the sending rate to avoid significant latency increases by setting a high α. During the experiments that we ran in the evaluation (Chapter 4), Squash provided the highest performance metrics when we set α to 0.5.

3.4.2 Quantifying link stability

Squash quantifies each link's stability to determine how many links it should replicate a backup frame to and which link(s) to send the backup frame. The stability score indicates whether the link module is sending the video data to the destination without significantly increasing queuing delay that can cause a frame deadline violation. The link component

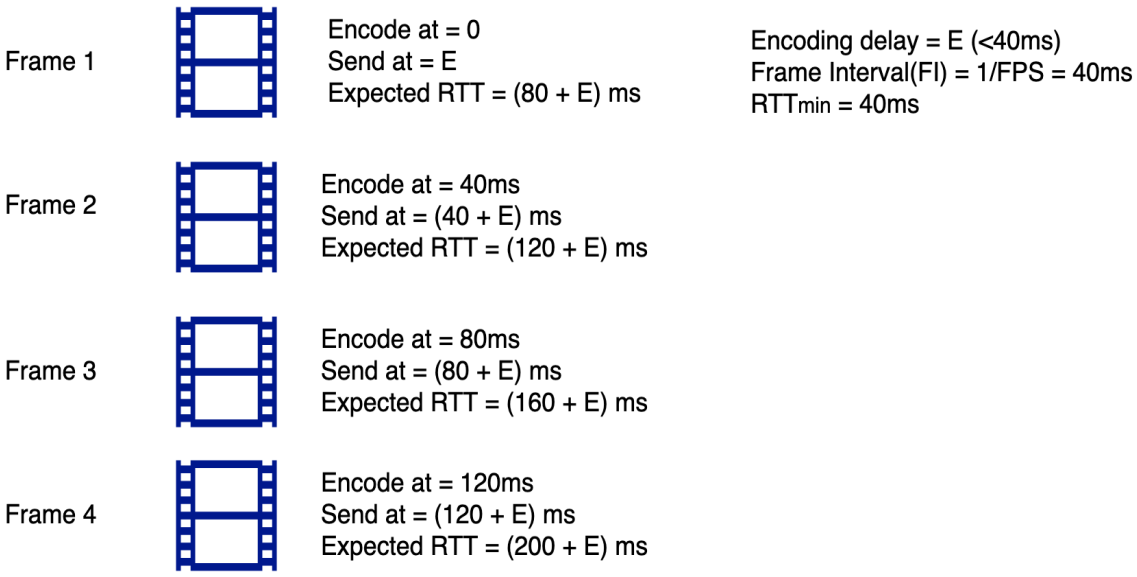

Figure 3.4: Anticipating the frame that expected to be acknowledged from the RTTmin and frame interval

determines the stability score by tracking the number of the acknowledgment messages that arrive in time. The link component calculates the stability score by dividing the number of acknowledged packets by the number of sent packets for each frame. Based on the estimated bandwidth, Squash encodes each frame at the video bitrate that can be fully transmitted within a frame interval. Therefore, the link component should receive an acknowledgement for a frame in (frame interval + RTTmin) ms after it sends the frame.

When all the packets from each frame receive acknowledgements in the expected period, the stability score becomes 1.0 and the link is considered stable. On the other hand, if some of the packets from the frame did not receive acknowledgements in time, it indicates that the link experienced queuing delay from over-sending or jitter from link layer retransmission. As there is more variability in latency, the stability decreases and the link is considered unstable. The Control Module sends a backup frame on a stable link and changes the number of links to replicate a backup frame to by considering how many links are stable.

Figure 3.4 illustrates an example of how the link module estimates the frame that should be acknowledged at each frame interval. The video is recorded at 25 FPS, and the RTTmin is measured to be 40 ms. The video data becomes available every 40 ms because the video is recorded at 25 FPS. The link component expects each frame to be acknowledged

in (Expected Transmission Delay + RTTmin + Encoding Delay) ms after Squash starts encoding the frame. In the example, each frame is expected to be acknowledged in (80 + E) ms. When **Frame 4** starts to be encoded, Squash calculates the stability score of each link module based on the packets from the **Frame 1** as Squash expects that all packets from **Frame 1** to be acknowledged. When Squash starts encoding the **Frame i**, it calculates the stability score based on the packets from the **Frame j**, which is estimated by following the formula 3.1.

$$j = i - (1 + \frac{FrameInterval + RTTmin}{FrameInterval}) \tag{3.1}$$

3.5 Control Module

The Control Module consists of the Packet Scheduler and the Encoding Bitrate Controller. Based on the information from each link component in the Sender Transport Module, the Packet Scheduler assigns the encoded packets to the different link components and the Encoding Bitrate Controller decides the video encoding bitrate of the encoders in the Encoding Module.

3.5.1 Packet Scheduler

The Packet Scheduler module determines how to transmit video frames on heterogeneous links. This chapter has two main parts, which are **Video Packet Scheduling** and **Backup Frame Prioritization**. The **Video Packet Scheduling** explains how the Packet Scheduler module decides which links to probe and which links to send the primary frame and the backup frame. The **Backup Frame Prioritization** explains why the Packet Scheduler has to prioritize backup frames over primary frames when both frames are ready to be sent.

Video Packet Scheduling

When the Packet Scheduler receives a video frame from the Encoding Module, it splits the frame into multiple packets that are smaller than or equal to the MTU to avoid packet fragmentation at the link layer. Squash aggregates multiple links to send a primary frame at a higher video bitrate. To aggregate the available links to transmit the primary frames, the Packet Scheduler distributes the packets on the link modules based on the their estimated bandwidth. In contrast, the Packet Scheduler sends backup frames without aggregating the

Algorithm 2 Packet Scheduling

Require: n ▷ the number of available links
Require: b ▷ number of backup replications
Require: $x \leftarrow 2 * \frac{FrameInterval+RTTmin}{FrameInterval}$
Require: $links$ ▷ array of n available links
Require: ProbingLinks ← [], StableLinks ← [], BackupLinks ← [], $i \leftarrow 0$, $j \leftarrow 0$
  while $i + j < n$ **do** ▷ Classify probing links
    if $links[i + j].NumberOfAckedPacketsOnPreviousFrames(x) = 0$ **then** ▷ Check if any packet is acknowledged from the previous x frames
      $ProbingLinks[i] \leftarrow links[i + j]$
      $i \leftarrow i + 1$
    else
      $StableLinks[j] \leftarrow links[i + j]$
      $j \leftarrow j + 1$
    end if
  end while
  $k \leftarrow 0$
  while $k < b$ **do** ▷ Choose the link(s) to replicate a backup frame
    $l \leftarrow 0$
    $HighestStabilityScore \leftarrow 0$
    $MostStableLink \leftarrow 0$
    while $l < j$ **do**
      if $StableLinks[l].StabilityScore > HighestStabilityScore$ **then**
        $HighestStabilityScore \leftarrow StableLinks[l].StabilityScore$
        $MostStableLink \leftarrow StableLinks[l]$
      end if
      if $StableLinks[l].StabilityScore = HighestStabilityScore$ **then**
        if $StableLinks[l].SendingRate > MostStableLink.SendingRate$ **then**
          $MostStableLink \leftarrow StableLinks[l]$
        end if
      end if
      $l \leftarrow l + 1$
    end while
    $BackupLinks[k] \leftarrow MostStableLink$
    $k \leftarrow k + 1$
  end while
  Send_Backup_Frame_On(ProbingLinks + BackupLinks) ▷ Replication
  Send_Primary_Frame_On(StableLinks) ▷ Aggregation

links. If the frame is sent over aggregated links, the frame may suffer additional network delays in the event that one of the links becomes congested. The Packet Scheduler decides which link to send the backup frame based on the link's estimated bandwidth and stability score.

