University of Saarland
Telecommunications Lab
Intel Visual Computing Institute

Multipath Adaptive Video Streaming over Multipath TCP

Master's Thesis in Computer and Communication Technology
by

Yashavanth Puttaswamy Gowda Chowrikoppalu

Supervisor

Prof. Dr.-Ing. Thorsten Herfet

Advisor

Goran Petrovic M.Sc.

Reviewers

Prof. Dr.-Ing. Thorsten Herfet

Prof. Dr.-Ing. Philipp Slusallek

March 2013

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Ich erkläre hiermit an Eides Statt, dass die vorliegende Arbeit mit der elektronischen Version übereinstimmt.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis. I hereby confirm the congruence of the contents of the printed data and the electronic version of the thesis.

Saarbrücken, den 27. März 2013,

(Yashavanth Puttaswamy Gowda Chowrikoppalu)

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlichtwird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, den 27. März 2013,

(Yashavanth Puttaswamy Gowda Chowrikoppalu)

For their unconditional love and support, I owe this thesis to my parents and my brother.

- Yash

Abstract

Multimedia streaming over the Internet has become popular over the years and expected to constitute 90% of Internet traffic in the near future. However, state-of-the-art solutions for multimedia streaming are inherently limited by their use of a single network path and a lack of efficient adaptation algorithms. