Analyzing DASH Video Streaming over Geostationary Satellite Networks

A Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science in

Computer Science

By:

Daniel Quackenbush
Kristi Prifti
Zijian Guan

Project Advisor:

Mark Claypool

Sponsored By:

Viasat

Date: April 2023

Abstract

Dynamic Adaptive Streaming over HTTP (DASH) is a popular, modern method for streaming media. Adaptive bitrate (ABR) functionality allows DASH to adjust to changing network conditions during the course of the stream. However, little work has assessed DASH's performance over geostationary satellite networks, where high latencies cause many network protocols to perform poorly. This paper presents results from experiments that evaluate DASH over a commercial geostationary satellite connection, comparing performance when adjusting two different configuration settings of the stream: The length of video segments, and the ABR algorithm used to make decisions. Results show that: 1) longer segment lengths are more stable but fail to adjust bitrate, staying at a low video quality; and 2) the buffer-based BOLA ABR algorithm has fewer stalls and better video quality than either a throughput-based approach or a hybrid approach.

Contents

1 Introduction **1**

2 Background **3**

2.1 Geostationary Satellite Networks **3**

2.2 Transport Protocols **4**

2.2.1 Transmission Control Protocol (TCP) 4

2.2.1.1 TCP Handshake 4

2.2.1.2 TCP and Congestion Control 4

2.3 HTTP **5**

2.3.1 HTTP Versions 5

2.4 Dynamic Adaptive Streaming over HTTP **6**

2.4.1 ABR Algorithms 6

3 Related Work **8**

3.1 Satellite Networks **8**

3.1.1 TCP Congestion Control 8

3.2 DASH **9**

3.2.1 Segment Lengths 9

3.2.2 ABR Algorithms 10

3.2.3 Quality of Experience (QoE) Metrics 11

4 Methodology **13**

4.1 Test Bench Setup **13**

4.1.1 Components 14

4.1.1.1 Glomma 14

4.1.1.2 MLCNet Servers 14

4.1.2 Network Connection 14

4.1.2.1 Local Area Network (LAN) Connection 15

4.1.2.2 Geostationary Satellite Connection 15

4.1.3 Test script 15

4.2 Test Video 16
4.3 Pilot Tests 18
4.3.1 Format 18
4.3.2 Results 19
4.3.3 Discussion 20
4.3.4 Conclusion 20
4.4 Experimental Setup 20
4.4.1 Test Procedure 20
4.4.2 Variables 21
4.4.2.1 Independent Variables 21
4.4.2.2 Dependent Variables 22

5 Analysis **24**
5.1 Segment Length Test 24
5.1.1 Time to Start Playing 24
5.1.2 Stall Counts 25
5.1.3 Stall Lengths 25
5.1.4 Total Amounts of Stall Time 25
5.1.5 Quality Level Changes 26
5.1.6 Network Throughputs 26
5.2 ABR Algorithm Test 29
5.2.1 Time to Start Playing 29
5.2.2 Stall Counts 29
5.2.3 Stall Lengths 30
5.2.4 Total Amounts of Stall Time 30
5.2.5 Quality Level Changes 31
5.2.6 Network Throughputs 31

6 Conclusion **35**

7 Future Work **37**
7.1 Limitations 37
7.2 Opportunities 37

Appendices **39**

A Appendix A - Segment Length Results **40**

B Appendix B - ABR Algorithm Results **46**
References 56

List of Tables

4.1 Settings for buffer-related configuration of Dash.js player in test script. 15

4.2 Descriptions of the data collected by the test script and the purposes of each point. 17

4.3 Quality levels the video was prepared in, and the specifications of each. 17

4.4 Quality levels of Streamroot video used in pilot tests. 19

4.5 Independent variables 22

4.6 Dependent variables 23

5.1 **Segment length test, time to start playing.** Time elapsed between starting the stream and the video beginning to play, due to downloading the first segment. 24

5.2 **Segment length test, stall counts.** Total number of individual stalls observed over the course of the test, regardless of the length of the stall. 25

5.3 **Segment length test, stall lengths.** Average lengths of stalls observed over the course of the test. 25

5.4 **Segment length test, total amounts of stall time.** Total amount of time spent in a stall over the course of the test, the product of the number of stalls and the average length of those stalls. 25

5.5 **Segment length test, number of quality level changes.** Number of quality level changes observed over the course of the test, regardless of whether it is an upswitch or downswitch. . 26

5.6 **Segment length test, estimated throughputs.** Throughput metrics, as reported by the Dash.js player's throughput estimator. Averaged over all runs of the setting. 27

5.7 **Segment length test, measured throughputs.** Throughput metrics, as measured by the Wireshark trace on the client. Averaged over all runs of the setting. 27

5.8 **ABR algorithm test, time to start playing.** Time elapsed between starting the stream and the video beginning to play, due to downloading the first segment. 29

5.9 **ABR algorithm test, stall counts.** Total number of individual stalls observed over the course of the test, regardless of the length of the stall. 29

5.10 **ABR algorithm test, stall lengths.** Average lengths of stalls observed over the course of the test. 30

5.11 **ABR algorithm test, total amounts of stall time.** Total amount of time spent in a stall over the course of the test, the product of the number of stalls and the average length of those stalls. 31

5.12 **ABR algorithm test, number of quality level changes.** Number of quality level changes observed over the course of the test, regardless of whether it is an upswitch or downswitch. . 31

5.13 **ABR algorithm test, estimated throughput.** Average and standard deviation of throughput, as reported by the Dash.js player's throughput estimator. Averaged over all runs of the setting. 33

5.14 **ABR algorithm test, measured throughput.** Average and standard deviation of throughput, as measured by the Wireshark trace on the client. Averaged over all runs of the setting. 33

List of Figures

4.1 Experimental Setup 13

5.1 **Segment length test, stall length box plots.** Distribution of stall lengths over the course of the test. 26

5.2 **Segment length test, estimated throughput vs time.** Throughput over time, as reported by the Dash.js player's throughput estimator. 5 second segment length. 28

5.3 **Segment length test, measured throughput vs time.** Throughput over time, as measured by the Wireshark trace on the client. 5 second segment length. 28

5.4 **Segment length test, estimated and measured throughput vs time.** Comparing estimated throughput to observed throughput over time. 5 second segment length, first run. . 28

5.5 **ABR algorithm test, stall length box plots.** Distribution of stall lengths over the course of the test. 30

5.6 **ABR algorithm test, quality level distribution.** Distribution of quality levels over the course of the test. 32

5.7 **ABR algorithm test, quality level change distribution.** Distribution of quality level changes per minute over the course of the test. 32

5.8 **ABR algorithm test, estimated throughput vs time.** Throughput over time, as reported by the Dash.js player's throughput estimator. "BOLA" ABR algorithm. 33

5.9 **ABR algorithm test, measured throughput vs time.** Throughput over time, as measured by the Wireshark trace on the client. "BOLA" ABR algorithm. 34

5.10 **ABR algorithm test, estimated and measured throughput vs time.** Comparing estimated throughput to observed throughput over time. "BOLA" ABR algorithm, first run. 34

A.1 **Segment length test, distributions of quality levels.** Distribution of quality levels over the course of the test. 41

A.2 **Segment length test, box plots of quality level changes per minute.** Distribution of quality level changes per minute over the course of the test. 41

A.3 **Segment length test, estimated throughput vs time.** Throughput over time, as reported by the Dash.js player's throughput estimator. 10 second segment length. 42

A.4 **Segment length test, estimated throughput vs time.** Throughput over time, as reported by the Dash.js player's throughput estimator. 20 second segment length. 42

A.5 **Segment length test, measured throughput vs time.** Throughput over time, as measured by the Wireshark trace on the client. 10 second segment length. 42

A.6 **Segment length test, measured throughput vs time.** Throughput over time, as measured by the Wireshark trace on the client. 20 second segment length. 43

A.7 **Segment length test, estimated and measured throughput vs time.** Comparing estimated throughput to observed throughput over time. 10 second segment length, first run. 43

A.8 **Segment length test, estimated and measured throughput vs time.** Comparing estimated throughput to observed throughput over time. 20 second segment length, first run. 43

A.9 **Out of order and missing packets. Packet status vs time.** 5 second segment length. . 44

A.10 **Out of order and missing packets. Packet status vs time.** 10 second segment length. 44

A.11 **Out of order and missing packets. Packet status vs time.** 20 second segment length. 45

B.1 Throughput over time, as reported by the Dash.js player's throughput estimator. "Dynamic" ABR algorithm. 47

B.2 Throughput over time, as reported by the Dash.js player's throughput estimator. "Throughput" ABR algorithm. 47

B.3 Throughput over time, as measured by the Wireshark trace on the client. "Dynamic" ABR algorithm. 48

B.4 Throughput over time, as measured by the Wireshark trace on the client. "Throughput" ABR algorithm. 48

B.5 Comparing estimated throughput to observed throughput over time. "Dynamic" ABR algorithm, first run. 49

B.6 Comparing estimated throughput to observed throughput over time. "Throughput" ABR algorithm, first run. 49

B.7 Comparison of Buffer Level Evolution for ABR BOLA,Throughput and Dynamic 50

B.8 Comparison of Stall Duration Evolution for ABR BOLA,Throughput and Dynamic 51

B.9 Comparison of Resolution Evolution for ABR BOLA, Throughput, and Dynamic Algorithms 52

B.10 Comparison of RTT Evolution for ABR BOLA, Throughput, and Dynamic Algorithms 53

B.11 **Out of order and missing packets. Packet status vs time.** BOLA. 54

B.12 **Out of order and missing packets. Packet status vs time.** Dynamic. 54

B.13 **Out of order and missing packets. Packet status vs time.** Throughput. 55

Chapter 1

Introduction

Video streaming has grown to be one of the largest consumers of Internet bandwidth. Much of this traffic is from mobile devices, leading to the popularity of modern streaming methods with support for adaptive bitrate (ABR), allowing them to adjust the stream to perform best even as network conditions change. The most common adaptive streaming method is Dynamic Adaptive Streaming over HTTP (DASH), supported by YouTube, Netflix, Fubo, and other services. DASH uses traditional HTTP servers to serve content, with the process largely being led by the DASH-aware client [20].

Geostationary satellite networks have been used to provide Internet access for many years [6], with their fixed position in the sky ensuring the same satellite can always be reached. However, even with technological improvements increasing their bandwidth capabilities, these connections always have substantially higher latencies.

These high latencies cause many network protocols to perform poorly, including TCP [23]. Relatively little research has evaluated the performance of DASH streams over these geostationary satellite networks. This paper explores that topic, examining how manipulating a set of independent variables improves or degrades the stream's performance.

All experiments are performed using a commercial geostationary satellite connection, not simulated or emulated. A 10 minute video is streamed in full, with data collected on the network throughput and indicators of the user's Quality of Experience (QoE). Depending on the experiment, one of two independent variables is manipulated to examine its effects: the length of the video segments, set to either 5 seconds, 10 seconds, or 20 seconds; or the ABR algorithm used to make bitrate decisions, set to either buffer-based BOLA, a throughput-based algorithm, or a hybrid algorithm that mixes both approaches.

Analysis of the segment length results finds that longer segment lengths never stall (never letting the buffer of video reach empty), but achieve this by staying at the minimum bitrate, never changing to higher ones. Analysis of the ABR algorithm results finds that the buffer-based BOLA algorithm performs largely better than the other two, with fewer and shorter stalls, fewer bitrate changes, and more time spent at higher bitrates.

The rest of this paper is structured as follows: Chapter 2 provides background knowledge of the topics used in the work; Chapter 3 reviews existing work on these topics; Chapter 4 defines the structure of the experiments; Chapter 5 presents and analyzes the results of these experiments; Chapter 6 summarizes key observations and takeaways; Chapter 7 discusses limitations and proposes topics for future research.

Chapter 2

Background

This chapter provides an overview of the topics applied in this project.

2.1 Geostationary Satellite Networks

Satellites are commonly used to provide Internet access in remote locations where terrestrial Internet access is rare [13]. Traditionally, the satellites used for Internet service are placed in geostationary/Geosynchronous Equatorial Orbit (GEO), so that service does not depend on the satellite's current position in its orbit. However, geostationary satellite networks have a high round-trip time (RTT) of approximately 600 ms [16] due to the far distance from Earth.

The Bandwidth-Delay Product (BDP) of a network, measured in bits, is computed by multiplying the bandwidth of the network by its round-trip time. It represents the maximum amount of data that can be traveling within the network without being acknowledged at any given time [21]. While high-BDP networks may have high bandwidths that allow large transmissions, the long delay in sending messages along them can interfere with the performance of some network protocols. Modern GEO networks are capable of large bandwidths in the hundreds of Mbps, but the high RTT also implies a high BDP, making them vulnerable to these performance problems. For example, a network capable of 100 Mbps with average RTT of 600 ms, has a BDP of 7.5 MB.

2.2 Transport Protocols

2.2.1 Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) builds on the Internet Protocol's (IP's) base for network communications to provide reliable, ordered message transmission with the ability to identify and correct errors. Thanks to these capabilities, TCP is used for a large portion of Internet traffic [29], especially traffic that must ensure all the data arrives correctly.

2.2.1.1 TCP Handshake

As part of providing reliable data transmission, TCP begins any connection with a three-way "handshake" between the client and the server. The client begins with a `SYN` (synchronize) message, showing that it wants to start a new connection with the server. The server gets the `SYN` message, then responds with a `SYN-ACK` (synchronize-acknowledge) message that confirms it is ready to establish the connection with that client. The client finally responds with an `ACK` (acknowledge) message, which confirms and fully establishes the connection [43]. This initial handshake is valuable for TCP's promises of reliability, but requires at least one full round-trip of transmissions. In a high-RTT environment like GEO satellites, this overhead can be significant.

2.2.1.2 TCP and Congestion Control

Network congestion occurs when the amount of data currently being transmitted on the network exceeds the capacity of the components of the network [24]. This condition can lead to packets being lost in transmission, so avoiding it is important for the performance of the network. While the lost packets can be retransmitted, this extra data only contributes to the issue. To resolve congestion, the senders must be throttled to reduce the rate of data being transmitted [24].

As a Transport layer protocol, TCP plays a key role in this congestion control. TCP's congestion control scheme has two phases: Slow start, and congestion avoidance. Slow start works by increasing the size of the "window" that determines how much data it sends at once. It starts with a small window, and with successive sequences, exponentially increases the size of this window until one of the following end conditions are met [24]. If a packet is lost, slow start is restarted from the beginning with a minimum window size. If the preset slow start threshold `ssthresh` is reached, slow start is complete and the next phase, congestion avoidance, begins. Slow start is used to quickly find the capacity of the network. By default, Linux starts the window size at `cwnd` = 10 packets [41], and doubles that size with each iteration. Taking a goal capacity of 144 Mbps and an RTT of 750 ms (based on observed results in the later experiments), it will take approximately 8.25 seconds to ramp up to this maximum capacity at this growth rate.

Congestion avoidance is entered when, theoretically, the network's capacity has nearly been reached. As such, while there are many congestion avoidance algorithms with different approaches and techniques, its continued increase of the window size is typically much slower, often linear instead of exponential. Since Linux kernel version 2.6.18, TCP-CUBIC has been implemented as the default algorithm [19], with an approach that rapidly increases window size to reach saturation but slows down as it approaches to not overshoot.

2.3 HTTP

The HyperText Transfer Protocol (HTTP) is the backbone connecting web servers and clients, allowing free data exchange regardless of device or platform. It is built on a model where a client makes a request for media, such as text or images, to a server, which returns a response with the requested media or an error message indicating why it could not be returned. HTTP is largely stateless, meaning that each request is independent of each other and the server response is not linked to previous requests. HTTP has traditionally been built on top of TCP, using it for data transmission [1].

2.3.1 HTTP Versions

HTTP has seen many evolutions that added features or changed approaches, from HTTP/1.0, published in 1997, to HTTP/3.0, published in 2022.

HTTP/1.0 builds a simple client-server and request-response model. This version closes the underlying TCP connection after each request, requiring a new one to be opened each time [17].

HTTP/1.1 was introduced in 1999 to address some of the limitations of HTTP/1.0. The Keep-Alive header allows TCP connections to be reused for multiple requests, significantly reducing overhead. It introduced chunk transfer encoding, which allows the server to send a response in parts. Chunk transfer encoding is used by DASH to send video chunks over a persistent HTTP connection [46].

HTTP/2.0 made changes to support the increased complexity and sizes of data transferred over the modern Internet, such as converting messages from plaintext to binary, enabling compression of headers, and allowing multiplexing where parallel requests can be made over the same connection [14]. HTTP/2.0 is currently used by 39.3% of websites [8].

HTTP/3.0, the latest version, was published in 2022. It replaces TCP with a new protocol, QUIC, which is designed to adapt to more mobile devices and address issues with TCP. Among its changes, QUIC improves multiplexing performance by resolving the Head-Of-Line blocking issue, allowing data to be transferred more effectively [25]. HTTP datagrams are introduced, where requests and responses are sent in different packets [4]. HTTP/3.0 is currently used by 25.7% of websites [8].

2.4 Dynamic Adaptive Streaming over HTTP

Dynamic Adaptive Streaming over HTTP (DASH) is a multimedia streaming technique that uses conventional HTTP servers to serve content to DASH-aware clients [20]. It supports adaptive bitrate, which allows the quality level of the content to be adjusted throughout the stream to adapt to network and device conditions, maximizing the user's experience while fitting within the constraints of the streaming system's capabilities. DASH streaming is designed to be client-driven, where the client receives information on what is available for the stream and makes its own decisions on what to request next [20].

Content is prepared for DASH streaming through a process known as "DASHing," where it is split up into chunks of a fixed length called "segments." Multiple versions of the content, each at a different bitrate, can be provided to allow the client to request lower bitrate content, which is easier to download, or higher bitrate content, which is higher quality but more difficult to download. A manifest file is created that describes the content being made available, including the different quality levels and the bitrates required for each of them. For each segment, the client decides which quality level it will request based on its capabilities, current network conditions, or other factors.

2.4.1 ABR Algorithms

Video streaming over the Internet faces challenges due to varying devices, fluctuating bandwidth, and challenging network conditions such as high RTT in satellite connections. Adaptive Bitrate (ABR) streaming aims to provide a smooth viewing experience by selecting the best bitrate based on these conditions [18].

ABR algorithms can be categorized into three main types: throughput-based, buffer-based, and hybrid algorithms:

Throughput-based algorithms estimate the available network bandwidth and predict the bitrate the network is capable of handling for the next video segment. They decrease video startup time by predicting the throughput before the video starts and requesting the appropriate bitrate [42].

Buffer-based algorithms, like BOLA (Buffer Occupancy based Lyapunov Algorithm), use buffer levels to make decisions on the bitrate without predicting the throughput. BOLA uses buffer level thresholds to change the bitrate when certain buffer levels are reached [38].

Hybrid algorithms combine aspects of both throughput-based and buffer-based algorithms. One such algorithm, DYNAMIC, uses the fact that throughput-based algorithms perform better when the buffer level is low and buffer-based algorithms perform better when the buffer level is high, switching between the two based on the current buffer level to gain the advantages of both [37].

Dash.js, the official DASH reference player, incorporates one algorithm of each type: A throughput-based algorithm, BOLA as buffer-based, and DYNAMIC as hybrid [37].

Chapter 3

Related Work

This chapter reviews existing work relevant to the topics involved in this paper, noting key results and contributions.

3.1 Satellite Networks

The high throughput but extremely high latency of GEO satellite networks is a substantially different scenario from most networks, meaning that the performance of many networking protocols and applications built on traditional assumptions can be significantly different, often much worse.

Kachooei et al. [23] note that TCP's traditional slow start phase performs poorly on satellite links, "often overshooting and causing significant packet loss," but the TCP HyStart method designed to prevent this can instead exit too early, also reducing performance. They propose a modified approach that finds a balanced point to exit, observing improved start-up performance on an actual satellite connection.

3.1.1 TCP Congestion Control

TCP has had many different algorithms proposed for its congestion control phase. However, the distinct high-latency characteristics of a satellite network break many of the assumptions that traditional methods are built on, creating the potential for these algorithms to perform poorly.

Claypool, Chung, and Claypool [7] compare congestion control algorithms TCP-BBR, TCP-Cubic, TCP-Hybla, and TCP-PCC over a real, not simulated, satellite connection. TCP-BBR, TCP-Cubic, and

TCP-PCC are each designed for traditional networks and represent three different approaches to congestion control, while TCP-Hybla is optimized for satellite networks. They find that TCP-BBR and TCP-Cubic perform similarly, but note important differences between TCP-PCC and TCP-Hybla: TCP-PCC achieves high throughputs and low RTTs; while TCP-Hybla ramps up speed very quickly, making it a strong candidate for small downloads.

Liu et al. [28] analyze the effects of a Performance Enhancing Proxy (PEP) on TCP performance over a real, not simulated, satellite connection, comparing the same four congestion control algorithms as [7]: TCP-BBR, TCP-Cubic, TCP-Hybla, and TCP-PCC. They found that the PEP significantly improved the start-up performance of all algorithms, with particular benefit to TCP-Cubic and TCP-PCC, which otherwise exhibited slow start-up times in the satellite environment. With the PEP, all algorithms achieved a small download (described as a webpage) in about 5 seconds, "about 3 times faster than a default TCP-Cubic flow without a PEP."

Obata, Tamehiro, and Ishida [31] assess the TCP-STAR algorithm on a real, not simulated, satellite connection. STAR [32] was designed to be optimized for satellite connections, reducing damage to throughput caused by some transmission errors, and increasing the congestion window more quickly. Obata, Tamehiro, and Ishida [31] compare STAR to TCP-NewReno and TCP-Hybla, finding that "STAR and Hybla achieve better throughput when segment losses do not occur, while STAR achieves the best throughput when segment losses occur."

3.2 DASH

3.2.1 Segment Lengths

DASH uses small segments of video encoded at various quality levels for adaptive bitrate streaming. Both shorter and longer segment lengths have pros and cons: longer lengths have less player overhead and can improve encoding efficiency, but have slower adaptation to network conditions; while shorter lengths provide better latency and network adaptability, but increased risk of stalls [26]. In live video applications, shorter segment sizes can improve latency but may increase the risk of stalls and playback issues on some devices [33].

Performance is also affected by the nature of the connection: A persistent connection (supported by HTTP/1.1 and up) where the HTTP connection is reused for multiple requests creates less overhead in segment requests, while a non-persistent connection requires a new TCP handshake for each request, resulting in more overhead. Lederer [26] finds that, for a persistent connection, segment lengths of around 2 to 4 seconds are ideal, providing “a good compromise between encoding efficiency and flexibility for stream adaptation to bandwidth changes”; while for a non-persistent connection, longer lengths of around 5 to 8 seconds are better.

3.2.2 ABR Algorithms

Many factors can change what is needed for a stream to perform optimally, such as the use of multiple devices, different connection links, and changing network conditions. Many ABR algorithms have been proposed to address different scenarios and requirements. These algorithms can largely be placed into these distinct categories based on what factors they use to make decisions:

Throughput-based ABR algorithms make decisions using some measure of network throughput to estimate the available bandwidth. The Dash.js library includes a simple throughput-based ABR rule, which uses a sliding window approach: It computes throughput based on how long it took to download each segment, and then picks the highest available encoded bitrate that fits within that measured throughput [10]. Jiang, Sekar, and Zhang [22] propose FESTIVE, another classic throughput-based algorithm. FESTIVE looks at the mean of the measured throughput for the past five video segments and uses that to decide on the next bitrate [22]. FESTIVE also considers the fairness of the bandwidth allocation among multiple video players sharing the same bottleneck link. Zhou et al. [48] propose TFDASH, a throughput-based control scheme for video streaming with multiple clients using DASH. The core idea behind TFDASH is to avoid “off” periods during the downloading process for all clients, i.e., between all the clients, aim for the bandwidth to be fully utilized.

Buffer-based algorithms make decisions solely based on the buffer level, avoiding the overhead of throughput estimation. Spiteri, Urgaonkar, and Sitaraman [38] propose Buffer Occupancy based Lyapunov Algorithm (BOLA), which chooses the bitrate of each segment to download based on the current buffer level and the expected future buffer level. It uses a set of thresholds on the buffer level to make decisions: When the buffer level gets high enough, it upswitches to a higher quality level, or when it gets low enough, it downswitches to a lower one. It models bitrate adaptation as a utility-maximization problem, using Lyapunov optimization to minimize stalls and maximize video quality.

Hybrid algorithms use a combination of both buffer and throughput-based algorithms. The Dash.js library includes Dynamic, which switches between BOLA and the throughput rule discussed above, to attempt to gain the benefits of both. It uses BOLA when the buffer level is above 10 seconds, and switches back to throughput when it falls below 10 seconds [37]. De Cicco et al. [12] present ELASTIC, which applies feedback control theory to create an algorithm that can share a single link with multiple devices, creating a fair result. ELASTIC uses a feedback loop to adjust the video bitrate based on the available network bandwidth and the buffer level. ELASTIC does not generate an on-off traffic pattern that can otherwise lead to unfairness and underutilization [12]. Bentaleb et al. [3] also attempts to create a more fair result when multiple devices are sharing a single link. They propose GTA, which uses game theory to model ABR decisions as a bargaining process. GTA works by modeling the interactions between the video player and the network as a non-cooperative game, where the player and the network compete to maximize their utility.

Learning-based approaches apply machine learning to the ABR problem. Mao, Netravali, and Alizadeh [30] present Pensieve, which uses reinforcement learning to train a neural network that chooses the best bitrate for upcoming video chunks. It gathers information from client video players and does not depend on pre-set models or assumptions. Instead, Pensieve learns to make ABR choices based on the outcomes of previous decisions.

3.2.3 Quality of Experience (QoE) Metrics

The experience of a user watching a stream is subjective, as each person is sensitive to different aspects of the stream. Additionally, the application may require different characteristics: A broadcast of a live sporting event needs to be stable and always keep playing; alternatively, a stream of a scenic video that plays in the background may prioritize having the video be high quality and look good. With this in mind, to be able to compare streams it is important to define consistent, objective measures that contribute to a user's Quality of Experience (QoE).

Seufert et al. [36] details factors affecting the QoE of a stream, first itemizing and categorizing these factors, and second identifying a hierarchy of which are most important and when two factors contradict each other. They note four categories of "Perceptual" factors: Waiting Times include the time to start playing and time in stalls; Video Adaptation includes how frequently the quality switches and how big the switches are; Video Quality includes all the factors of the video quality; and Context Factors includes details of what is being watched and how it is being played. They found that stalls were more impactful to QoE than quality level changes, time to start playing, or video quality. Additionally, video quality was more important than framerate in "spatially complex videos," on small screens, or with fast foreground motion.

Yin et al. [47] and Bentaleb, Begen, and Zimmermann [2] use four QoE factors: Average video quality, average size/magnitude of quality changes, total stall time, and time to start playing. They use a weighted sum of these factors to be able to address different user preferences or application requirements. Yin et al. [47] discusses three archetypes: a balanced user, one that prefers the video quality not change much, and one that prefers the stream not stall.

Chapter 4

Methodology

This chapter describes the test bench used for and the procedures of the experiments.

4.1 Test Bench Setup

The experimental setup is depicted in Figure 4.1, consisting of a client, a server, and a network connection between them. Its components are discussed below.

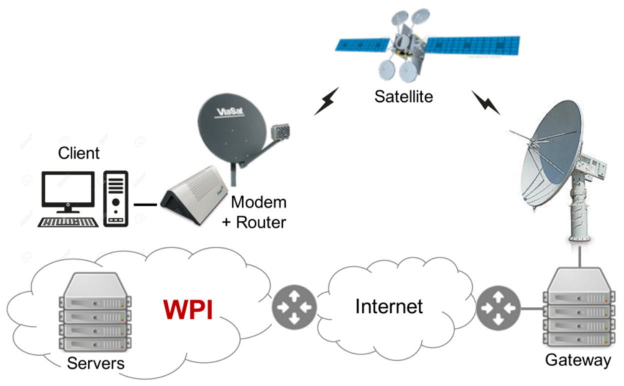


Figure 4.1: Experimental Setup

4.1.1 Components

4.1.1.1 Glomma

Glomma is the client device that initiates the video streams and receives the data. It runs Ubuntu 18.04.6 LTS with kernel version `4.15.0-202-generic`. It is equipped with a single Intel Core i7-5820k CPU at 3.3 GHz, 32 GB of RAM, and a single 4 TB HDD.

4.1.1.2 MLCNet Servers

The MLCNet servers are used as the source for the streaming data during the tests. Four servers are available, denoted A, B, C, and D. They are virtualized using Kubernetes, running on shared resources. While the configurations of each are slightly different, all tests discussed were performed on MLCNetC, so only its configuration is discussed. MLCNetC runs Ubuntu 18.04.6 LTS with kernel version `4.15.0-194-generic`. It is equipped with two cores of an Intel E312xx CPU (shared with other servers) at 2.5 GHz, 4 GB of RAM, and 42 GB of available HDD space.

Apache version 2.4.29 is used to serve the data used in the tests. DASH is designed to run on standard HTTP servers, so no DASH-specific configuration was performed. However, a Let's Encrypt SSL certificate was configured using Certbot, to provide HTTPS support for the streams.

4.1.2 Network Connection

Glomma can be configured to transfer data to the servers using either the satellite connection or the WPI LAN. MLCNet servers receive data from both sources via the WPI LAN, with the only distinction being the IP address that the data is received from.

Glomma is connected to the network by two Intel I210 Gigabit Ethernet Controllers, one connected to the WPI LAN via a Gigabit-capable link, and one connected to the Viasat router via a Gigabit-capable link. The Linux `ip route` utility is used to create static routes that ensure outgoing network connections directed towards an MLCNet server hostname are routed through the network controller connected to the Viasat link. These routes are deleted when the device restarts and may be deleted in other situations, so it must be verified that the correct routes were in place when a test began.

The MLCNet servers are connected to the network by a virtual Ethernet adapter, connected to the WPI network via a Gigabit-capable link.

4.1.2.1 Local Area Network (LAN) Connection

The WPI LAN connects to the Internet via several 10 Gigabit links, with each user throttled to a maximum of 1 Gbps. All devices used in these experiments are connected to the LAN via wired Ethernet.

4.1.2.2 Geostationary Satellite Connection

Satellite connectivity is provided by the Viasat network, which uses a set of satellites in geostationary orbit/Geosynchronous Equatorial Orbit (GEO). The large distance required by the orbits causes unusually high Round-Trip Times (RTT), despite the high bandwidth capability of the network. Due to the geographic location of the test bench at WPI, it connects to the ViaSat-2 satellite. A Viasat router/modem combination device is connected via Ethernet to the local device(s), and communicates with the satellite using a Ka-band outdoor antenna. The service plan set up provides a peak data rate of 144 Mbps. The Performance Enhancing Proxy (PEP) on the network was disabled during tests.

4.1.3 Test script

A test script was created for use during experiments. The first component is a webpage that integrates the Dash.js player and collects statistics from the player during the stream. It was based on the script created by Xu and Claypool [45], but with significant additions to collect more data during the course of the test and better support automation. While the script simply loads the latest version of Dash.js from its CDN, v4.6.0 was the latest published at the time of both experiments. The webpage is run in a headless environment, using Firefox v109.0.

The Dash.js player is highly configurable, including settings that affect its buffering behavior [11]. While these settings were experimented with early in the course of study, they were not focused on. The settings were fixed at the same values throughout all experiments, with these values depicted in Table 4.1.

Name	Value
`bufferToKeep`	20 sec
`stableBufferTime`	20 sec
`bufferTimeAtTopQuality`	40 sec
`fastSwitchEnabled`	`true`
`initialBufferLevel`	`NaN`

Table 4.1: Settings for buffer-related configuration of Dash.js player in test script.

The other component of the test script is a Python script, responsible for automating the majority of the test procedure so it can be easily and reliably replicated. It is run using Python v3.10.9. This script integrates with the webpage using Selenium v4.6.0, automatically inputting the manifest file to stream, starting the stream, and detecting when the stream has completed.

The Python script also launches other programs that collect additional data. UDPing [44] is used to track round-trip-times during the course of the test, sending a packet every 250 ms. This UDP-based utility is necessary because WPI's network blocks traditional ICMP-based pings. tshark, a terminal-oriented version of Wireshark, is used to collect network traces on both the client (v3.6.7) and server (v2.6.10) during the test. It takes all the collected data and puts it into a single folder for the current test.

The data collected by the test script and the reasoning for collecting each point are described in Table 4.2. The source code for the test script is available at [40].

4.2 Test Video

One test video was prepared and used for all tests, based on the Big Buck Bunny [15] sample video, an open-source film commonly used in the literature [3, 27, 45]. The video is 10 minutes and 33 seconds long, or 633 seconds in total. To support DASH streaming's adaptive bitrate, the video is re-encoded at multiple bitrates. Creating more quality levels allows more flexibility in bitrate adjustment, but takes up more storage space on the server hosting the content. Our choices for bitrates (referred to as "quality levels") were based on Xu and Claypool [45], but added an additional, higher quality level to ensure that the maximum bitrate was sufficient to stress the satellite link. The quality levels and their specifications are documented in Table 4.3. The base video, at a resolution of `3840x2160` pixels and 60 frames per second, was independently re-encoded into each quality level using FFmpeg [35] version `git-2020-02-24-bc9b635`. Those encoded videos were then DASHed into segments using mp4box [34] version `2.0-rev0-g418db414-master`. The segment length was adjusted depending on the experiment, and the settings are shared in those sections. Note that no audio was included in the encoded videos (using the `-an` FFmpeg command), as the DASH streams would not play correctly when audio was included.