Algorithm 2 shows how the sender selects the candidates to probe or send a backup frame and primary frame. The link module updates the estimated bandwidth based on RTT and delivery rate from acknowledgement messages. When no recent acknowledgement messages arrive due to temporal link disconnection or significant congestion, the link module does not have sufficient samples to estimate the current network status. If there is no acknowledgement message for a frame during twice the time that the frame is expected to be acknowledged, the sender defines it as a link failure and starts probing by sending a copy of the backup frame on that link until it receives an acknowledgement message. The link module estimates the time that it takes to receive an acknowledgement for a frame based on the estimated bandwidth and RTTmin (Chapter 3.4). When the Packet Scheduler aggregates links to send a primary frame, it excludes failed links to reduce the risk of late frames.

The Packet Scheduler decides which links to send a backup frame by their stability scores. When more than one link has the same stability score, it compares the estimated bandwidth of the links. The Packet Scheduler sends packets on the link with higher network bandwidth. After deciding the links to send a backup frame, the Packet Scheduler sends a primary frame on the aggregated links.

Backup Frame Prioritization

The Packet Scheduler supports prioritized packet scheduling. When the Packet Scheduler sends video data, it places primary frames and backup frames on separate queues to distinguish their priorities. When the Packet Scheduler schedules frames to send, it pulls the packets from the backup frame queue before the primary frame queue to increase the probability of the backup frame's on-time arrival. This is because packets are generally delivered in order when they are sent on the same link. If the sender transmits a primary frame before a backup frame when the link experiences a significant bandwidth drop, the packets from the backup frame are blocked until the queued packets from the primary frame are drained. This causes both the primary and backup frames to be dropped due to a deadline violation. Alternatively, if the backup frame is sent before the primary frame on a link where its decreased bandwidth is larger than the backup frame's video bitrate, the backup frame can arrive on time when the link experiences a significant bandwidth drop.

However, prioritized packet scheduling is not helpful when the available bandwidth becomes lower than the backup encoder's video bitrate or when the link is already congested before the sender sends the backup frame. Squash can replicate a backup frame on multiple links to provide higher resilience to link congestion. The Encoding Bitrate Controller decides how many links it replicates the backup frame to and the video bitrate of the backup frame and the primary frame. As described in the algorithm 2, the Packet Scheduler probes unstable links by sending a backup frame instead of packets from the primary frame.

3.5.2 Encoding Bitrate Controller

The Encoding Bitrate Controller determines two main decisions: the number of links that the backup frame should replicate to and the video bitrate for each encoder in the Encoding Module. The Encoding Bitrate Controller makes these decisions based on the estimated bandwidth and stability score of each link. After making a decision, the Encoding Bitrate Controller notifies the Packet Scheduler of the number of links used for backup frame replication.

Algorithm 3 illustrates how the Encoding Bitrate Controller decides the video bitrate for the Backup Encoder and the Primary Encoder. The Encoding Bitrate Controller calculates the video bitrate for the Primary Encoder by subtracting the video bitrate for the Backup Encoder from the total available bandwidth. In order to determine the video bitrate of a backup frame, the Encoding Bitrate Controller has to decide on the number of links used for backup frame replication.

The Encoding Bitrate Controller dynamically changes the number of replications to *minReplicationNum* or *maxReplicationNum* depending on whether there is a stable link. When at least one of the available links has a stability score of 1.0, the Encoding Bitrate Controller sets the number of replications to *minReplicationNum*. The link having a stability score of 1.0 has a low probability of becoming congested because the link currently does not have packets in the network queue. On the other hand, when all links have a stability score below 1.0, the Encoding Bitrate Controller sets the number of replications to *maxReplicationNum* to increase the probability of delivering the backup frame before the deadline in the event that some of the links get congested.

When the application starts, Squash receives the parameters, *minReplicationNum* and *maxReplicationNum*, from the user. Considering the number of available network interfaces and application requirement, the user can choose these parameters. The user may choose to provide higher availability of a backup frame by increasing these parameters. However, the user should be cautious about increasing the number of replications since it may reduce

Algorithm 3 Encoding Bitrate Control

Require: n ▷ the number of available links
Require: b ▷ number of backup replications
Require: $availableBandwidth$ ▷ The total available bandwidth
Require: $maxBackupBitrate$ ▷ The maximum bitrate for a backup frame
Require: $minLinkSendingRate$ ▷ The minimum link's sending rate
Require: $links$ ▷ array of n available links
Require: $numberOfReplications$
Require: $backupFrameEncodingRate$
Require: $primaryFrameEncodingRate$
Require: $minReplicationNum$
Require: $maxReplicationNum$

```
i ← 0                                          ▷ Check if any link's stability score is 1.0
isAnyLinkStable ← FALSE
while i < n do                                 ▷ Classify probing links
    if links[i].stabiilityscore == 1 then
        IsAnyLinkStable ← TRUE
        break;
    end if
    i ← i + 1
end while
if IsAnyLinkStable = TRUE then                 ▷ Decide the number of replications
    numberOfReplications ← minReplicationNum
else
    numberOfReplications ← maxReplicationNum
end if
backupFrameEncodingRate ← MIN(minLinkSendingRate, maxBackupBitrate / numberOfReplications)
primaryFrameEncodingRate ← availableBandwidth − backupFrameEncodingRate
```

the video bitrate of both the backup frame and the primary frame by using large portions of the network capacity for replication.

After deciding the number of replications for a backup frame, the Encoding Bitrate Controller determines the size of the backup frame by getting the smaller of the *minLinkSendingRate* and *maxBackupBitrate*. The *minLinkSendingRate* is the minimum sending rate among the available links. As Squash streams video over heterogeneous links. A backup frame should be encoded at a video bitrate that is smaller than or equal to the *minLinkSendingRate* to be sent the backup frame over any stable link without link aggregation.

The *maxBackupBitrate* is the maximum video bitrate that the Encoding Bitrate Controller can assign to a backup frame. Squash takes the maximum bitrate percentage of the overall network bandwidth for a backup frame from the user as a parameter when the application starts. This limits bandwidth usage for the backup frame when the minimum link's sending rate is similar to the bandwidth of the other links. For example, when Squash uses two links and both links have the same bandwidth, the minimum link's bandwidth is half of the overall bandwidth. If the Encoding Bitrate Controller only considers the minimum link's bandwidth, the bandwidth usage for the backup frame becomes the same as the bandwidth usage for the primary frame. Sending a large backup frame reduces the primary frame's encoding rate and risks deadline violations when the link experiences a significant bandwidth drop. The user can increase the *maxBackupBitrate* to enhance a backup frame's quality when the user expects that backup frames are going to frequently replace primary frames. However, as the available bandwidth is limited, increasing the *maxBackupBitrate* reduces the video quality when backup frames are barely used. During the experiments using the emulated wireless LTE network environment (Chapter 4.1), I found that Squash provides the highest video quality when the *maxBackupBitrate* is 10% of the total available bandwidth.

The *maxBackupBitrate* is divided by the number of replications. As the backup frame is replicated on more links, the backup frame size should become smaller to prevent reducing the primary frame's image quality by using a large portion of the total available bandwidth for a backup frame. For example, when Squash replicates a backup frame on two links and the *maxBackupBitrate* is 10% of the overall bandwidth, the backup frame encoding rate should be less than 5% of the overall bandwidth.

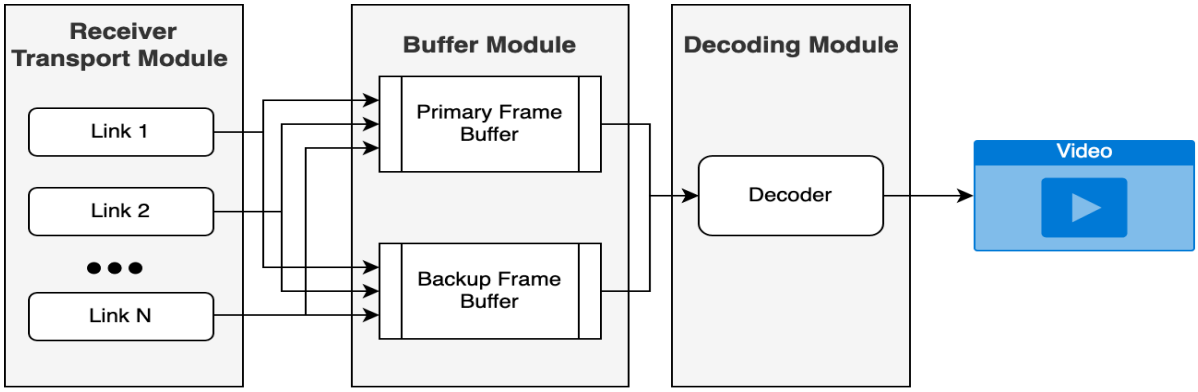

Figure 3.5: Squash Receiver. When a packet arrives, the receiver sends an acknowledgement containing the one-way delay and the measured delivery rate. The received packets are stored in a buffer until the entire frame is received. If the entirety of the primary frame does not arrive on time, Squash provides the backup frame to the decoder. If the primary and backup frames are both late, the frame is skipped.

3.6 Modules of Squash Receiver

Figure 3.5 illustrates the Squash receiver's components. The receiver has a simpler structure than the sender because most decisions about streaming video are made by the sender, and the receiver provides information for the sender to consider. The receiver consists of the **Receiver Transport Module**, the **Buffer Module**, and the **Decoding Module**. The Receiver Transport Module manages each link that receives video packets from the sender and passes the received packets to the Buffer Module. When a link receives a video packet, it updates the delivery rate on the link and sends an acknowledgement message for the received packet. The acknowledgement includes the delivery rate, which is used by the sender to estimate bandwidth.

The Buffer Module stores the video packets from the Receiver Transport Module until the complete frame is received before passing them to the Decoding Module. The Buffer Module consists of the Primary Frame Buffer and the Backup Frame Buffer. The Buffer Module stores frames in separate buffers because they are serviced under different conditions depending on whether the frame is ready before its deadline. The Buffer Module delivers the frames from the Primary Frame Buffer to the Decoding Module when the frames are received on time. If the primary frame does not arrive on time, the Buffer Module passes the frames from the backup frame buffer. When both frames do not arrive before the deadline, the Buffer Module signals the Decoding Module to skip the frame.

The Decoding Module decodes the frames received from the Buffer Module to display on the user's screen.

Chapter 4

Evaluation

The goal of this evaluation is to answer the following question: Does sending multi-bitrate frames improve the received video quality for low latency video streaming over wireless LTE networks? I answer this by conducting experiments that compare my approach against other reference systems using SSIM [70] as a primary metric. The experimental results show that sending multiple versions of a frame to avoid a frame skip enhances the quality of received video in the wireless LTE network environment. These experiments were conducted in an emulated network environment using Mininet [41], which I describe next.

4.1 Implementation & Experimental Setup

Squash consists of 1724 lines of C++ code. Squash uses the x264 library [3] to encode raw video in H.264 format. It pipes the raw video data into the application through stdin in Linux Ubuntu 20.04.3. The video source is FHD and 25 FPS. The original video used in the experiments was downloaded from the Blender Foundation [27]. I converted the original video from 60 FPS to 25 FPS. I selected 25 FPS instead of 60 FPS as 60 FPS is not supported to pipe raw video into the application with FFmpeg that regulates the speed of video. I ran the tests on a machine with the Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz (12 physical cores total).

I tested Squash and the reference systems on a virtual network that was created by using Mininet [41]. In my setup, there are two hosts: one for the sender and one for the receiver. Each host has multiple network interfaces that are connected to each other, emulating multiple available links. The propagation delay is set to 20 ms each way. The

minimum RTT is 40 ms. The network bandwidth is emulated through LTE bandwidth traces, which are collected from [72]. These traces are designed for Mahimahi [49], where Mahimahi [49] is able to emulate only a single link. I have converted these traces to Mininet traces [41], which can emulate multiple LTE links.

For all of the tests, I evaluate Squash and the reference systems by using different combinations of Mahimahi [49] traces. Mahimahi provides six LTE traces collected from major Internet service providers, including T-Mobile, AT&T, and Verizon. I exclude traces whose mean available bandwidth is less than 1 Mbps for upload because it seems outdated these days when the global average LTE upload speed is 12 Mbps [38]. I repeat each test, which comprises a single combination of traces, 20 times to obtain confidence intervals.

I measure the network delay by comparing the sender and receiver timestamps of each frame. I do not worry about clock synchronization during the experiments because both the sender and the receiver are on the same machine. After a frame is encoded into H.264, I assign the sender timestamp to the frame. When all the packets in the frame are received, the receiver assigns the frame's receive timestamp. The receiver subtracts the sender timestamp from the receiver timestamp to calculate the network delay. Squash receiver drops late frames whose network delay exceeds the acceptable delay. The receiver skips writing late frames to the output video file. I compare this file with the original video file to determine the SSIM [70] that Squash can achieve. SSIM is one of the most prominent metrics to evaluate perceived video quality [50, 55, 69], and many video streaming protocols [26, 40, 54, 76, 78] use SSIM to compare the video quality.

4.2 Quality Improvement from Backup Frame

In this experiment, I demonstrate the potential benefits of utilizing backup frames for low latency video streaming over multiple wireless LTE network links. Even though sending a backup frame has additional bandwidth overhead, the backup frame can mitigate the quality drop from a skipped frame that is the result of aggregated links experiencing a significant loss of available bandwidth. I explore whether or not it is better to use a small portion of the bandwidth for the backup frame as opposed to using that bandwidth to increase the main video stream's encoding bitrate.

I compare Squash against two baseline adaptive bitrate protocols. The first baseline protocol (baseline-1) is designed in the same manner as Squash except that it employs single-frame encoding. Both Squash and the baseline-1 uses the same bandwidth estimation and packet scheduling approach because I wish to compare the impact of using

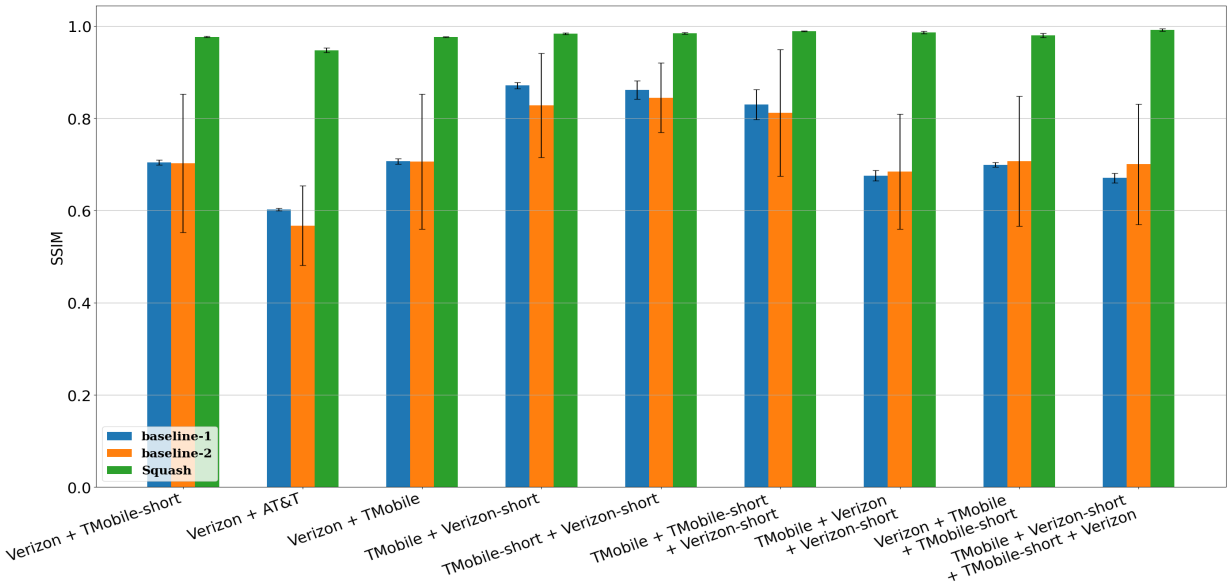

Figure 4.1: Each boxplot shows the SSIM of received video compared to the original video when the video is streamed using the baseline protocol and Squash on each combination of emulated LTE traces.

backup frames. The baseline-1 encodes the main video stream at a higher bitrate than Squash because it does not require additional bandwidth to send backup frames. However, the baseline-1 experiences skipped frames during the experiment due to deadline violations resulting from unexpected bandwidth drops on the aggregated links. This results in significant decreases in the received video's SSIM.