Note that both the encoding and DASHing steps have several parameters that can be changed, affecting the output and results. One pair of parameters caused unexpected issues. With the mp4box utility, the `-dash (number)` option creates output videos with segment lengths (a DASH concept) of the given duration, while the `-frag (number)` option creates output videos with fragment lengths (an MP4 concept) of the given duration. In theory, these may be set to different values as long as the fragment length is less than or equal to the segment length, since the MP4 video may not play properly if a fragment is split up and

Name	Frequency	Method	Purpose
Realtime elapsed since start of test	Every 0.5 seconds	JavaScript on webpage	To find out how long the test has been running when events occur.
Video time elapsed since start of test	Every 0.5 seconds	JavaScript on webpage	To find out how far in the video has gotten when events occur.
Current framerate, resolution, and bitrate of video	Every 0.5 seconds	JavaScript on webpage	Determine the quality level playing at any given time.
Current size of buffer	Every 0.5 seconds	JavaScript on webpage	Understand how buffer-based ABR algorithms make their decisions.
Current estimated throughput	Every 0.5 seconds	Dash.js player output	View the player's understanding of the network conditions.
Time video took to start playing	Once at beginning	JavaScript on webpage	Measure QoE metric.
Start time, stop time, and length of stalls	At every stall	JavaScript on webpage	Measure QoE metric.
Time and new level of quality change	At every quality change	JavaScript on webpage	Measure QoE metric.
Wireshark trace from client	Constant throughout test	Wireshark	View network activity on the client device for test data and to identify conflicting traffic.
Wireshark trace from server	Constant throughout test	Wireshark	View network activity on the client device for test data and to identify conflicting traffic.
Packet round-trip time and loss data	Every 0.25 seconds	UDPing	Identify if the network is behaving differently from expected or dropping too many packets.

Table 4.2: Descriptions of the data collected by the test script and the purposes of each point.

Quality Level	Resolution	Framerate	Bitrate
1	480x270	60 FPS	2 Mbps
2	640x360	60 FPS	3 Mbps
3	960x540	60 FPS	5 Mbps
4	1280x720	60 FPS	10 Mbps
5	1920x1080	60 FPS	17.2 Mbps
6	3840x2160	60 FPS	40 Mbps

Table 4.3: Quality levels the video was prepared in, and the specifications of each.

only part is downloaded at a time. However, in testing, this produced videos that were encoded incorrectly, displaying visual artifacts, skipping parts of the video, and jumping around to other parts of the video than expected. Setting fragment length always equal to segment length resolved this issue, and all results shared in this paper were DASHed using this condition.

4.3 Pilot Tests

While the Dash.js player is a common DASH player in the literature [3], the test script used by this paper customizes Dash.js's behavior (see Section 4.1.3) and adds many additional systems and routines. In addition, the videos used during the tests were encoded and DASHed with different tools and different settings. As such, it is important to verify that these modifications do not substantially affect the behavior of the stream, incorrectly skewing results. A series of pilot tests were conducted to check this.

4.3.1 Format

The pilot tests were conducted on a different computer from the rest of this paper, a desktop PC running Windows 10 with a display connected. This computer was configured to route all of its traffic through the Viasat link. Tests were recorded using Open Broadcaster Software Studio (OBS Studio) for later review and assessment. Since the other players did not have the same systems for collecting performance data during the test, objective comparison between them could not be made using these performance metrics. Instead, a text description of the performance was written down during the course of the test, focusing on the stalls and quality level changes experienced.

The player used in this paper was compared to two other players. The DASH Industry Forum (DASH-IF) provides an online reference player for the Dash.js library [9], demonstrating its capabilities and customization options. v4.5.0 of this player was included in the comparison. Bitmovin provides an online reference player for Bitdash [5], their proprietary player based on the open-source "libdash" library also published by them. The version of this demo available on 2023-01-22 was included in the comparison; the exact version number was not recorded during the test, but it was between v8.98.0 and v8.111.0.

Each player was tested on a video provided by Streamroot [39], which is delivered over the Akamai CDN. Its quality levels are described in Table 4.4, with bitrates ranging from 3 Mbps to 20 Mbps. The 20 Mbps maximum bitrate is large enough to stress the satellite connection, though not as high as the maximum of the video prepared for this paper. The video is approximately three minutes long, not as long

Quality Level	Resolution	Framerate	Bitrate
1	`1280x720`	∼24 FPS	3 Mbps
2	`1920x1080`	∼24 FPS	5 Mbps
3	`2880x1620`	∼24 FPS	7 Mbps
4	`3840x2160`	∼24 FPS	10 Mbps
5	`3840x2160`	∼24 FPS	15 Mbps
6	`3840x2160`	∼24 FPS	20 Mbps

Table 4.4: Quality levels of Streamroot video used in pilot tests.

as this paper's test video but long enough for the satellite connection to ramp up to maximum bitrates. The manifest file used the "SegmentBase" format, same as this paper's test video.

4.3.2 Results

Only one full trial of each test was completed. For concision, quality levels (QL) are abbreviated as QL1 (lowest) through QL6 (highest). **Note that the quality levels described in this section are *not* connected to the quality levels in the rest of the paper.**

DASH-IF player: Took 10 seconds to begin playing. Began at QL1. QL changed frequently, in the range QL1 to QL6, for the first 40 seconds of the video. At that point, QL changed to QL5 and stayed there, maintaining a buffer size of around 5 seconds, for 45 seconds. Then QL dropped to QL1, and the buffer size started to build up to around 15 seconds, with QL slowly climbing up to QL6. After 20 seconds, QL dropped to QL1 as the buffer level nearly hit zero, but quickly returned to QL5. After 40 seconds, QL increased to QL6 and stayed there for the last 10 seconds of the video.

Bitmovin player: Took 15 seconds to begin playing. Began at QL1. Within the first 10 seconds, QL increased to QL4 and then stalled for 7 seconds. After about 20 seconds, QL increased to QL5 and immediately stalled for 10 seconds. QL dropped to QL4 and resumed playing, maintaining buffer level size of 1-6 seconds, for 1 minute. QL dropped to QL2 as it stalled for 3 seconds. QL stayed at QL2, with buffer level size increasing to 15 seconds, for about 1 minute. At that point, QL increased to QL6 for the last 20 seconds of the video.

Our player: Took 13 seconds to begin playing. Began at QL1. Attempted QL2 for a short time but returned to QL1 and built up buffer for around 30 seconds. Upswitched to QL6 and swapped between that and QL5 for about 45 seconds. With buffer size falling, QL dropped down between QL2 and QL4 and built up buffer size to around 15 seconds. After 10-15 seconds of time elapsed, upswitched back to QL6 and stayed there for the rest of the test, maintaining a buffer size of 10-15 seconds.

4.3.3 Discussion

All three players took a comparable amount of time to start playing, about 10-15 seconds, and started at the lowest quality level. The DASH-IF player made by far the most quality level changes at the beginning, going through the entire range of levels. Our player changed fewer times, briefly higher attempting a higher quality level before returning to the minimum. The Bitmovin player attempted a couple upswitches early on, but was unsuccessful with both resulting in stalls.

After the start, all players found a stable quality level that they stayed at for some time. However, after roughly a minute, each found the buffer level falling and downswitched, with DASH-IF and our player avoiding the stall experienced by Bitmovin.

Bitmovin stayed at a low quality level for most of the remaining time, only changing to maximum for the last 20 seconds. Both the DASH-IF player and our player were able to reach the top two quality levels and stay in that range for the rest of the test. Note that all players were able to reach the maximum quality level by the end of the video, but varied in how long they spent there.

4.3.4 Conclusion

The Bitmovin player appeared to be the most aggressive in upswitches, and was also the only player to stall. While all three players experienced a sudden drop in buffer level in the middle of the test, the Dash.js-based players (DASH-IF, ours) were just barely successful in avoiding stalls. The DASH-IF player changed quality levels many more times at the start of the video than the others. Overall, our player behaved similarly to the other Dash.js-based player (DASH-IF), indicating that the choice of player is not significantly different.

4.4 Experimental Setup

4.4.1 Test Procedure

The test is run using an automated script that handles launching the test and organizing the resulting data in to a consistent location. At the beginning, it outputs the current values for CPU utilization, logged-in users, and network route configuration. These are used to verify that the client has the correct routes configured and that external factors, like excessive utilization from another user, are not impacting test results. The test is run in headless mode.

The order of startup for each component was chosen to ensure that each data collection component has sufficient time to start operating, before the stream itself begins. For example, tshark will return from a command to launch and allow execution to continue before it has completely started collecting data. This ordering is depicted in Algorithm 1.

The stream completing is automatically detected by Selenium using a DOM element that is placed on the page by JavaScript when the video ends. If the stream takes more than 30 minutes to complete, it is considered to have "timed out," and is automatically stopped. Timed-out results are manually discarded.

Algorithm 1 Sequence of events in the test procedure.

```
1: Output validity verification data
2: Start Wireshark on server
3: Start UDPing server on server
4: Start Wireshark on client
5: Start UDPing client on client
6: Launch browser and open test webpage
7: Enter test manifest file on webpage
8: Detect stream finishing (or timeout and kill manually)
9: Stop UDPing client on client
10: Stop UDPing server on server
11: Stop Wireshark on server
12: Stop Wireshark on client
13: Download Wireshark data from server
14: Wait 5 seconds for local Wireshark capture to complete
15: Move all collected data into a directory for this test and output the name
16: Close browser and exit
```

4.4.2 Variables

4.4.2.1 Independent Variables

Two variables were studied to understand their impact on DASH performance. Table 4.5 introduces the independent variables investigated in this experiment.

The segment length affects the duration of the individually downloadable video segments. Generally, longer segments are larger in size. Lederer [26] found that segment lengths had a notable impact on DASH performance, due to the overhead required for requesting and downloading each segment. Shorter segments take less time to download due to the smaller size, but the overhead caused by starting the download is relatively fixed and therefore is a more significant portion of the overall size for a shorter segment. They found that 2-4 seconds was an optimal segment length. However, their study used a network latency of

Parameters	Rationale
Segment length	Overhead in requesting segments
ABR algorithm	Characteristics of different algorithms may fit high RTT

Table 4.5: Independent variables

150 ms, significantly less than the 600 ms expected in a GEO satellite network, so the overhead was expected to play an even more significant role in this paper's scenario.

The ABR algorithm affects what information the DASH player uses to make quality level decisions and how it uses it. The Dash.js player includes three standard algorithms, each representing a different category of approach: Throughput (rate-based); BOLA (buffer-based); and Dynamic (hybrid) [37]. Dynamic is set as the default of Dash.js. While these algorithms represent a mix of approaches on their own, there is still significant interest in studying their performance and proposing new algorithms that may perform better in certain scenarios [3, 12, 22]. Research on ABR algorithm performance in GEO satellite networks is limited.

4.4.2.2 Dependent Variables

A variety of metrics were collected to first check the validity of results, then to objectively evaluate and compare the performance of each stream. The full set of data collected is described in Section 4.1.3; the metrics treated as indicators of the stream performance are discussed here. "Measured throughput" is calculated using the packet capture data from the Wireshark trace on the client, using the packet sizes to determine the amount of data transferred over time. It is the only indicator that is not observed by the user, but provides insight into how successful the client was at receiving stream data. A higher average measured throughput should indicate a better stream as it would be able to transfer higher quality level video segments, but if the measured throughput was sometimes high and sometimes low it could indicate the quality level changing frequently which is not desirable.

The other indicators can be considered Quality of Experience (QoE) metrics, as they determine how good the stream appears to the user. Most QoE metrics assessed in this paper are based on Bentaleb, Begen, and Zimmermann [2] and Yin et al. [47]. "Time to begin playing" is the amount of time elapsed between the stream being started and the content itself actually beginning to play. The content only begins to download once the stream is started, and it takes time to be downloaded. A longer time to start means the user has to wait for longer. "Number/size of stalls" relates to how frequently the stream's buffer reaches zero and has to stop to wait for more content to download (termed a "stall"), and how long the user has to

Parameters	Rationale
Measured throughput	Determines video quality that can be transferred
Time to begin playing	Waiting time should be as minimum as possible
Number/size of stalls	Stalls frustrate user
Number of quality changes	Unstable quality worsens experience
Average quality level	Higher quality is better
Absolute time to finish	Determined by stalls and other issues along the way

Table 4.6: Dependent variables

wait for that content to download once a stall begins. More and longer stalls frustrate the user. "Number of quality changes" captures how often the stream player changes to a higher or lower quality level via the adaptive bitrate selection algorithm. A stream that oscillates between the lowest and the highest quality levels may be less desirable than a stream that chooses a middle quality level and stays there consistently. "Average quality level" measures what quality level the stream was at most of the time; calculated in this paper as the mode, or most common. A stream that spends more time at a higher quality level is more desirable. "Absolute time to finish" is based on how long elapsed between the stream starting and the stream completing, and is compared to the actual length of the video to determine how much extra time was needed to finish it, due to stalls and other issues. A stream that took 30 minutes to complete a 10 minute video is less desirable than a stream that took 15 minutes to complete that same video. It is important to note that these metrics do not always change together, and in fact sometimes act as tradeoffs, where improving one metric worsens another. For example, a stream can be tuned to reduce the number of stalls by fixing it at the lowest quality level, which decreases the average quality level. Depending on the application of the stream, some tradeoffs may be more optimal.

Table 4.6 includes the dependent variables we are trying to quantify.

Chapter 5

Analysis

This chapter presents our test data and identifies notable results. The first subsection focuses on the segment length experiment, while the second focuses on the Adaptive Bitrate (ABR) experiment.

5.1 Segment Length Test

In the segment length test, a total of $n = 3$ full runs were completed for each of the three settings (5 second, 10 second, and 20 second). All results that do not identify individual runs or name a specific one are aggregate results where all runs were factored in.

5.1.1 Time to Start Playing

Longer segments are larger in file size, so they are expected to take longer to start playing. As observed in Table 5.1, both 10 sec and 20 sec segment lengths take longer than 5 sec, but 10 sec and 20 sec are not substantially different. This suggests that the time required to download may "plateau" once the connection has enough time to ramp up in speed.

Setting	Mean (sec)	95% Confidence (sec)	Stdev (sec)
5 sec	6.8	±2.0	0.6
10 sec	11.1	±0.8	0.3
20 sec	11.5	±1.9	0.6

Table 5.1: **Segment length test, time to start playing.** Time elapsed between starting the stream and the video beginning to play, due to downloading the first segment.

Setting	Mean (#)	95% Confidence (#)	Stdev (#)
5 sec	45.0	±8.2	5.9
10 sec	0	±0	0
20 sec	0	±0	0

Table 5.2: **Segment length test, stall counts.** Total number of individual stalls observed over the course of the test, regardless of the length of the stall.

Setting	Mean (sec)	95% Confidence (sec)	Stdev (sec)
5 sec	4.0	±0.6	3.2
10 sec	0	±0	0
20 sec	0	±0	0

Table 5.3: **Segment length test, stall lengths.** Average lengths of stalls observed over the course of the test.

5.1.2 Stall Counts

Longer segments are supposed to be more stable, as they mean the player has more video to work with when before it needs to download a new segment. The data in Table 5.2 support this, with 10 sec and 20 sec never stalling at all.

5.1.3 Stall Lengths

Data on stall lengths are depicted in Table 5.3. Once again, since only 5 sec segment length stalled, it is the only one with nonzero results, exhibiting average stall lengths of 4.0 seconds. Figure 5.1 shows the distribution of stall lengths for each setting visually.

5.1.4 Total Amounts of Stall Time

Data on the total amount of time spent in a stall over the course of the test are depicted in Table 5.4, computed as the product of the number of stalls and the average length of those stalls. Since only 5 sec segment length stalled, it is the only one with nonzero results. With a total video time of 633 seconds, stall time took up 22% of the total time required to complete a stream on average.

Setting	Total (sec)
5 sec	180.3
10 sec	0
20 sec	0

Table 5.4: **Segment length test, total amounts of stall time.** Total amount of time spent in a stall over the course of the test, the product of the number of stalls and the average length of those stalls.

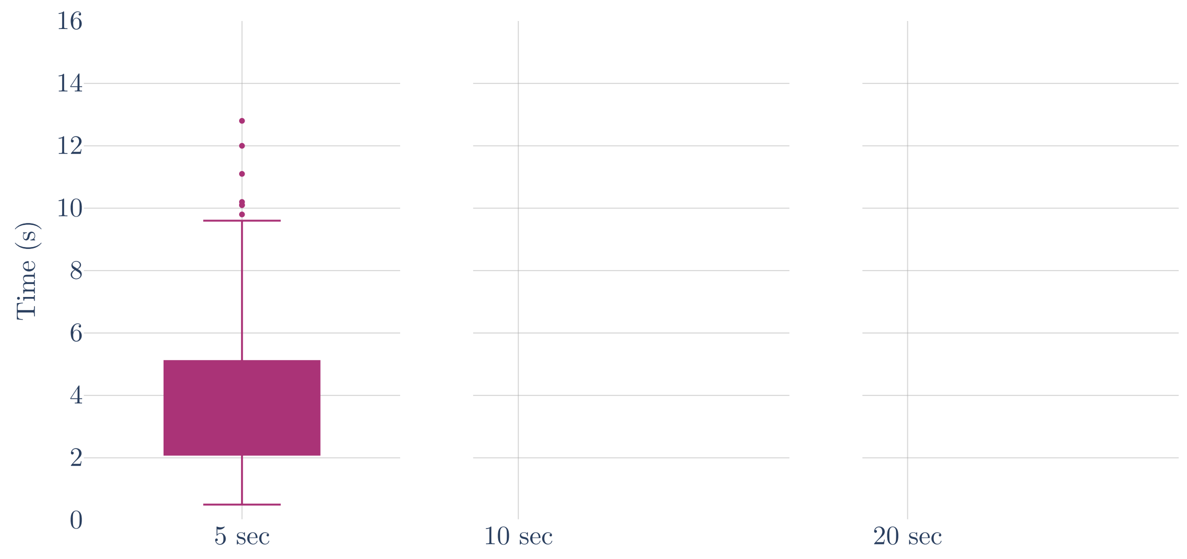


Figure 5.1: **Segment length test, stall length box plots.** Distribution of stall lengths over the course of the test.

Setting	Mean (#)	Stdev (#)
5 sec	62.7	3.3
10 sec	0	0
20 sec	0	0

Table 5.5: **Segment length test, number of quality level changes.** Number of quality level changes observed over the course of the test, regardless of whether it is an upswitch or downswitch.

5.1.5 Quality Level Changes

Table 5.5 shows the average number of quality level changes observed over the course of the segment length tests. 5 seconds is the only one to change quality at all, and changed an average of 62.7 times in total.

5.1.6 Network Throughputs

Table 5.6 compares estimated throughput numbers between each setting. 5 sec segment length achieves substantially higher estimated throughputs, but it should be noted that the throughput is limited by the quality levels: If a low bitrate segment is being downloaded, the throughput will never be that high because the ramp-up time becomes a more significant part. With 10 sec and 20 sec segment lengths never changing quality, this occurs, and limits their throughputs. Figure 5.2 shows the estimated throughput over time for each run of the 5 second segment length setting.

Setting	Mean (Mbps)	Stdev (Mbps)
5 sec	36.6	29.8
10 sec	3.0	0.8
20 sec	4.1	0.9

Table 5.6: **Segment length test, estimated throughputs.** Throughput metrics, as reported by the Dash.js player's throughput estimator. Averaged over all runs of the setting.

Setting	Mean (Mbps)	Stdev (Mbps)
5 sec	7.5	17.0
10 sec	2.1	2.9
20 sec	2.1	4.0

Table 5.7: **Segment length test, measured throughputs.** Throughput metrics, as measured by the Wireshark trace on the client. Averaged over all runs of the setting.

Table 5.7 compares measured throughput numbers between each setting. Similar to the estimated numbers, 5 sec segment length achieves much higher throughputs, though this can be explained by the same issue referenced above. Figure 5.3 shows the measured throughput over time for each run of the 5 second segment length setting.

Figure 5.4 compares the estimate of throughput to the measured throughput for the first run of the 5 sec segment length. The estimated throughput spikes as segments are downloaded, then falls when the buffer is filled up. In this case, it creates an "envelope" over the top of the measured throughput.

Notice that in the 5 second segment length test, the measured throughputs are smaller, roughly 1/5 of the estimated throughputs. This difference may be explained by the fact that the estimated throughputs are "smoothed" by being a lagging indicator of network traffic, as opposed to the very bursty nature of actual network requests. Since the estimated throughput does not return to near zero almost immediately after spiking (see Figure 5.4), there are fewer zero values to reduce the arithmetic mean compared to the measured throughput.

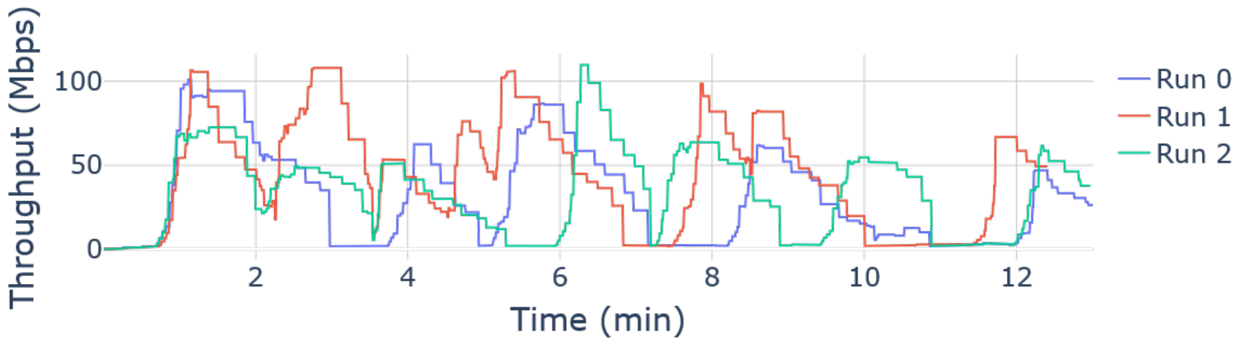


Figure 5.2: **Segment length test, estimated throughput vs time.** Throughput over time, as reported by the Dash.js player's throughput estimator. 5 second segment length.

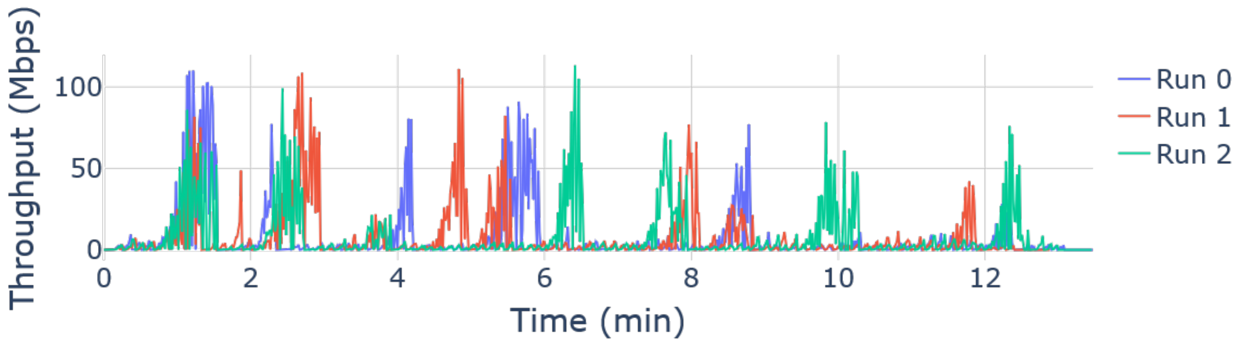


Figure 5.3: **Segment length test, measured throughput vs time.** Throughput over time, as measured by the Wireshark trace on the client. 5 second segment length.

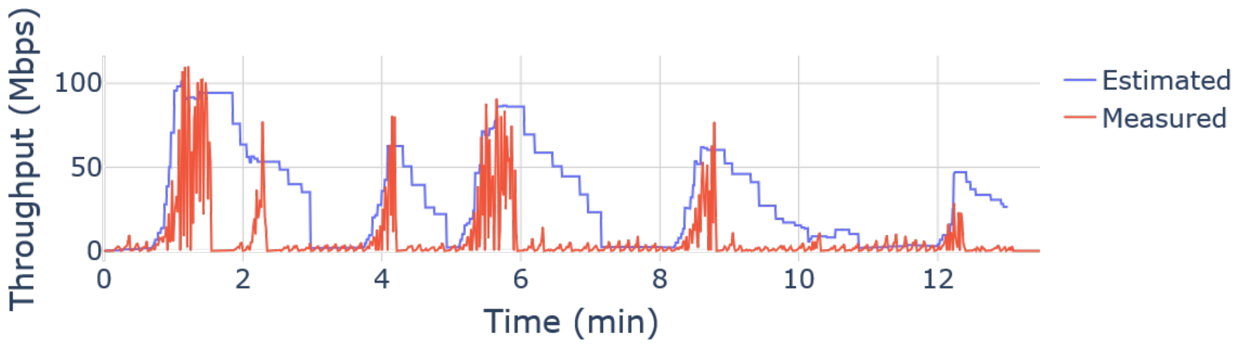


Figure 5.4: **Segment length test, estimated and measured throughput vs time.** Comparing estimated throughput to observed throughput over time. 5 second segment length, first run.

Setting	Mean (sec)	95% Confidence (sec)	Stdev (sec)
Dynamic	4.6	±0.7	0.6
Throughput	4.4	±0.4	0.3
BOLA	4.5	±1.7	1.2

Table 5.8: **ABR algorithm test, time to start playing.** Time elapsed between starting the stream and the video beginning to play, due to downloading the first segment.

Setting	Mean (#)	95% Confidence (#)	Stdev (#)
Dynamic	25.8	±14.5	16.5
Throughput	27.7	±15.8	18.0
BOLA	16.6	±11.1	11.3

Table 5.9: **ABR algorithm test, stall counts.** Total number of individual stalls observed over the course of the test, regardless of the length of the stall.

5.2 ABR Algorithm Test

In the ABR algorithm test, a total of $n = 6$ full runs were completed for each of the three settings (Dynamic, Throughput, BOLA). All results that do not identify individual runs or name a specific one are aggregate results where all runs were factored in. However, the third run of BOLA timed out and was not repeated, so its results were discarded. Therefore, any aggregate results of BOLA only have $n = 5$ runs factored in.

5.2.1 Time to Start Playing

Longer segments are larger in file size, so they are expected to take longer to start playing. Table 5.8 depicts measurements of the time it took each setting to start in the ABR algorithm experiments. The means of all three algorithms are similar, though BOLA has a slightly higher standard deviation.

5.2.2 Stall Counts

Table 5.9 shows that BOLA had a lower average total number of stalls, though it still fell within the confidence intervals of the other values. This may indicate that BOLA is more successful in selecting quality levels in the satellite scenario.

Setting	Mean (sec)	95% Confidence (sec)	Stdev (sec)
Dynamic	4.6	±0.49	3.1
Throughput	4.6	±0.46	3.0
BOLA	3.6	±0.53	2.4

Table 5.10: **ABR algorithm test, stall lengths.** Average lengths of stalls observed over the course of the test.

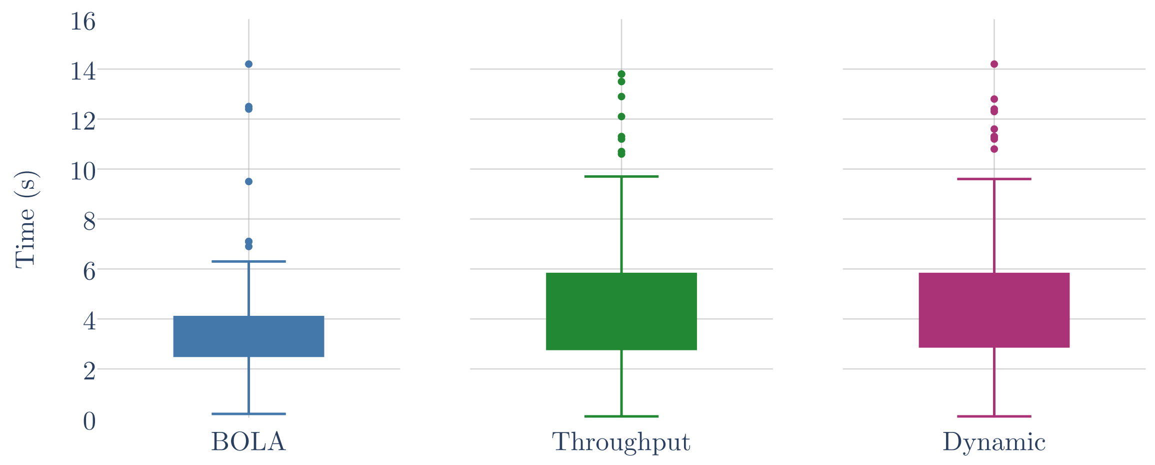


Figure 5.5: **ABR algorithm test, stall length box plots.** Distribution of stall lengths over the course of the test.

5.2.3 Stall Lengths

Table 5.10 contains the average lengths of stalls for each algorithm, while Figure 5.5 shows the distribution of stall lengths for each setting. BOLA has a lower average length of stall, and its confidence interval falls just outside the confidence interval of the Throughput algorithm, indicating a significant difference between these. It is just barely within the confidence interval of Dynamic, which could also suggest a difference there. Once again, BOLA exhibits signs of being better at selecting bitrates in the satellite network.

5.2.4 Total Amounts of Stall Time

Table 5.11 shows the total amount of time each setting spent in a stall on average, over the course of the test. As expected with fewer, shorter stalls, BOLA spends the least time in a stall, roughly half as long as the others.

Setting	Total (sec)
Dynamic	117.7
Throughput	128.1
BOLA	59.8

Table 5.11: **ABR algorithm test, total amounts of stall time.** Total amount of time spent in a stall over the course of the test, the product of the number of stalls and the average length of those stalls.

Setting	Mean (#)	Stdev (#)
Dynamic	48.2	17.7
Throughput	48.0	18.6
BOLA	35.2	19.8

Table 5.12: **ABR algorithm test, number of quality level changes.** Number of quality level changes observed over the course of the test, regardless of whether it is an upswitch or downswitch.

5.2.5 Quality Level Changes

Figure 5.7 and Table 5.12 show BOLA experiencing fewer quality level changes, and its confidence interval for this number falling outside those of the other two algorithms. This provides evidence for BOLA picking more optimal quality levels, having to then try again less often. Additionally, Figure 5.6 shows that BOLA spends less time at the lower quality levels (1-4) and much more time at Quality Level 5. Between fewer changes and a larger share of higher quality levels, BOLA appears to perform better.

5.2.6 Network Throughputs

Table 5.13 shows the DASH player's averages for its estimate of throughput for each algorithm. The standard deviations of each are comparable, but BOLA has a notably higher mean value for estimated throughput, roughly 23% higher. With BOLA being the only algorithm of the two to not consider throughput at all in its decisions, it suggests that ignoring its measures of throughput allows it to achieve better performance. When looking at actual measured throughputs in Table 5.14, the difference between BOLA and the other algorithms is smaller, but still around 17%.

Figure 5.8 shows the estimated throughput over time for the BOLA algorithm, for each run. Figure 5.9 shows the actual measured throughput over time for the BOLA algorithm, for each run. Figure 5.10 compares the measured to actual throughputs directly to each other, for the BOLA algorithm, for each run.

As with Section 5.1.6, the measured throughputs are smaller, though this time the measured throughputs are instead about 1/3 of the estimated throughputs. The reason for this is likely similar, as Figure 5.10 exhibits the same behavior.

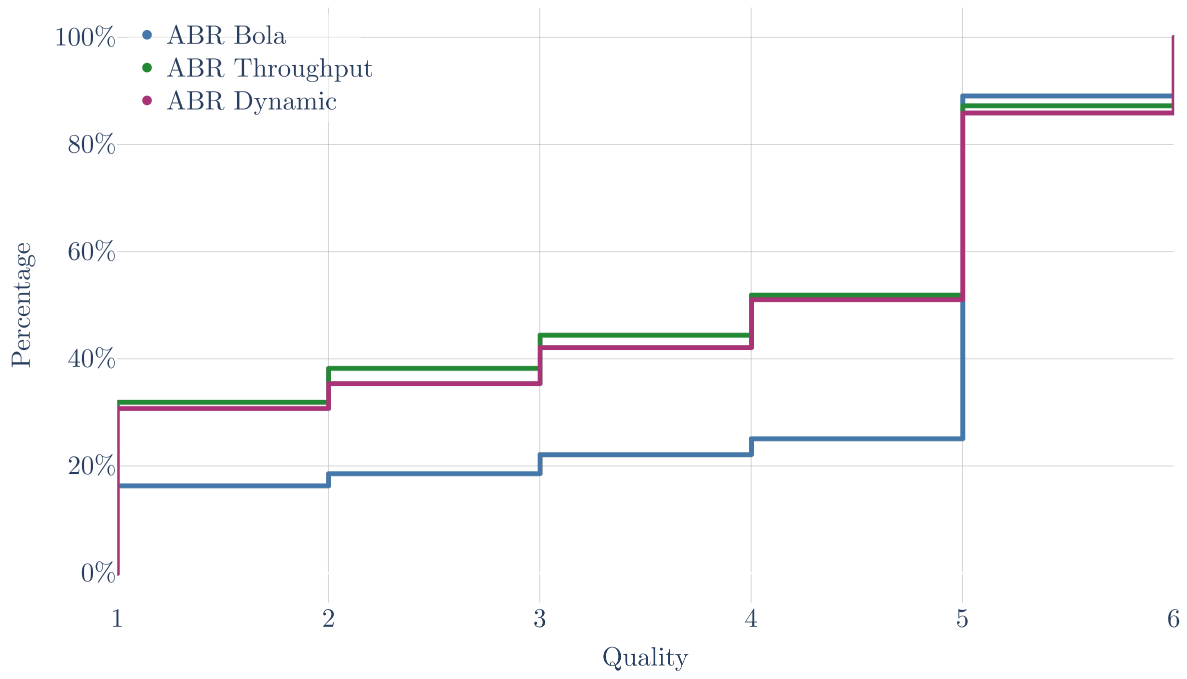


Figure 5.6: **ABR algorithm test, quality level distribution.** Distribution of quality levels over the course of the test.