The second baseline protocol (baseline-2) also employs single-frame encoding that encodes at the same video bitrate as Squash's primary frame by tracking Squash's encoding rate logs. It shows the video quality if Squash only sends primary frames. The baseline-2 experiences fewer skipped frames from unexpected bandwidth drop than the baseline-1 on some traces. This is because the baseline-2 intentionally under-utilizes the links. It only uses the bandwidth that transmits the size of Squash's primary frame, while the baseline-1 uses all of the estimated bandwidth to send the main video stream.

Figure 4.1 displays the received video's SSIM on various combinations of LTE traces for each approach. The SSIM of the video streamed on Squash is 13 – 57% higher than that streamed on the baseline-1 for all combinations of LTE traces. In comparison to the baseline-2, the SSIM of the video transmitted over Squash is 13 – 50% higher. Squash outperforms the baseline protocols because the baseline protocols skip frames due to deadline

Figure 4.2: The graph shows the delivery rate and packet latency of Squash and the baseline protocol during the test using on Verizon-short LTE trace and AT&T LTE 2016 trace

violations when one of the links is congested. The difference in SSIM between Squash and the baseline protocols became larger in the test that uses the AT&T LTE 2016 trace since this trace frequently experiences significant bandwidth drops. The part of the frame sent over the AT&T link does not arrive before the deadline. This causes a frame skip due to a deadline violation.

During the tests, I found that the baseline protocols experienced more frame skips as the number of links that a frame is distributed across is increased. Aggregating more links to send a frame increases image quality; however, each time one of the links experiences a significant bandwidth drop, the frame is discarded due to a deadline violation. The experimental results indicate that using a backup frame can mitigate the distortion of the received video caused by skipped frames when there are unexpected drops in available bandwidth.

Figure 4.2 illustrates link utilization and packet latency when video is streamed using

Squash and the baseline-1 over multiple wireless LTE links. This test uses the Verizon-short LTE trace and AT&T LTE 2016 trace [49]. The spike in packet latency indicates network congestion. Both links experience temporary disconnection or significant bandwidth drops. Both protocols experience congestion. Congestion on a link may cause a skipped frame since the frame may be late as it is distributed over multiple links. However, Squash prevents a frame skip by transmitting a backup frame on the stable link when one link becomes congested. Sending backup frames mitigates the impact of skipped frames on video quality in highly variable network environments.

4.3 Optimized Link Replication

In this experiment, I test different Squash replication strategies for the backup frame to explore the impact of the different replication strategies on video quality. Squash provides the following three backup frame modes:

- **Single replication**: Single backup frame is generated and sent
- **Full replication**: A backup frame is replicated and sent on all links
- **Optimized replication**: Squash assesses link stability to dynamically determine the number copies of the backup frame that are sent.

Replicating the backup frame on more links increases resilience; however, additional replication requires more bandwidth. When the available bandwidth is limited, using more bandwidth for replication reduces the video encoding bitrate of the backup frame. This results in lower image quality. It is challenging to determine the minimum replication factor (i.e., the number of backup frame copies) that secures frame delivery.

Figure 4.3 shows the received video's SSIM using the different replication strategies. In most tests, the SSIM achieved using the single replication mode is lower than the SSIM achieved using the full replication mode and optimized replication mode. The SSIM of the video streamed using Optimized replication mode is 0.31 – 11.77% higher than the Single replication mode for all tests. Single replication mode provides lower video quality because it skips 2 – 5 frames out of 3000 frames on average due to deadline violations. If the link that transmits the backup frame experiences congestion from a sudden bandwidth drop, the backup frame arrives late. Squash fails to predict the link's stability about 0.07 – 0.16% of the time because of sudden bandwidth drops, and these mispredictions significantly

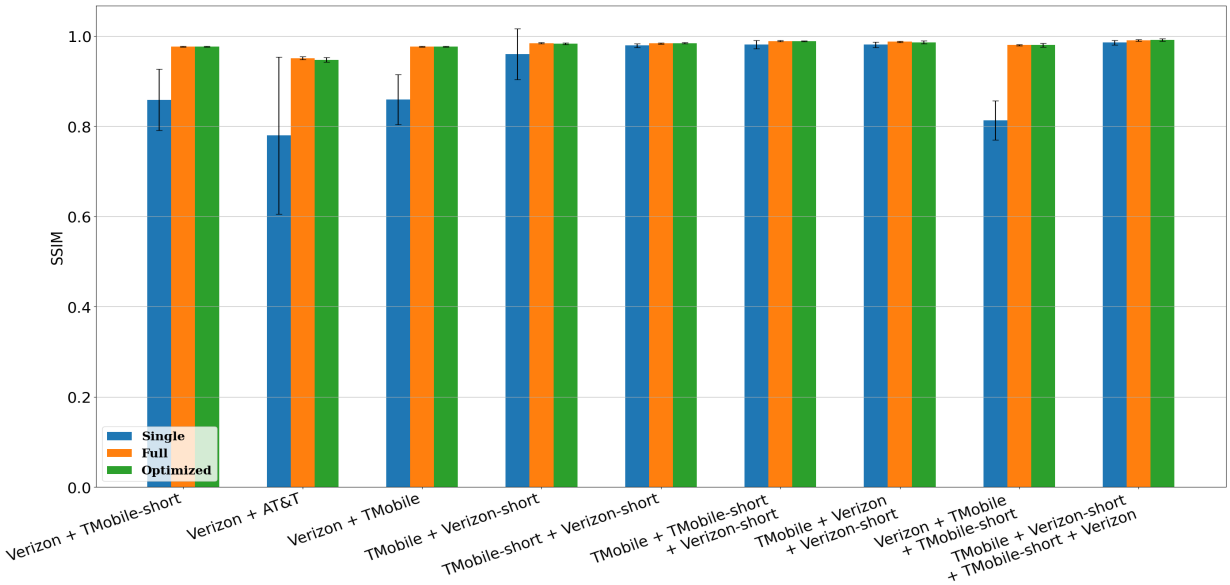

Figure 4.3: Comparing Squash's replication modes. Single replication mode provides less stable video streaming than the full and optimized replication modes when the links have variable bandwidth.

impact video quality. When the decoder skips a video frame, the video quality significantly decreases. Despite having a higher video bitrate when sending one backup frame, the video stream is more susceptible to delayed frames due to unexpected bandwidth drops. Alternatively, the full replication mode and the optimized replication mode require more bandwidth; however, the backup frames arrive on time, and there are no skipped frames in all tests. Replicating a backup frame on multiple links is necessary when network links have highly variable bandwidth.

The experiment showed that the difference between the full replication mode and the optimized replication mode is not noticeable with regards to SSIM, as their medians and confidence interval are similar for all of the tests. The optimized replication mode uses less bandwidth for replication than the full replication mode. However, using the optimized replication mode has a minor impact on the received video quality.

When Squash streams video over links with higher bandwidth variability, such as AT&T LTE trace and Verizon LTE trace, the resulting SSIM is lower. In these traces, Squash frequently experiences significant bandwidth drops that result in delivering backup frames instead of primary frames. I found that the received video's SSIM primarily depends on the number of backup frames that replace primary frames. Using backup frames results

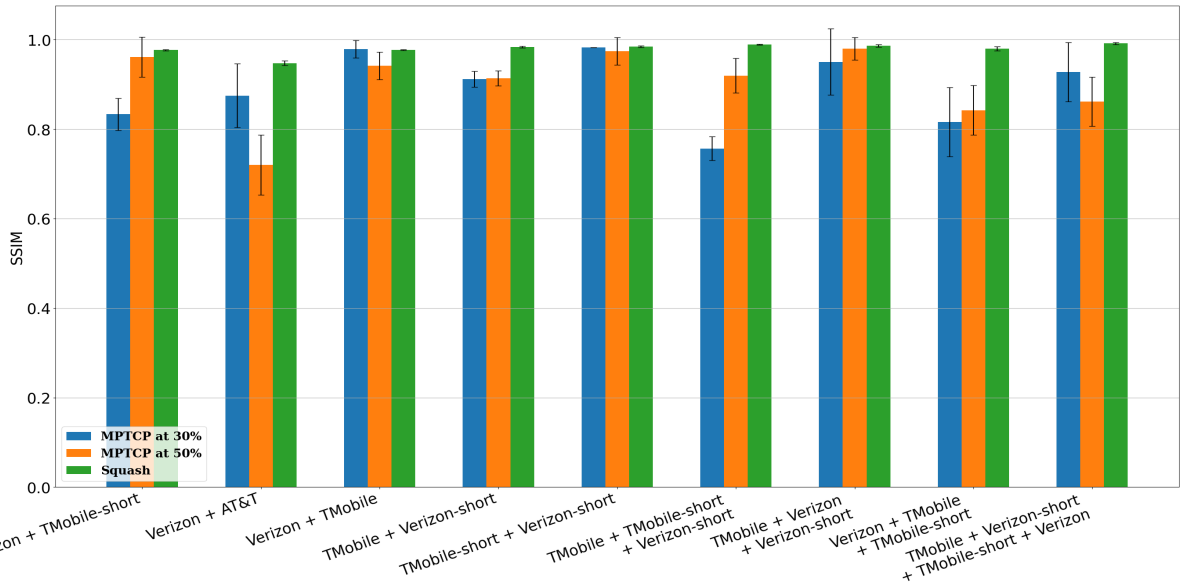

Figure 4.4: The graph shows the received video's SSIM compared to the original video when the video is streamed using MPTCP and Squash over each combination of emulated LTE traces.

in higher video quality than incurring a missing frame; however, the quality is lower than when using primary frames. When no frame is skipped, fewer number of frame replacement results in higher received video quality.

In addition to the number of frame replacements, the decoded video quality depends on the type of frame that is replaced by a backup frame. X264 video encoding uses three types of frames: keyframes (I-frame), predictive frames (P-frame), and bi-directional predicted frames (B-frame) [56]. I-frame is the full frame of the image in a video. P-frame and B-frame contains newly added information between the previous frame and the next frame. When an I-frame is replaced by a backup frame, the video quality experiences more distortion than when a P-frame or B-frame is replaced. Currently, Squash does not differentiate the frame type when it sends a frame. This should be considered for packet scheduling in future work.

4.4 Multiple Link Application

In this experiment, I compare Squash against MPTCP, which is one of the most widely used multi-path network transport protocols. The video streaming application running on MPTCP uses the same encoder settings as Squash, and both systems use the x264 library. They skip a frame when the frame's latency exceeds the acceptable delay. I provide knowledge of future network bandwidth on each link to the video streaming application that uses MPTCP. This is because MPTCP is a network transport protocol and does not inform the encoder of the desired video bitrate for the current link status. When encoding the next frame to be sent, the video streaming application knows all available links' bandwidth. The video streaming application encodes the video at a bitrate that is calculated by aggregating each link's bandwidth to provide the amount of video data that the network can handle.

Figure 4.4 shows the SSIM of the received video when the video is streamed over various combinations of LTE traces by using Squash and the MPTCP video streaming application that encodes the video at 30% and 50% of the total available bandwidth. Initially, I tested the video streaming application by setting the target encoding rate to be 90% of the total available bandwidth after accounting for the packet header and the variability of the encoded frame size. However, I found that the received video had a higher SSIM when the target encoding rate was lower than 50% of the total available bandwidth than when the target encoding bitrate is 90% of the total available bandwidth. While encoding the video at 90% of the available bandwidth provides high-quality video frames, it suffers from more frequent frame skips, resulting in a major decrease in the quality of the received video. I tested the MPTCP video streaming application with different target encoding rates (30 – 90%), and it provided the highest median SSIM at 30% and 50% target encoding rates.

Comparing the median SSIM in all tests, Squash's SSIM is 7 – 8% higher on average than video streaming on MPTCP, which encodes video at 30 – 50% of the total available bandwidth. I also compare network utilization, which is measured by ethstats. The MPTCP video streaming application uses only 25 – 43% of the total available bandwidth to avoid a frame skip due to deadline violation while Squash utilizes 66% of the available bandwidth. MPTCP's default scheduler sends data on the link with the lowest RTT until its congestion window becomes full. This approach does not consider each link's capacity when data is transmitted over heterogeneous links [23, 45]. A low bandwidth link with low RTT may experience congestion due to over-sending while other links with high bandwidth and a high RTT are under-utilized.

Another reason that Squash outperforms MPTCP is that the default MPTCP scheduler uses ARQ for lost packets to provide reliable transport. ARQ requires additional delay to

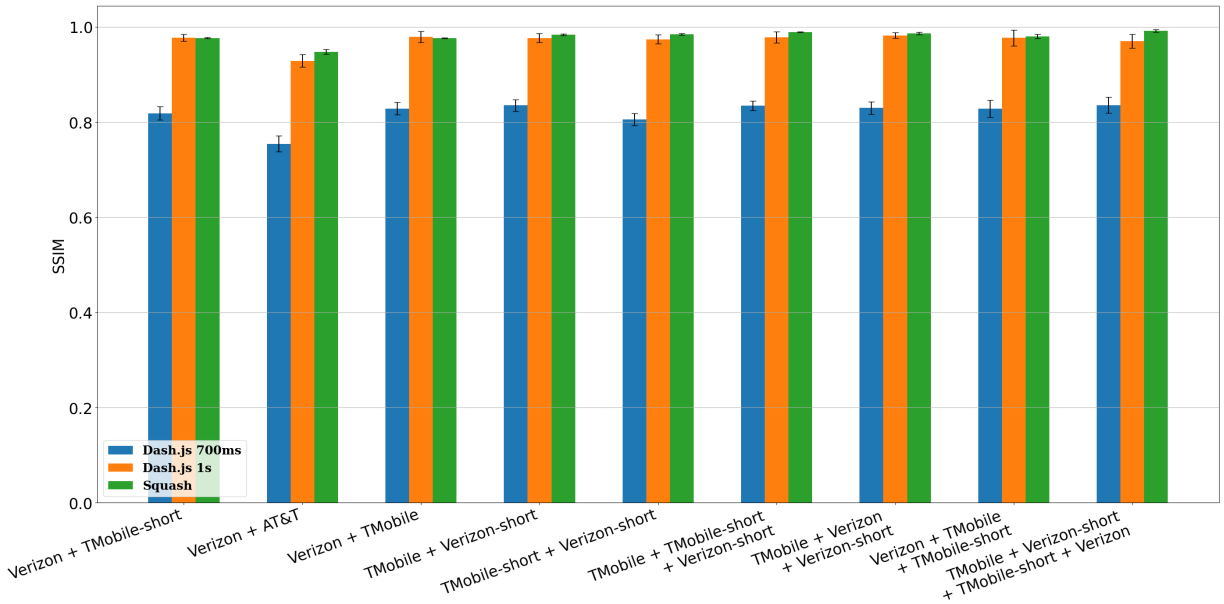

Figure 4.5: Comparing the received video's SSIM when the video is streamed on Squash against on Dash.js, which encode at different frame deadline.

detect packet loss and to resend the packet. It causes the head-of-line blocking problem where the received part of a frame has to wait in the receiver buffer until the resent packets arrive [24, 34]. Each frame has a short deadline in a low latency video streaming environment. The receiver has to discard the frames that arrive late because of the retransmission delay.