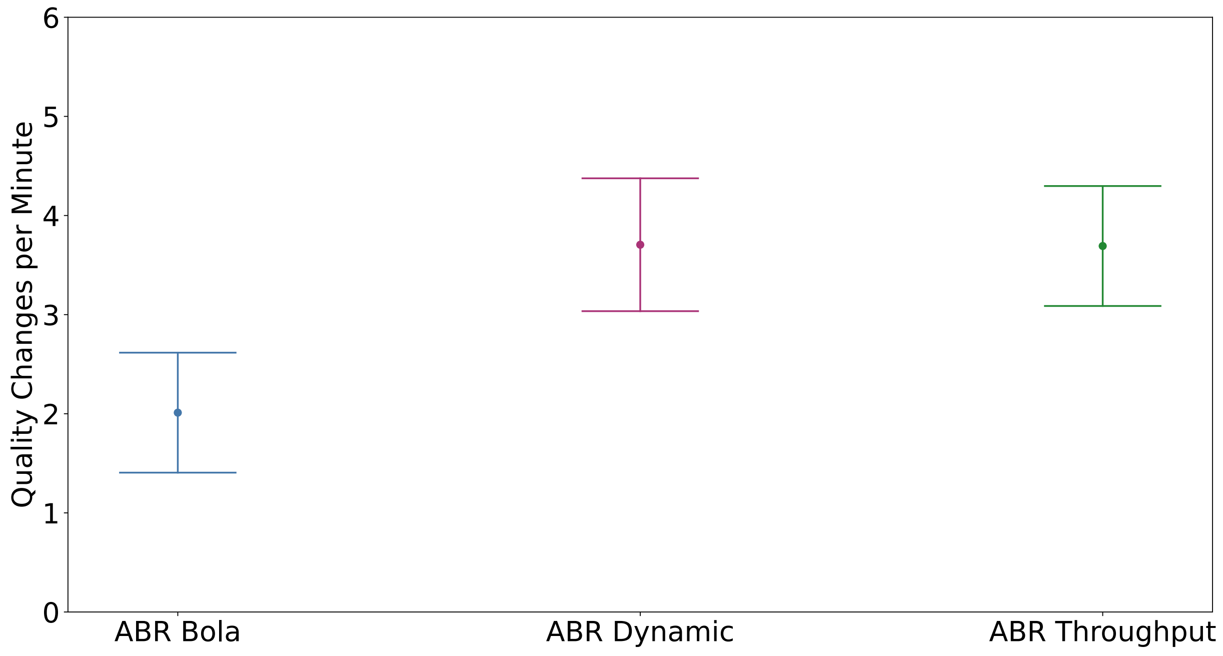


Figure 5.7: **ABR algorithm test, quality level change distribution.** Distribution of quality level changes per minute over the course of the test.

Setting	Mean (Mbps)	Stdev (Mbps)
Dynamic	48.8	31.4
Throughput	46.4	32.2
BOLA	60.1	32.2

Table 5.13: **ABR algorithm test, estimated throughput.** Average and standard deviation of throughput, as reported by the Dash.js player's throughput estimator. Averaged over all runs of the setting.

Setting	Mean (Mbps)	Stdev (Mbps)
Dynamic	14.7	22.6
Throughput	14.5	22.4
BOLA	17.1	26.6

Table 5.14: **ABR algorithm test, measured throughput.** Average and standard deviation of throughput, as measured by the Wireshark trace on the client. Averaged over all runs of the setting.

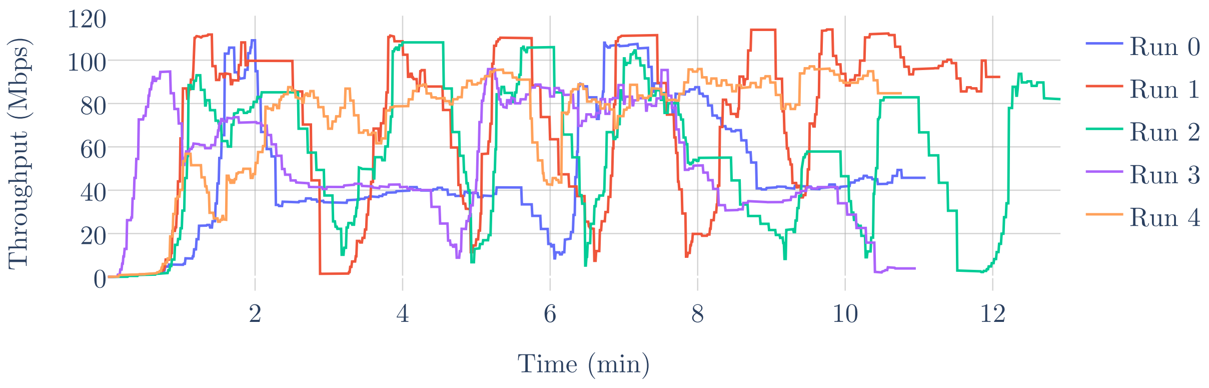


Figure 5.8: **ABR algorithm test, estimated throughput vs time.** Throughput over time, as reported by the Dash.js player's throughput estimator. "BOLA" ABR algorithm.

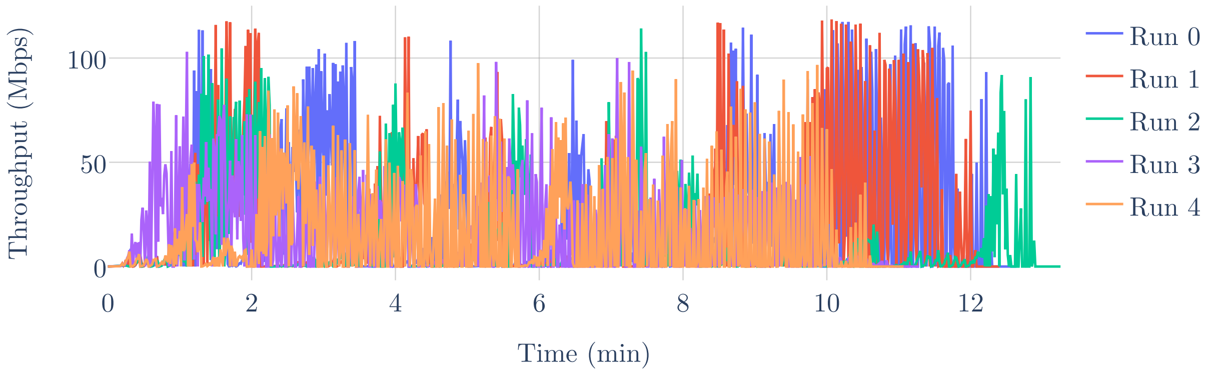


Figure 5.9: **ABR algorithm test, measured throughput vs time.** Throughput over time, as measured by the Wireshark trace on the client. "BOLA" ABR algorithm.

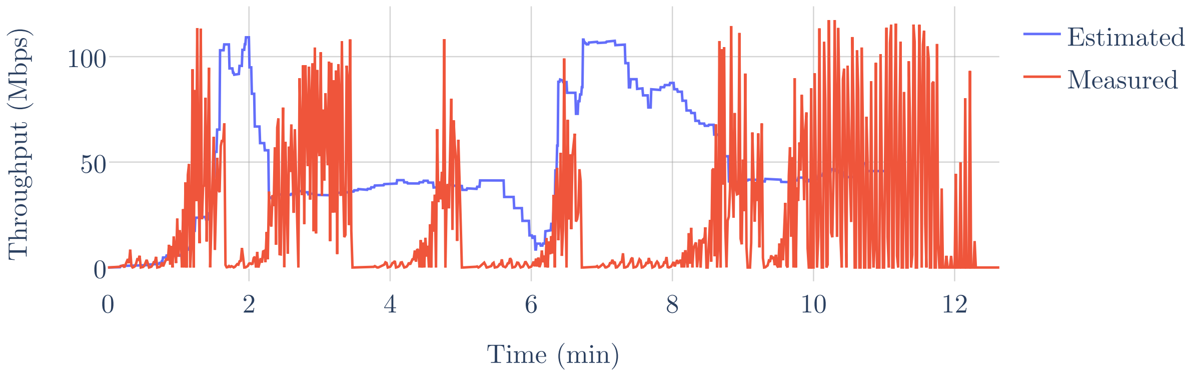


Figure 5.10: **ABR algorithm test, estimated and measured throughput vs time.** Comparing estimated throughput to observed throughput over time. "BOLA" ABR algorithm, first run.

Chapter 6

Conclusion

The results of the segment length experiment are fairly simplistic, due to a clear division between the settings: The 10 second and 20 second segment lengths never stalled, but also never changed quality levels and always stayed at the minimum level. While the stability of those higher segment lengths is desirable, the 3 Mbps bitrate of the video is poor, well below the capabilities of the satellite connection. Overall, the hypothesis that the larger segments would allow the HTTP connection more time to ramp up to high speeds was not supported by our evaluation.

A clear reason for the unexpected behavior of the higher segment lengths was not identified. The estimated throughput never gets very high, which could suggest that the player may be limiting itself by assuming the low throughput indicates it could not handle a higher bitrate, while it is actually just because the low bitrate video never gives it enough time to ramp up the bitrate. However, all experiments showed the player starting at the same low bitrate, and the others achieved higher bitrates. It could be a consequence of the limited buffer size set: The regular buffer size of 20 seconds allows a maximum of four segments at 5 seconds each, but only two for 10 seconds, or one for 20 seconds. With this restricted buffer, the player may not think it has the space to try any downloads that could be unsuccessful.

The ABR algorithm experiment showed notable differences in performance between the algorithms. BOLA had fewer stalls and those stalls were, on average, shorter. Its stall lengths were shorter than the Throughput algorithm's, though just barely not significantly different from the Dynamic algorithm. BOLA also had fewer quality level changes, significantly lower than Throughput and Dynamic; and spent a much larger share of time at the second-highest quality level. BOLA achieved higher average throughputs, both in its internal estimate and the actual observed values.

These results largely suggest that BOLA is better for the high-BDP GEO satellite network. This could be because it is difficult for the throughput estimator to achieve a good estimate of the network's bandwidth, due to the high latency, so the algorithm that does not use this estimate is more successful.

Chapter 7

Future Work

There is substantial opportunity to either address limitations of this work, or to continue it by expanding the scope further.

7.1 Limitations

The test video used did not include audio due to the stream with audio not playing correctly. While this does mean the results focus only on the video stream, avoiding any other conflicting factors, most practical applications of video streaming include at least one audio track, so its effects could be investigated. This separate track is also typically served over DASH as a separate stream, and it would be valuable to see how the satellite scenario handles two streams at the same time, one low bitrate (audio) and one high bitrate (video). Future work should investigate streams of both video and audio over a satellite.

This work does not fully establish a cause for why the 10 second and 20 second segment length streams never change quality level from the minimum. But since they were never observed to stall, it indicates that there is excess bandwidth capacity for higher bitrates. Further work should identify the cause for this issue, and resolve it if possible to truly understand DASH performance with these segment lengths.

7.2 Opportunities

We conducted six tests for each ABR algorithm and found statistically significant results indicating better performance by ABR BOLA. However, there is potential for conducting additional tests with each algorithm to gather more data. Furthermore, our tests used a segment length of 4 seconds for all ABR algorithms. Examining the impact of varying segment lengths on each algorithm's performance could deliver insightful results.

As discussed in Chapter 3, there have been many ABR algorithms proposed with different approaches to the ABR problem. Since the BOLA algorithm is shown in this paper to have significantly different performance, there is reason to believe other ABR algorithms could have similar improvements or degradations in performance. These algorithms should be investigated over a GEO satellite network.

While the three segment lengths tested in this paper represent a wide range of values, they still do not extend down to the lowest lengths recommended for typical networks [26], nor to extremely high lengths. Comparing a low segment length over a traditional network directly to a GEO satellite network could help to identify precisely how differently the two networks behave. Testing extremely high segment lengths could demonstrate if there is a "peak" of performance at a sweet spot, or if unusual behavior arises at the high end.

This work only runs a single stream at a time, but having multiple DASH streams competing for bandwidth at once has been shown to cause many performance challenges [3, 12, 22]. It is possible that some fairness schemes, such as those that require information to be transmitted across the network between agents, could struggle on a GEO satellite network, where sending even small messages can take a long time. Future work should run multiple streams at once to observe how various ABR algorithms compete over a satellite connection.

The modern QUIC transport protocol is becoming the new standard, over the long-living TCP. While QUIC provides little benefit over TCP when running through a GEO satellite network [29], it is still possible that the application of DASH will behave differently and exhibit more of a difference. Additionally, TCP has many different congestion control algorithms available, some of which are designed specifically for high-latency satellite connections. These should be compared to identify if there are any opportunities to improve performance.

Appendices

Appendix A

Appendix A - Segment Length Results

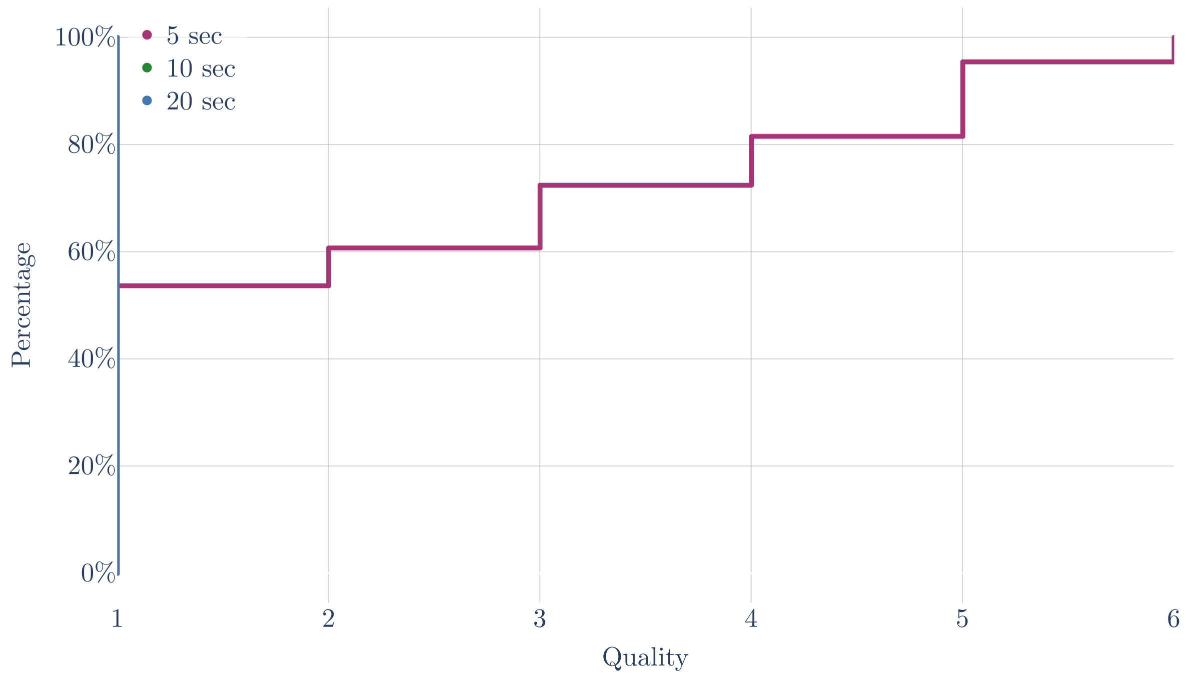


Figure A.1: **Segment length test, distributions of quality levels.** Distribution of quality levels over the course of the test.

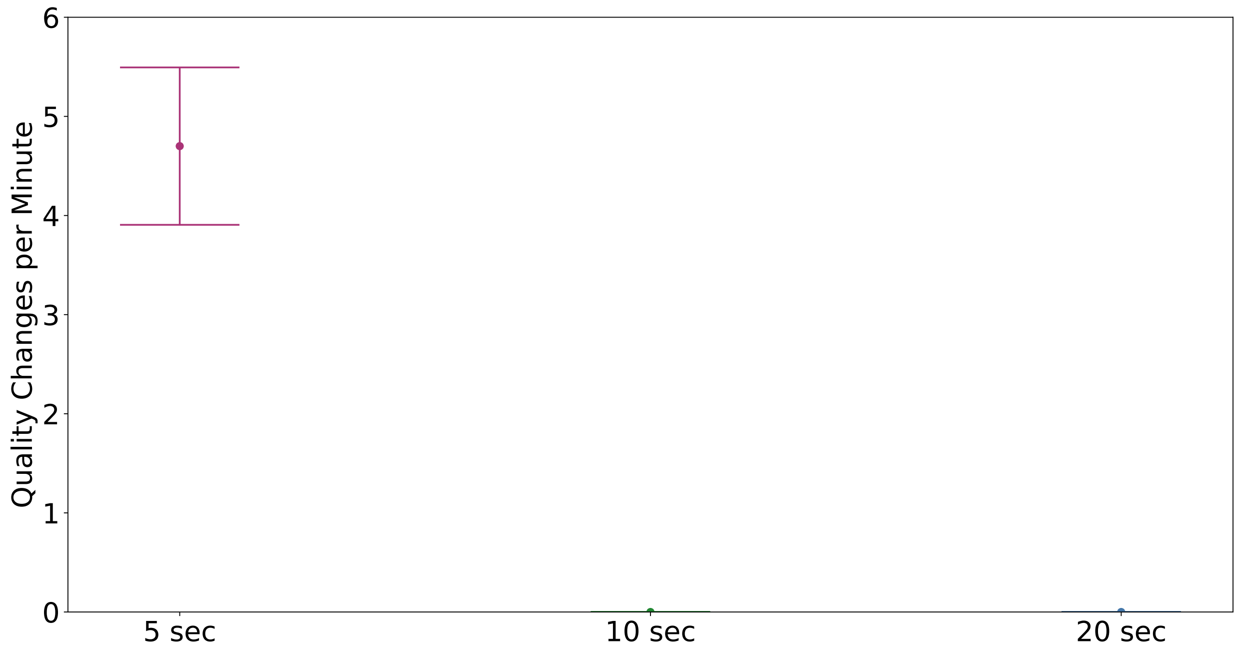


Figure A.2: **Segment length test, box plots of quality level changes per minute.** Distribution of quality level changes per minute over the course of the test.