4.5 Single Link Approach

In this experiment, I compare Squash against Dash.js [15] to evaluate how well Squash is able to overcome the challenges of multihoming. Both systems have the same amount of total available bandwidth at their disposal. In the case of Dash.js, I aggregate the available bandwidth of the LTE traces into a single trace representing a single link. Dash.js uses this trace for its experiments; therefore, it has the same amount of available bandwidth as Squash and does not have to contend with the challenges of utilizing multiple heterogeneous links. Dash.js is a widely used video streaming application. Dash.js supports a low latency mode that allows the user to set a target latency, which is defined as the acceptable

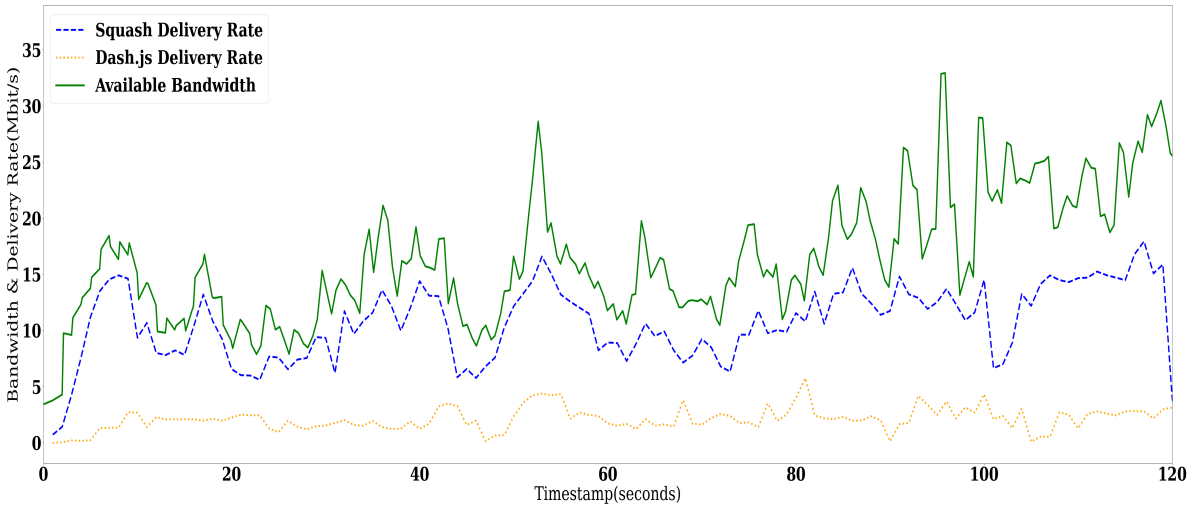

Figure 4.6: The graph shows the delivery rate and available bandwidth when Squash and Dash.js streams video over Verizon LTE trace and T-Mobile LTE trace.

amount of time between video encoding and display. I use DASH-IF DASH Live Source Simulator [14] to simulate live streaming with pre-encoded video. The original video is encoded at 6 different video encoding rates (0.1 Mbps, 0.5 Mbps, 1 Mbps, 2 Mbps, 4 Mbps, 8 Mbps, 16 Mbps) to provide ABR streaming. Each video chunk length is set to 100ms, which is the minimum chunk length in the DASH-IF DASH Live Source Simulator.

Figure 4.5 illustrates the measured SSIM of the received video using Squash and Dash.js over various combinations of LTE traces. I ran the tests for Dash.js with different frame deadlines while the frame deadline for Squash is fixed to 500ms. This is because Dash.js sends at the minimum encoding rate when I set the frame deadline to 500 ms target latency. Squash outperformed Dash.js by 18 – 25% in SSIM when Dash.js sets the acceptable delay to 700ms. I have to set the frame deadline for Dash.js to be at least 1 second to provide comparable video quality with Squash. This is because of its longer chunk length than Squash. The minimum chunk length in the DASH-IF DASH Live Source Simulator is 100 ms while Squash encodes a frame every 40 ms. The video chunk is larger and takes longer to deliver to the receiver.

Another reason that Squash outperforms Dash.js in the tests is Dash.js' conservative ABR streaming. Figure 4.6 displays the available bandwidth and how much Dash.js uses on the link when the acceptable delay is 1 second. The link provides the same bandwidth as 2 links. Dash.js uses only 6% of the available bandwidth while Squash uses 66%. Squash

can aggressively increase the video bitrate with a shorter acceptable delay (500ms) because it utilizes a backup frame.

Chapter 5

Conclusion

5.1 Contribution

Squash is a multi-homed, low latency video transport protocol that uses multi-bitrate encoding. To the best of my knowledge, Squash is the first approach to increase resilience on multiple links by utilizing copies of frames that are encoded at multiple bitrates. Squash increases resilience against link congestion by using only a small portion of the networks' bandwidth compared to FEC. From our experimental evaluation, the average size of the backup frames is less than 8.3% of the Squash's total bandwidth usage. Using multiple encodings can efficiently mitigate the impact of sudden, unexpected bandwidth drops.

The evaluation (Chapter 4) demonstrates the advantages of sending a backup frame on the links having variable bandwidth. Firstly, I compare Squash with a baseline protocol that uses single encoded frames to show that utilizing backup frame mitigates video quality drop from frame skip caused by deadline violation. Secondly, I explore different replication strategies to determine the most efficient way to decide the video encoding rate and replication factor. Thirdly, I compare Squash with the implemented video streaming application that uses MPTCP, which is one of the most widely used multi-path protocols. The test results show that the SSIM attained by Squash is 7 – 8% higher on average than when using MPTCP. Finally, I also compare Squash to Dash.js in a single-homed environment where it does not have to deal with the challenges of using multiple links. Dash.js requires at least 1 second of acceptable delay to provide comparable video quality with Squash when Squash's acceptable delay is 500 ms.

5.2 Limitations and Future Work

The design and evaluation of Squash reveal several future research directions, which I describe in the following paragraphs.

5.2.1 Deciding the size of backup frame

When Squash decides the video encoding rate, it does not consider the impact of changing the video encoding rate on the received video's image quality. This is because Squash does not know the quantified quality of the encoded video frame in real time. As shown in Appendix A, the SSIM of a video frame does not increase linearly as the encoding rate is increased. The impact of increasing the encoding rate is different for each video. More research on the relationship between video encoding rate and encoded video quality. If Squash can determine the expected quality of the encoded frame on the specific encoding rate, Squash can improve the decision making model to determine the encoding rate for the backup frame and the primary frame.

5.2.2 Limitations in Evaluation

LTE trace conversion I convert the Mahimahi [49] LTE traces into Mininet LTE traces because Mahimahi does not support multi-link network emulation. When I convert, I need to change the frequency that the network bandwidth is changed from millisecond to second. This is because I observed that the changed bandwidth is not applied correctly on Mininet when the network bandwidth is changed every millisecond. In the future, extending the Mahimahi project to support multi-link network emulation will make the experiment environment closer to the collected LTE data.

Outdated LTE trace The wireless LTE traces from [72] are collected in 2013. There have been many works improving the quality of wireless LTE network since it first came out. These traces may not represent current network environment, but this is one of the most practical LTE network traces among publicly available LTE network traces. It reflects the highly variable bandwidth of wireless LTE network, and there is no significant difference from the more recently collected network traces [5] in link capacity.

User Fairness The evaluation does not contain the tests for user fairness on the network resource because the primary focus of Squash is to show the effect of using a backup frame. Future work can include experiments running multiple Squash applications that share the same network links. It will make this work more practical.

Comparison with a single link application The experiment from Chapter 4.5 compares Squash with Dash.js because it allows us to set the target delay in low latency mode while other commonly used applications do not. However, according to their document [60], the recommended target delay is from 1.5 to 2 seconds, while our goal is under 500ms. WebRTC supports lower latency requirements than Dash.js, but I could not compare WebRTC with Squash because it does not allow setting the latency requirement to skip late frames. Adding more competitors that support lower latency requirements than Dash.js will make the experiment comparing Squash with a single link more reasonable.

References

[1] Mininet. http://mininet.org. Accessed: 2022-08-09.

[2] Webrtc. https://webrtc.org/. Accessed: 2022-05-28.

[3] x264 repository. https://code.videolan.org/videolan/x264. Accessed: 2022-11-28.

[4] x265 repository. https://bitbucket.org/multicoreware/x265_git/wiki/Home. Accessed: 2022-05-28.

[5] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 632–647, 2020.

[6] Venkat Arun and Hari Balakrishnan. Copa: Practical {Delay-Based} congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 329–342, 2018.

[7] Inc. Bitmovin. Bitmovin's 5th annual video developer report 2021.

[8] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *ACM Queue*, 14, September-October:20 – 53, 2016.

[9] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.

[10] Yung-Chih Chen, Yeon-sup Lim, Richard J Gibbens, Erich M Nahum, Ramin Khalili, and Don Towsley. A measurement-based study of multipath tcp performance over wireless networks. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 455–468, 2013.

[11] Federico Chiariotti, Stepan Kucera, Andrea Zanella, and Holger Claussen. Analysis and design of a latency control protocol for multi-path data delivery with pre-defined qos guarantees. *IEEE/ACM Transactions on Networking*, 27(3):1165–1178, 2019.

[12] Cisco. Cisco annual internet report (2018–2023) white paper, 2020.

[13] Xavier Corbillon, Ramon Aparicio-Pardo, Nicolas Kuhn, Géraldine Texier, and Gwendal Simon. Cross-layer scheduler for video streaming over mptcp. In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.

[14] Dash-Industry-Forum. dash-industry-forum/dash-live-source-simulator.

[15] Dash-Industry-Forum. Dash.js repository.

[16] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. {PCC} vivace: Online-learning congestion control. In *15th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, 2018.

[17] Anis Elgabli and Vaneet Aggarwal. Smartstreamer: Preference-aware multipath video streaming over mptcp. *IEEE Transactions on Vehicular Technology*, 68(7):6975–6984, 2019.

[18] Anis Elgabli and Vaneet Aggarwal. Smartstreamer: Preference-aware multipath video streaming over mptcp. *IEEE Transactions on Vehicular Technology*, 68(7):6975–6984, 2019.

[19] Anis Elgabli, Ke Liu, and Vaneet Aggarwal. Optimized preference-aware multi-path video streaming with scalable video coding. *IEEE Transactions on Mobile Computing*, 19(1):159–172, 2018.

[20] Kristian Evensen, Dominik Kaspar, Carsten Griwodz, Pål Halvorsen, Audun Hansen, and Paal Engelstad. Improving the performance of quality-adaptive video streaming over multiple heterogeneous access networks. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 57–68, 2011.

[21] G. Fairhurst and L. Wood. RFC3366: Advice to link designers on link automatic repeat request (ARQ). Technical report, USA, 2002.

[22] Christian Feller, Juergen Wuenschmann, Thorsten Roll, and Albrecht Rothermel. The vp8 video codec-overview and comparison to h. 264/avc. In *2011 IEEE International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*, pages 57–61. IEEE, 2011.

[23] Simone Ferlin, Özgü Alay, Olivier Mehani, and Roksana Boreli. Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks. In *2016 IFIP networking conference (IFIP networking) and workshops*, pages 431–439. IEEE, 2016.

[24] Simone Ferlin, Stepan Kucera, Holger Claussen, and Özgü Alay. Mptcp meets fec: Supporting latency-sensitive applications over heterogeneous networks. *IEEE/ACM Transactions on Networking*, 26(5):2005–2018, 2018.

[25] Alan Ford, Costin Raiciu, Mark J. Handley, Olivier Bonaventure, and Christoph Paasch. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 8684, March 2020.

[26] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *15th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 267–282, 2018.

[27] Blender Foundation. Big buck bunny, sunflower, 2008.

[28] Bo Han, Feng Qian, Lusheng Ji, and Vijay Gopalakrishnan. Mp-dash: Adaptive video streaming over preference-aware multipath. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 129–143, 2016.

[29] Stefan Holmer, Henrik Lundin, Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. A Google Congestion Control Algorithm for Real-Time Communication. Internet-Draft draft-alvestrand-rmcat-congestion-03, Internet Engineering Task Force, June 2015. Work in Progress.

[30] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.

[31] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. An in-depth study of lte: effect of network

protocol and application behavior on performance. *ACM SIGCOMM Computer Communication Review*, 43(4):363–374, 2013.

[32] Junxian Huang, Qiang Xu, Birjodh Tiwana, Z Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 165–178, 2010.

[33] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[34] Per Hurtig, Karl-Johan Grinnemo, Anna Brunstrom, Simone Ferlin, Özgü Alay, and Nicolas Kuhn. Low-latency scheduling in mptcp. *IEEE/ACM Transactions on Networking*, 27(1):302–315, 2018.

[35] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. *ACM SIGCOMM Computer Communication Review*, 32(4):295–308, 2002.

[36] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. Tackling bufferbloat in 3g/4g networks. In *Proceedings of the 2012 Internet Measurement Conference*, pages 329–342, 2012.

[37] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 97–108, 2012.

[38] Dan Jones and Kevin Beaver. Lte (long-term evolution), Apr 2021.

[39] Srinivasan Keshav. A control-theoretic approach to flow control. In *Proceedings of the conference on Communications architecture & protocols*, pages 3–15, 1991.

[40] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 107–125, 2020.

[41] Bob Lantz, Brandon Heller, and Nick 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*, pages 1–6, 2010.

[42] Ming Li, Andrey Lukyanenko, Sasu Tarkoma, Yong Cui, and Antti Ylä-Jääski. Tolerating path heterogeneity in multipath tcp with bounded receive buffers. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, page 375–376, New York, NY, USA, 2013. Association for Computing Machinery.

[43] Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. Tack: Improving wireless transport performance by taming acknowledgments. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 15–30, 2020.

[44] May Lim, Mehmet N Akcay, Abdelhak Bentaleb, Ali C Begen, and Roger Zimmermann. When they go high, we go low: low-latency live streaming in dash. js with lol. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 321–326, 2020.

[45] Yeon-sup Lim, Erich M Nahum, Don Towsley, and Richard J Gibbens. Ecf: An mptcp path scheduler to manage heterogeneous paths. In *Proceedings of the 13th international conference on emerging networking experiments and technologies*, pages 147–159, 2017.

[46] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210, 2017.

[47] Loren Merritt and Rahul Vanam. x264: A high performance h. 264/avc encoder. *online] http://neuron2. net/library/avc/overview_x264_v8_5. pdf*, 2006.