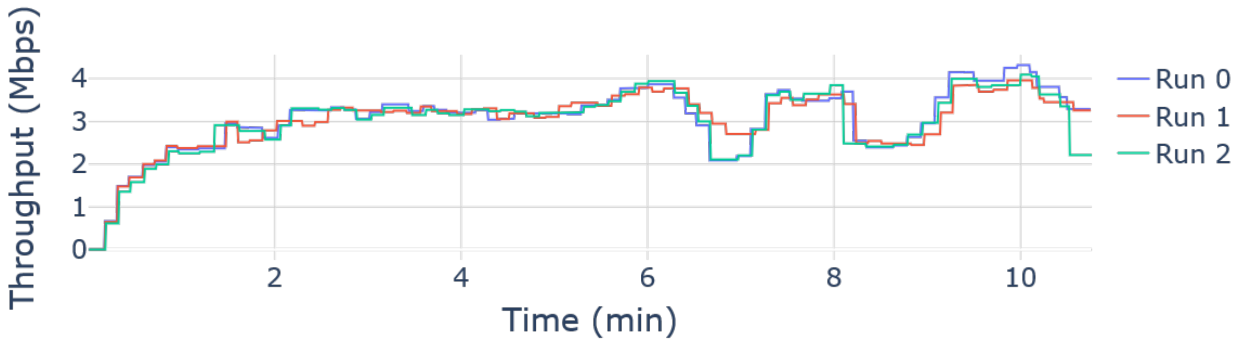


Figure A.3: **Segment length test, estimated throughput vs time.** Throughput over time, as reported by the Dash.js player's throughput estimator. 10 second segment length.

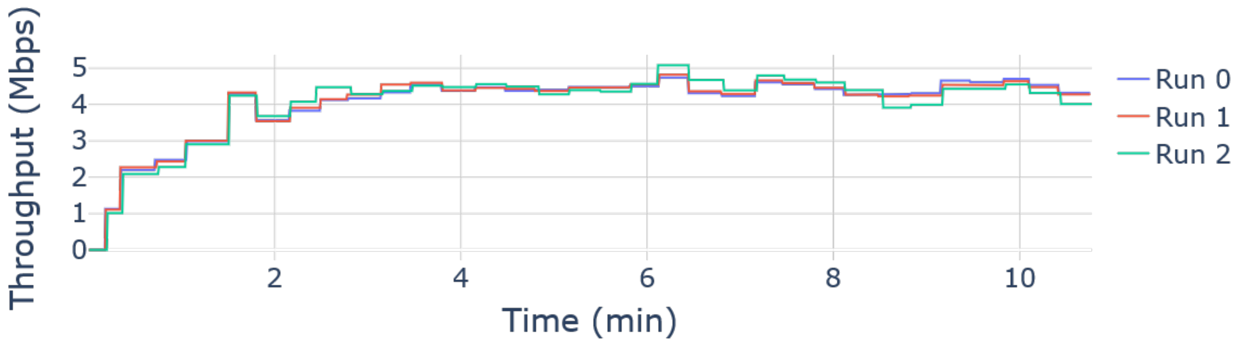


Figure A.4: **Segment length test, estimated throughput vs time.** Throughput over time, as reported by the Dash.js player's throughput estimator. 20 second segment length.

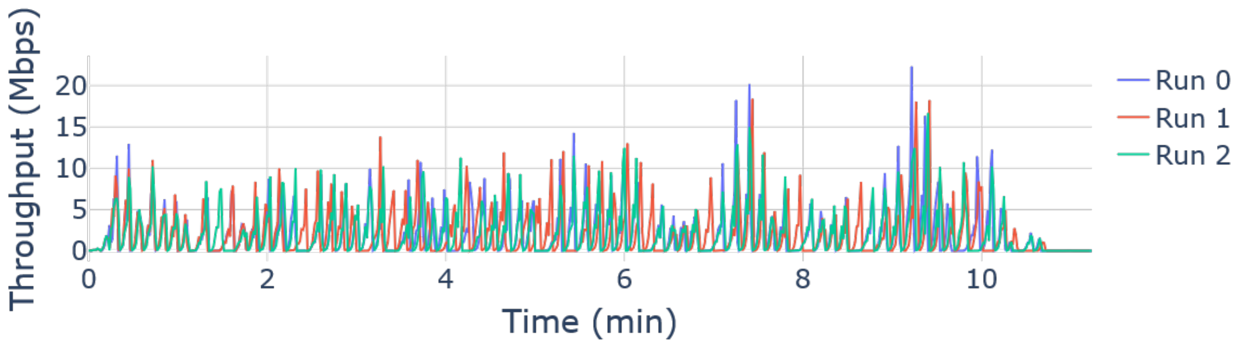


Figure A.5: **Segment length test, measured throughput vs time.** Throughput over time, as measured by the Wireshark trace on the client. 10 second segment length.

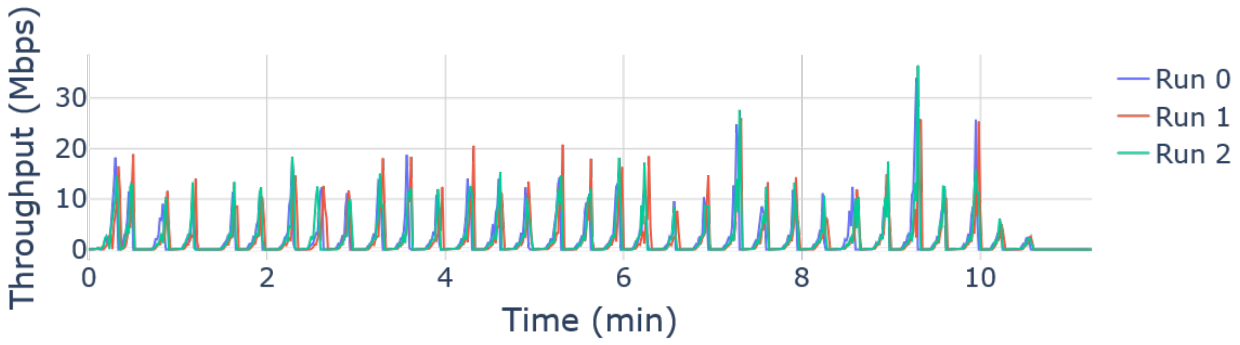


Figure A.6: **Segment length test, measured throughput vs time.** Throughput over time, as measured by the Wireshark trace on the client. 20 second segment length.

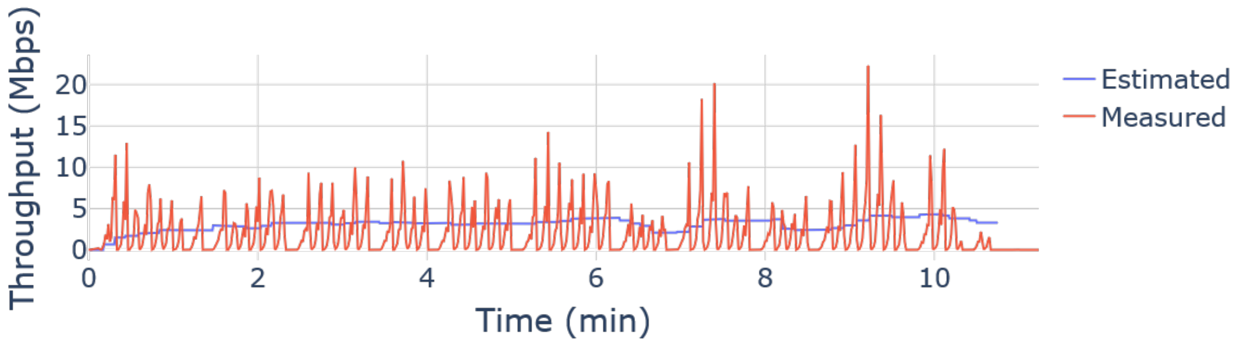


Figure A.7: **Segment length test, estimated and measured throughput vs time.** Comparing estimated throughput to observed throughput over time. 10 second segment length, first run.

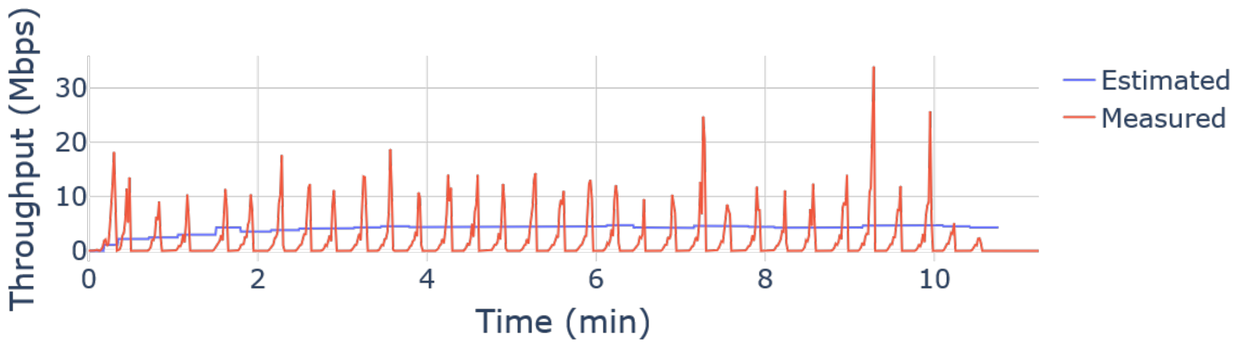


Figure A.8: **Segment length test, estimated and measured throughput vs time.** Comparing estimated throughput to observed throughput over time. 20 second segment length, first run.

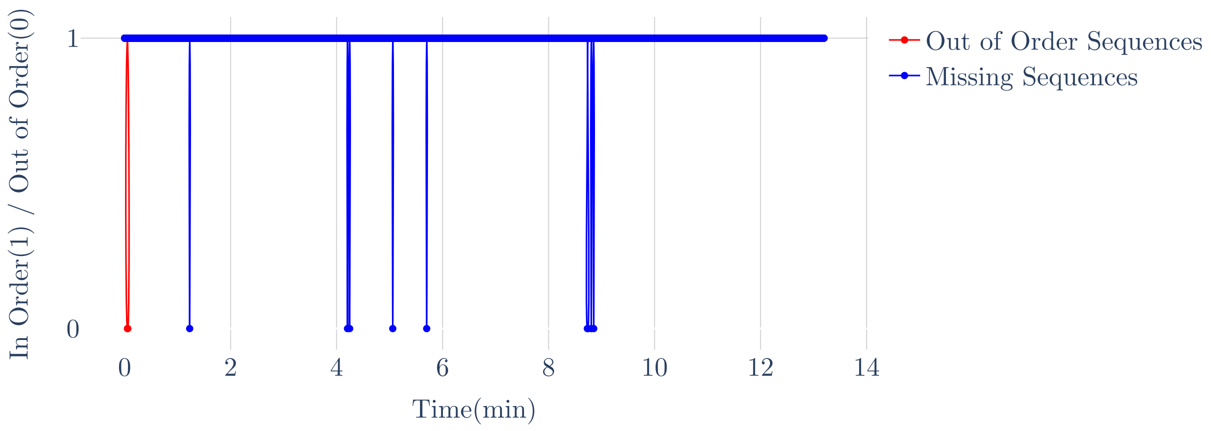


Figure A.9: **Out of order and missing packets. Packet status vs time.** 5 second segment length.

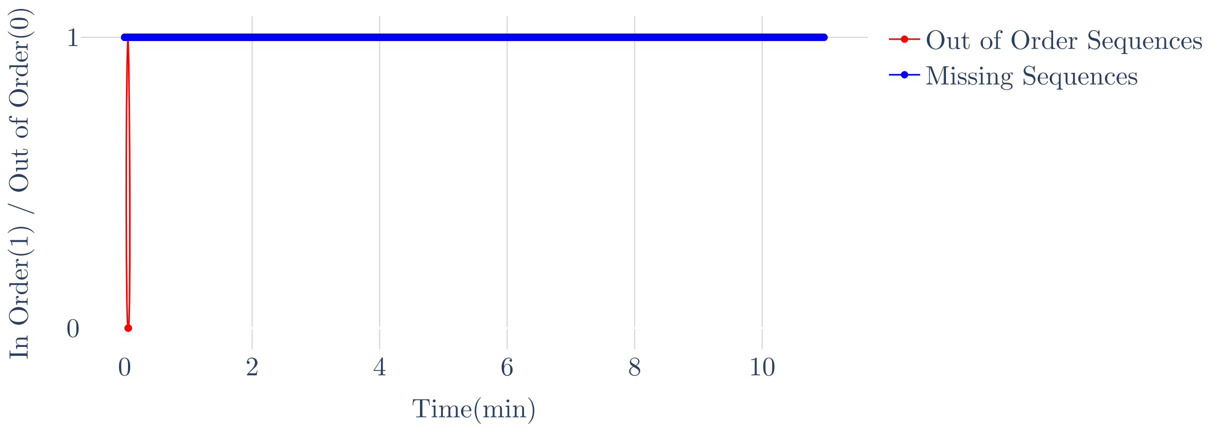


Figure A.10: **Out of order and missing packets. Packet status vs time.** 10 second segment length.

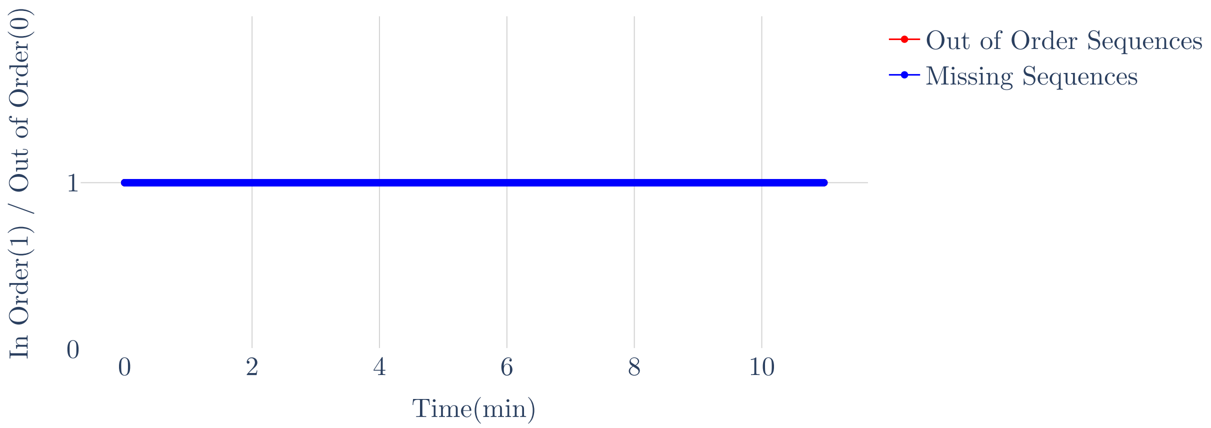


Figure A.11: **Out of order and missing packets. Packet status vs time.** 20 second segment length.

Appendix B

Appendix B - ABR Algorithm Results

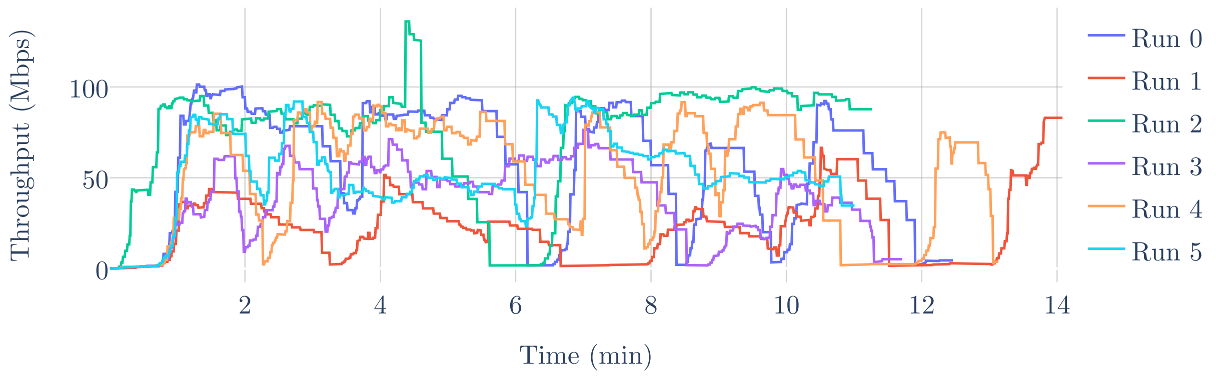


Figure B.1: Throughput over time, as reported by the Dash.js player's throughput estimator. "Dynamic" ABR algorithm.

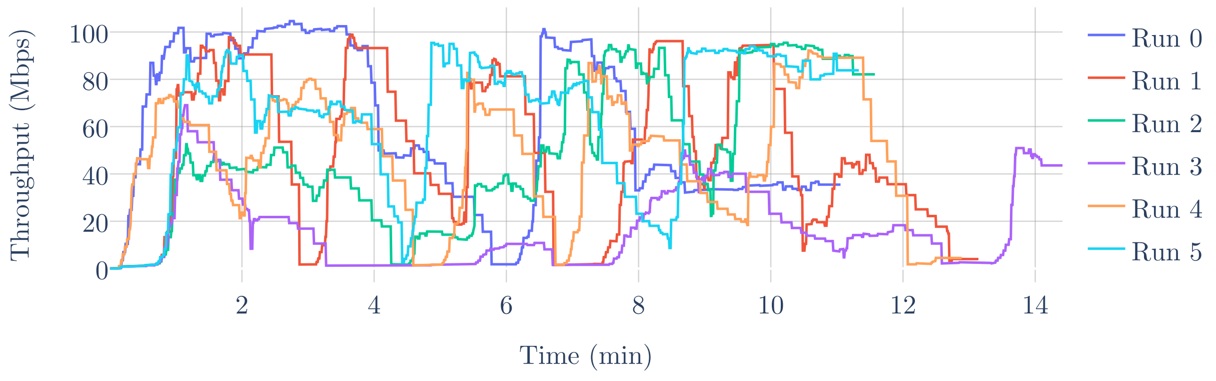


Figure B.2: Throughput over time, as reported by the Dash.js player's throughput estimator. "Throughput" ABR algorithm.

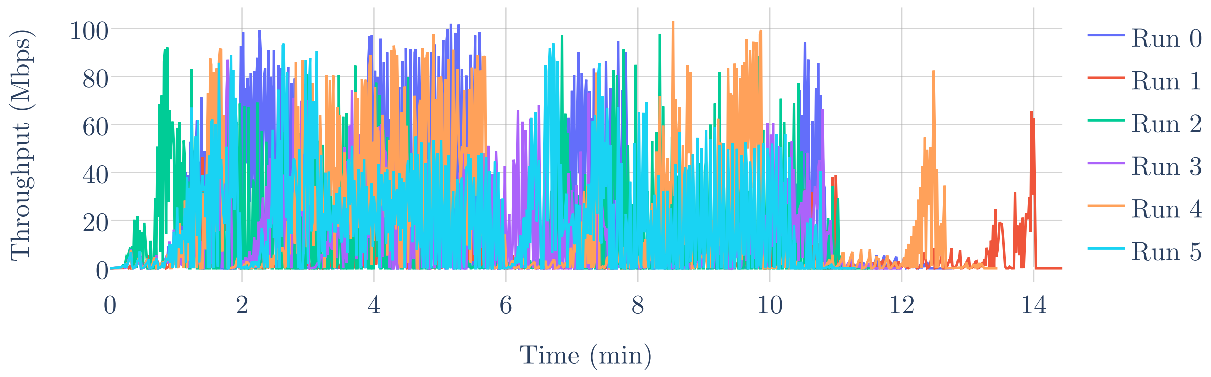


Figure B.3: Throughput over time, as measured by the Wireshark trace on the client. “Dynamic” ABR algorithm.

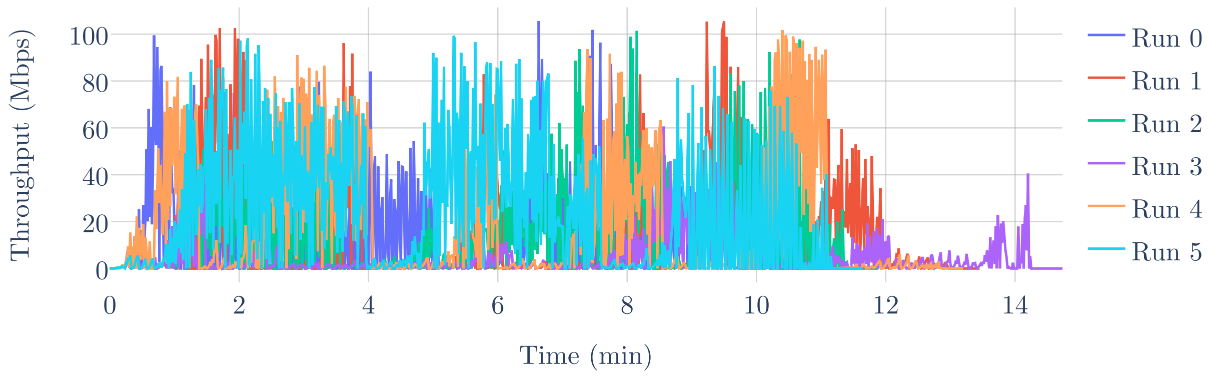


Figure B.4: Throughput over time, as measured by the Wireshark trace on the client. “Throughput” ABR algorithm.

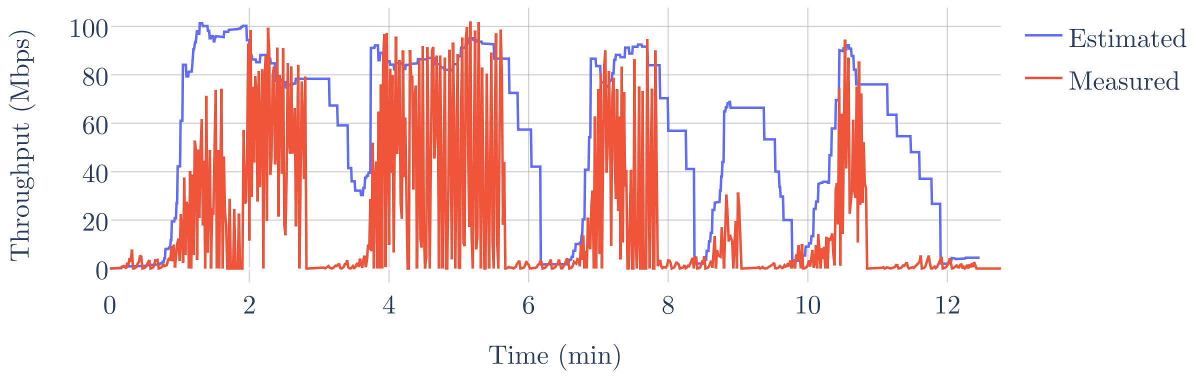


Figure B.5: Comparing estimated throughput to observed throughput over time. “Dynamic” ABR algorithm, first run.

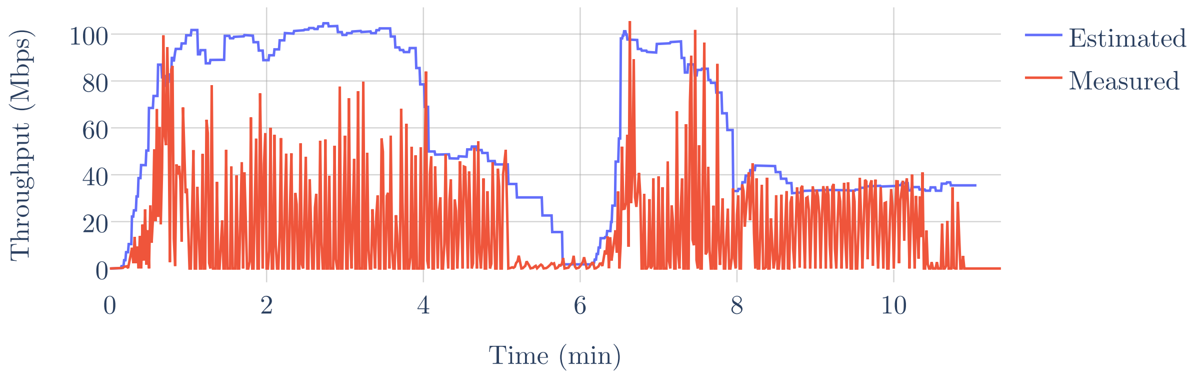
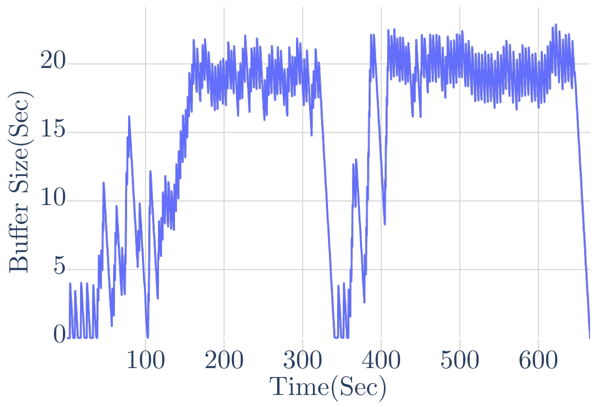
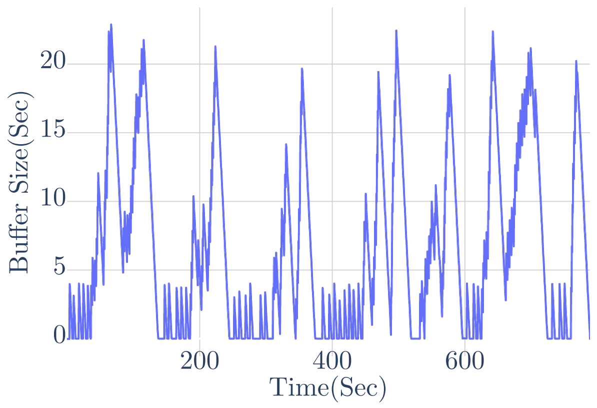


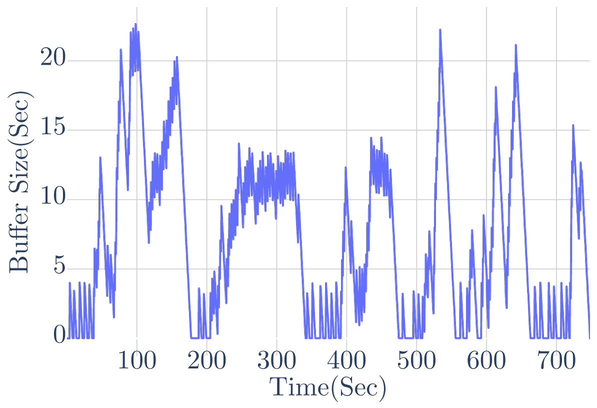
Figure B.6: Comparing estimated throughput to observed throughput over time. “Throughput” ABR algorithm, first run.



(a) BOLA ABR Algorithm: Buffer Level Evolution

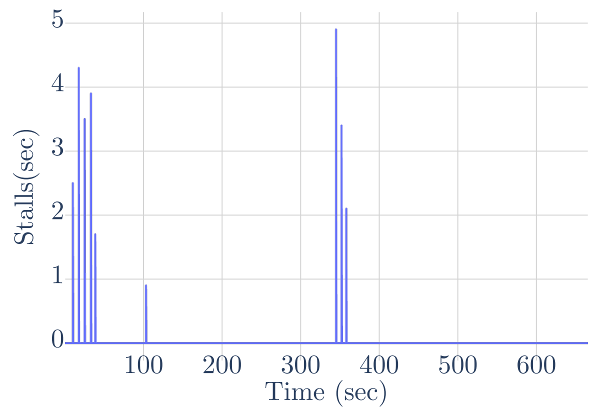


(b) Throughput-based ABR Algorithm: Buffer Level Evolution

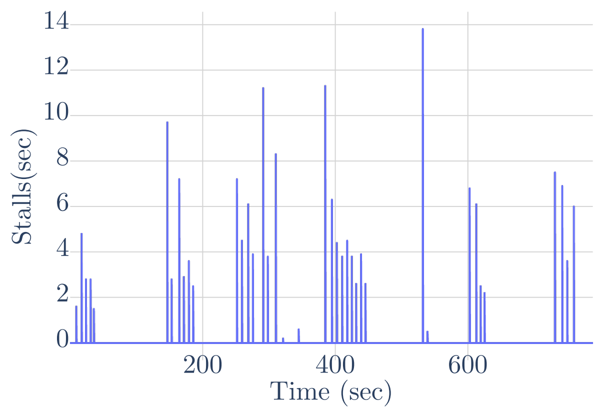


(c) Dynamic ABR Algorithm: Buffer Level Evolution

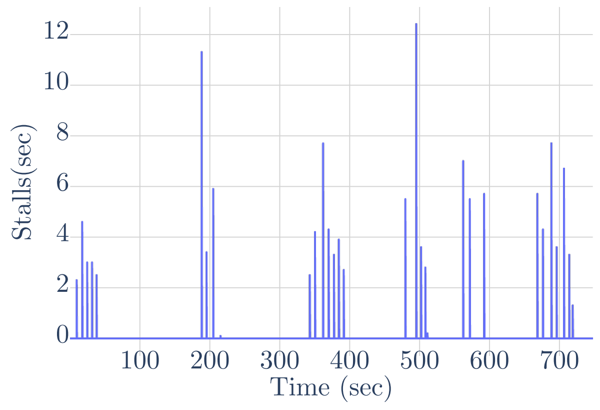
Figure B.7: Comparison of Buffer Level Evolution for ABR BOLA,Throughput and Dynamic



(a) BOLA ABR Algorithm: Stall Duration Evolution

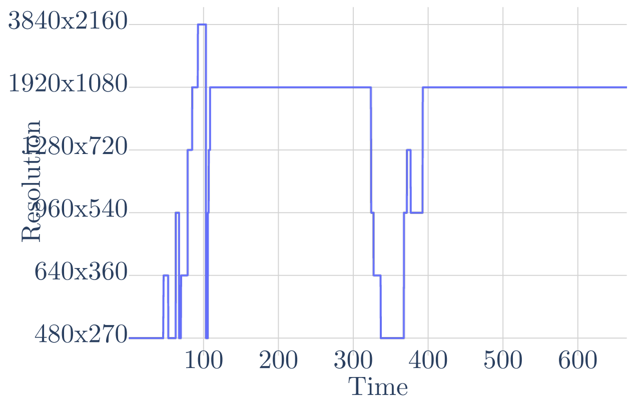


(b) Throughput-based ABR Algorithm: Stall Duration Evolution

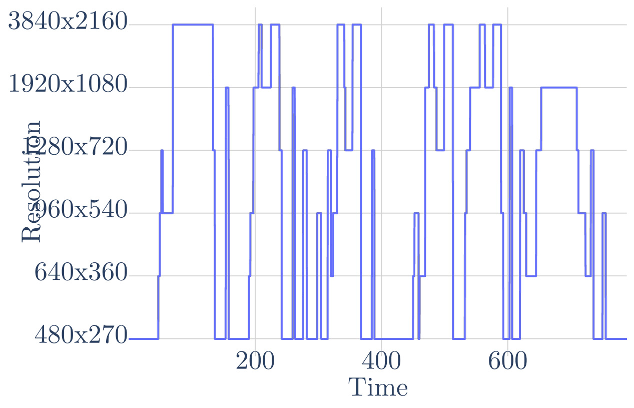


(c) Dynamic ABR Algorithm: Stall Duration Evolution

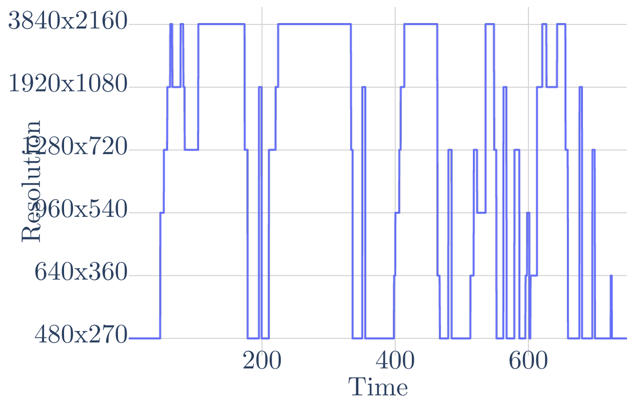
Figure B.8: Comparison of Stall Duration Evolution for ABR BOLA,Throughput and Dynamic