[48] Abdelhamid Nafaa, Tarik Taleb, and Liam Murphy. Forward error correction strategies for media streaming over wireless networks. *IEEE Communications Magazine*, 46(1):72–79, 2008.

[49] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *Usenix annual technical conference*, pages 417–429, 2015.

[50] Tao-Sheng Ou, Yi-Hsin Huang, and Homer H Chen. Ssim-based perceptual rate control for video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(5):682–691, 2011.

[51] C. Paasch, S. Barre, and et al. Multipath tcp in the linux kernel. available from https://www.multipath-tcp.org.

[52] Christoph Paasch, Gregory Detal, Fabien Duchene, Costin Raiciu, and Olivier Bonaventure. Exploring mobile/wifi handover with multipath tcp. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*, pages 31–36, 2012.

[53] Shiva Raj Pokhrel, Manoj Panda, and Hai L. Vu. Analytical modeling of multipath tcp over last-mile wireless. *IEEE/ACM Transactions on Networking*, 25(3):1876–1891, 2017.

[54] Devdeep Ray, Jack Kosaian, KV Rashmi, and Srinivasan Seshan. Vantage: optimizing video upload for time-shifted viewing of social live streams. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 380–393. 2019.

[55] Abdul Rehman and Zhou Wang. Ssim-inspired perceptual video coding for hevc. In *2012 IEEE International Conference on Multimedia and Expo*, pages 497–502. IEEE, 2012.

[56] Anthony Romero. Keyframes, interframe & video compression. https://blog.video.ibm.com/streaming-video-tips/keyframes-interframe-video-compression, Apr 2021. Accessed: 2022-11-28.

[57] Yusuf Sani, Andreas Mauthe, and Christopher Edwards. Adaptive bitrate selection: A survey. *IEEE Communications Surveys & Tutorials*, 19(4):2985–3014, 2017.

[58] Patrick Seeling, Frank H. P. Fitzek, Gergö Ertli, Akshay Pulipaka, and Martin Reisslein. Video network traffic and quality comparison of vp8 and h.264 svc. In *Proceedings of the 3rd Workshop on Mobile Video Delivery*, MoViD '10, page 33–38, New York, NY, USA, 2010. Association for Computing Machinery.

[59] Yousef O. Sharrab and Nabil J. Sarhan. Detailed comparative analysis of vp8 and h.264. In *2012 IEEE International Symposium on Multimedia*, pages 133–140, 2012.

[60] Daniel Silhavy. Low latency streaming · dash-industry-forum/dash.js wiki.

[61] Varun Singh, Saba Ahsan, and Jörg Ott. Mprtp: multipath considerations for real-time media. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 190–201, 2013.

[62] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking*, 28(4):1698–1711, 2020.

[63] Randall Stewart and Christopher Metz. Sctp: new transport protocol for tcp/ip. *IEEE Internet Computing*, 5(6):64–69, 2001.

[64] Thomas Stockhammer. Dynamic adaptive streaming over http– standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144, 2011.

[65] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[66] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 272–285, 2016.

[67] Ian Swett. Quic FEC V1. https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit. Accessed: 2022-08-10.

[68] Tobias Viernickel, Alexander Froemmgen, Amr Rizk, Boris Koldehofe, and Ralf Steinmetz. Multipath QUIC: A deployable multipath transport protocol. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[69] Shiqi Wang, Abdul Rehman, Zhou Wang, Siwei Ma, and Wen Gao. Ssim-motivated rate-distortion optimization for video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(4):516–529, 2011.

[70] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[71] Paul Wilkins, Yaowu Xu, Lou Quillio, James Bankoski, Janne Salonen, and John Koleszar. VP8 Data Format and Decoding Guide. RFC 6386, November 2011.

[72] Keith Winstein, Anirudh Sivaraman, Hari Balakrishnan, et al. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI*, volume 1, pages 2–3, 2013.

[73] Jiyan Wu, Chau Yuen, Bo Cheng, Ming Wang, and Junliang Chen. Streaming high-quality mobile video with multipath TCP in heterogeneous wireless networks. *IEEE Transactions on Mobile Computing*, 15(9):2345–2361, 2015.

[74] Yitao Xing, Kaiping Xue, Yuan Zhang, Jiangping Han, Jian Li, Jianqing Liu, and Ruidong Li. A low-latency mptcp scheduler for live video streaming in mobile networks. *IEEE Transactions on Wireless Communications*, 20(11):7230–7242, 2021.

[75] Yang Xu, Chenguang Yu, Jingjiang Li, and Yong Liu. Video telephony for end-consumers: Measurement study of google+, ichat, and skype. In *Proceedings of the 2012 Internet Measurement Conference*, pages 371–384, 2012.

[76] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Alexander Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *NSDI*, volume 20, pages 495–511, 2020.

[77] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.

[78] Pinghua Zhao, Yanwei Liu, Jinxia Liu, Ruixiao Yao, and Song Ci. Ssim-based cross-layer optimized video streaming over lte downlink. In *2014 IEEE Global Communications Conference*, pages 1394–1399. IEEE, 2014.

[79] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. Learning to coordinate video codec with transport protocol for mobile video telephony. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[80] Xuan Kelvin Zou, Jeffrey Erman, Vijay Gopalakrishnan, Emir Halepovic, Rittwik Jana, Xin Jin, Jennifer Rexford, and Rakesh K Sinha. Can accurate predictions improve video streaming in cellular networks? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 57–62, 2015.

APPENDICES

Appendix A

SSIM / PSNR comparison on different video bitrate

A.1 Big Buck Bunny and recorded geese video

Bitrate(kBit/s)	SSIM(BBB)	PSNR(BBB)	SSIM(geese)	PSNR(geese)
100	0.719988	18.170111	0.494211	20.432911
200	0.748171	21.092699	0.550298	23.676379
300	0.795023	23.275786	0.620311	24.794877
400	0.828264	24.673462	0.689590	25.815639
500	0.861270	26.862021	0.744523	26.788844
600	0.882503	28.578364	0.785305	27.651534
700	0.898651	30.079848	0.814827	28.384299
800	0.910010	31.068385	0.836524	28.991489
900	0.918962	31.889986	0.853365	29.515943
1000	0.926141	32.602574	0.866605	29.970969
1500	0.949240	35.110441	0.905434	31.588178
2000	0.961390	36.749313	0.924596	32.648890
2500	0.969020	37.993513	0.936357	33.438221
3000	0.974212	39.018333	0.944564	34.086373
4000	0.980860	40.709562	0.955508	35.130700
5000	0.984910	42.037322	0.962757	35.981292
8000	0.990874	44.803367	0.975341	37.951469
10000	0.992836	46.157395	0.979946	38.966658
15000	0.995334	48.541878	0.986199	40.889639
20000	0.996557	50.206696	0.989393	42.327756
25000	0.997314	51.533888	0.991356	43.512647
30000	0.997826	52.652603	0.992678	44.508586
35000	0.998195	53.565877	0.993640	45.380160
40000	0.998477	54.385593	0.994365	46.152090
45000	0.998693	55.064173	0.994942	46.855329
50000	0.998877	55.847535	0.995413	47.499676
60000	0.999148	57.149799	0.996175	48.683342
75000	0.999415	58.655838	0.997020	50.242375
90000	0.999567	60.121177	0.997667	51.651845
100000	0.999625	60.853671	0.998033	52.545534
120000	0.999699	61.977986	0.998685	54.452298
160000	0.999755	62.955999	0.999607	59.722215
200000	0.999763	63.283097	0.999703	60.839319

Table A.1: This table compares the SSIM and PSNR of two FHD videos when they are encoded at different video bitrates. The Big Buck Bunny is a short computer-animated comedy film. The geese video is a recording of geese using an iPhone 8.