(a) BOLA ABR Algorithm: Resolution Evolution

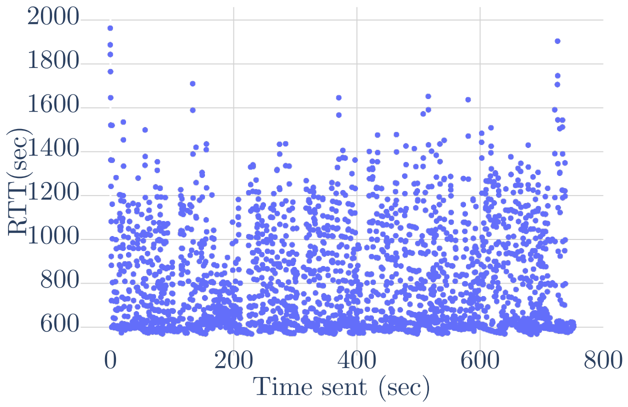


(b) Throughput-based ABR Algorithm: Resolution Evolution

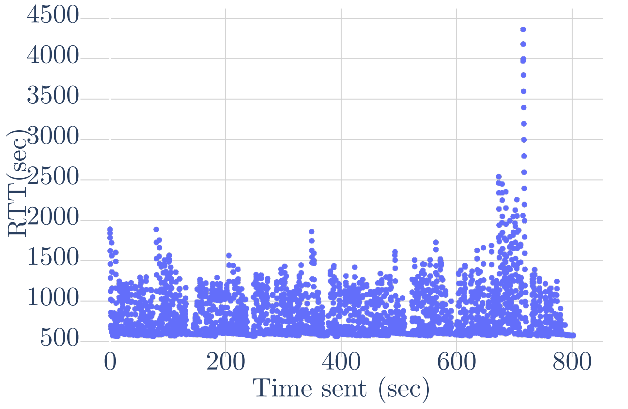


(c) Dynamic ABR Algorithm: Resolution Evolution

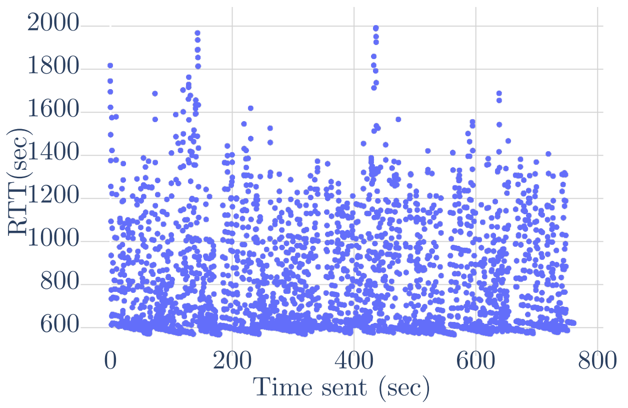
Figure B.9: Comparison of Resolution Evolution for ABR BOLA, Throughput, and Dynamic Algorithms



(a) BOLA ABR Algorithm: RTT Evolution



(b) Throughput-based ABR Algorithm: RTT Evolution



(c) Dynamic ABR Algorithm: RTT Evolution

Figure B.10: Comparison of RTT Evolution for ABR BOLA, Throughput, and Dynamic Algorithms

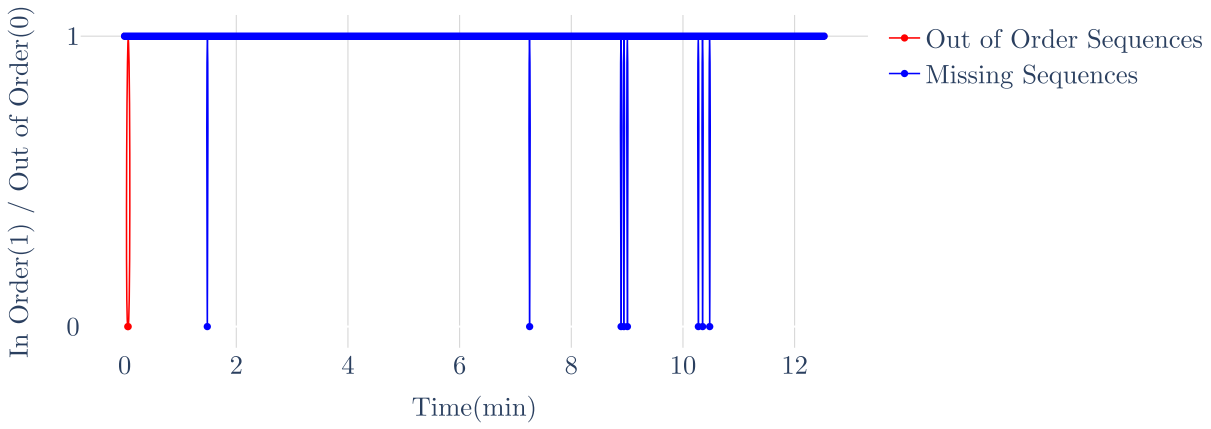


Figure B.11: **Out of order and missing packets. Packet status vs time.** BOLA.

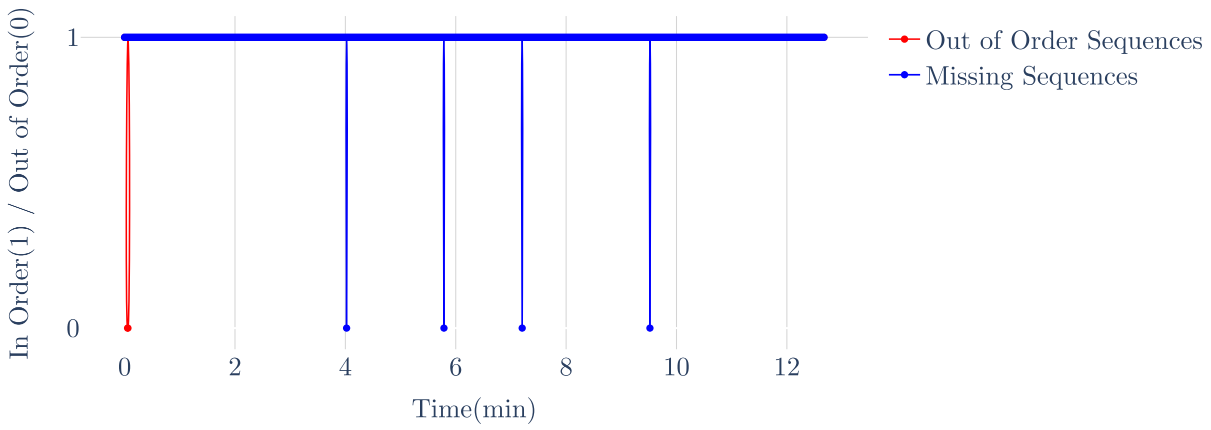


Figure B.12: **Out of order and missing packets. Packet status vs time.** Dynamic.

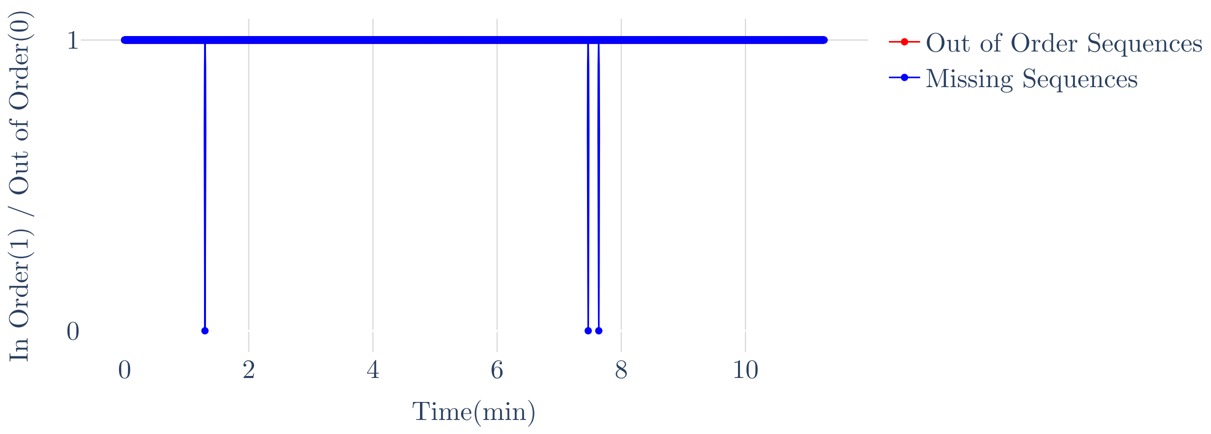


Figure B.13: **Out of order and missing packets. Packet status vs time.** Throughput.

Bibliography

[1] *An overview of HTTP*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[2] Abdelhak Bentaleb, Ali C. Begen, and Roger Zimmermann. "SDNDASH: Improving QoE of HTTP Adaptive Streaming Using Software Defined Networking". In: *Proceedings of the 24th ACM International Conference on Multimedia*. MM '16. Amsterdam, The Netherlands: Association for Computing Machinery, 2016, pp. 1296–1305. ISBN: 9781450336031. DOI: 10.1145/2964284.2964332. URL: https://doi.org/10.1145/2964284.2964332.

[3] Abdelhak Bentaleb et al. "Want to play DASH?: a game theoretic approach for adaptive streaming over HTTP". In: June 2018, pp. 13–26. DOI: 10.1145/3204949.3204961.

[4] Bradley Biketi. *Comparison between the HTTP/3 and HTTP/2 Protocols*. 2021. URL: https://www.section.io/engineering-education/http3-vs-http2/.

[5] Bitmovin. *Bitmovin Test Player*. URL: https://bitmovin.com/demos/stream-test.

[6] Jon Brodkin. "Satellite Internet faster than advertised, but latency still awful". In: (Feb. 2013). URL: https://arstechnica.com/information-technology/2013/02/satellite-internet-faster-than-advertised-but-latency-still-awful/#p3n.

[7] Saahil Claypool, Jae Chung, and Mark Claypool. "Comparison of TCP Congestion Control Performance over a Satellite Network". In: Mar. 2021, pp. 499–512. ISBN: 978-3-030-72581-5. DOI: 10.1007/978-3-030-72582-2_29.

[8] *Comparison of the usage statistics of HTTP/2 vs. HTTP/3 for websites*. Accessed: 2023-04-19. URL: https://w3techs.com/technologies/comparison/ce-http2,ce-http3.

[9] DASH-IF. *DASH-IF Reference Client v4.5.0*. URL: https://reference.dashif.org/dash.js/v4.5.0/samples/dash-if-reference-player/index.html.

[10] DASH-IF. *Dash.js Documentation: ABR Logic*. URL: https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic.

[11] *dash.js Module: Settings*. URL: http://cdn.dashjs.org/latest/jsdoc/module-Settings.html.

[12] Luca De Cicco et al. "ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH)". In: *2013 20th International Packet Video Workshop*. 2013, pp. 1–8. DOI: 10.1109/PV.2013.6691442.

[13] Jörg Deutschmann, Kai-Steffen Hielscher, and Reinhard German. "Satellite Internet Performance Measurements". In: *2019 International Conference on Networked Systems (NetSys)*. 2019, pp. 1–4. DOI: 10.1109/NetSys.2019.8854494.

[14] *Evolution of HTTP*. Accessed: 2023-04-19. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP.

[15] Blender Foundation and Janus B. Kristensen. *Big Buck Bunny Sunflower Version*. 2013. URL: http://bbb3d.renderfarming.net/download.html.

[16] Muhammad Furqan and Bhargavi Goswami. "Satellite Communication Networks". In: Jan. 2022, pp. 1–22. ISBN: 978-981-4585-87-3. DOI: 10.1007/978-981-4585-87-3_70-1.

[17] Thilina Ashen Gamage. *Evolution of HTTP — HTTP/0.9, HTTP/1.0, HTTP/1.1, Keep-Alive, Upgrade, and HTTPS*. 2017. URL: https://medium.com/platform-engineer/evolution-of-http-69cfe6531ba0.

[18] Kevin Graham. *Adaptive Bitrate Streaming: What it Is and How the ABR Algorithm Works [2022 Update]*. June 2022. URL: https://www.dacast.com/blog/adaptive-bitrate-streaming/.

[19] Sangtae Ha, Injong Rhee, and Lisong Xu. "CUBIC: A New TCP-Friendly High-Speed TCP Variant". In: *SIGOPS Oper. Syst. Rev.* 42.5 (July 2008), pp. 64–74. ISSN: 0163-5980. DOI: 10.1145/1400097.1400105. URL: https://doi.org/10.1145/1400097.1400105.

[20] *Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats.* Standard. Geneva, CH: International Organization for Standardization, Mar. 2022.

[21] V. Jacobson and R. Braden. *TCP extensions for long-delay paths.* RFC 1072. Oct. 1988. DOI: 10.17487/RFC1072. URL: https://www.rfc-editor.org/info/rfc1072.

[22] Junchen Jiang, Vyas Sekar, and Hui Zhang. "Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive". In: *IEEE/ACM Trans. Netw.* 22.1 (Feb. 2014), pp. 326–340. ISSN: 1063-6692.

[23] Maryam Ataei Kachooei et al. "Fixing TCP Slow Start for Slow Fat Links". In: Oct. 2022. URL: https://web.cs.wpi.edu/~claypool/papers/tcp-best-netdev-22/paper.pdf.

[24] J.F. Kurose and K.W. Ross. *Computer Networking: A Top-down Approach.* Pearson, 2021. ISBN: 9781292405469. URL: https://books.google.com/books?id=g8R5zgEACAAJ.

[25] Adam Langley et al. "The QUIC Transport Protocol: Design and Internet-Scale Deployment". In: 2017.

[26] Stefan Lederer. *Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length.* Apr. 9, 2020. URL: https://bitmovin.com/mpeg-dash-hls-segment-length/.

[27] Stefan Lederer, Christopher Müller, and Christian Timmerer. "Dynamic Adaptive Streaming over HTTP Dataset". In: *Proceedings of the 3rd Multimedia Systems Conference.* MMSys '12. Chapel Hill, North Carolina: Association for Computing Machinery, 2012, pp. 89–94. ISBN: 9781450311311. DOI: 10.1145/2155555.2155570. URL: https://doi.org/10.1145/2155555.2155570.

[28] Mingxi Liu et al. "The Effects of a Performance Enhancing Proxy on TCP Congestion Control over a Satellite Network". In: Nov. 2022, pp. 325–331. DOI: 10.1109/IPCCC55026.2022.9894351.

[29] Matthew Malone and Noah Preston Capucilli-Shatan. *Performance Analysis of QUIC vs TCP on Satellite Networks.* Tech. rep. 2022. URL: https://web.cs.wpi.edu/~claypool/mqp/satellite-21/final-report.pdf.

[30] 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.* SIGCOMM '17. Los Angeles, CA, USA: Association for Computing Machinery, 2017, pp. 197–210. ISBN: 9781450346535. DOI: 10.1145/3098822.3098843. URL: https://doi.org/10.1145/3098822.3098843.

[31] Hiroyasu Obata, Kazuya Tamehiro, and Kenji Ishida. "Experimental evaluation of TCP-STAR for satellite Internet over WINDS". In: Apr. 2011, pp. 605–610. DOI: 10.1109/ISADS.2011.86.

[32] Hiroyasu Obata et al. "TCP-STAR: TCP Congestion Control Method for Satellite Internet". In: (Jan. 2006). DOI: 10.1093/ietcom/e89-b.6.1766.

[33] Jan Ozer. *Choosing the Segment Duration for DASH or HLS.* Aug. 4, 2016. URL: https://streaminglearningcenter.com/learning/choosing-the-optimal-segment-duration.html.

[34] Télécom Paris. *GPAC.* URL: https://gpac.wp.imt.fr/.

[35] FFmpeg Project. *FFmpeg.* URL: https://ffmpeg.org/about.html.

[36] Michael Seufert et al. "A Survey on Quality of Experience of HTTP Adaptive Streaming". In: *IEEE Communications Surveys & Tutorials* 17.1 (2015), pp. 469–492. DOI: 10.1109/COMST.2014.2360940.

[37] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. "From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 15.2s (July 2019). ISSN: 1551-6857. DOI: 10.1145/3336497. URL: https://doi.org/10.1145/3336497.

[38] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. “BOLA: Near-Optimal Bitrate Adaptation for Online Videos”. In: *IEEE/ACM Transactions on Networking* 28.4 (2020), pp. 1698–1711. DOI: 10.1109/TNET.2020.2996964.

[39] Streamroot. *Streamroot DASH test video manifest file.* URL: https://dash.akamaized.net/akamai/streamroot/050714/Spring_4Ktest.mpd.

[40] *Test Script GitHub URL.* URL: https://github.com/WPISatStreamMQP/ModifiedClientCode/.

[41] Linus Torvalds. *Linux tcp.h.* URL: https://github.com/torvalds/linux/blob/master/include/net/tcp.h.

[42] Krishna Rao Vijayanagar. *What’s ABR Streaming and How Does ABR Work?* July 2020. URL: https://ottverse.com/what-is-abr-video-streaming/.

[43] Lawrence Williams. *TCP 3-Way Handshake (SYN, SYN-ACK,ACK).* 2023. URL: https://www.guru99.com/tcp-3-way-handshake.html.

[44] Huahui Wu. *UDPing.* URL: http://perform.wpi.edu/tools/.

[45] Xiaokun Xu and Mark Claypool. *Measurement of Cloud-based Game Streaming Systems Competing with DASH Flows, WPI-CS-TR-23-02.* Tech. rep. 2023. URL: https://ftp.cs.wpi.edu/pub/techreports/pdf/23-02.pdf.

[46] Praveen Kumar Yadav et al. “Playing Chunk-Transferred DASH Segments at Low Latency with QLive”. In: *Proceedings of the 12th ACM Multimedia Systems Conference.* MMSys ’21. Istanbul, Turkey: Association for Computing Machinery, 2021, pp. 51–64. ISBN: 9781450384346. DOI: 10.1145/3458305.3463376. URL: https://doi.org/10.1145/3458305.3463376.

[47] Xiaoqi Yin et al. “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.* SIGCOMM ’15. London, United Kingdom: Association for Computing Machinery, 2015, pp. 325–338. ISBN: 9781450335423. DOI: 10.1145/2785956.2787486. URL: https://doi.org/10.1145/2785956.2787486.

[48] Chao Zhou et al. *TFDASH: A Fairness, Stability, and Efficiency Aware Rate Control Approach for Multiple Clients over DASH.* 2017. arXiv: 1704.08535 [cs.MM].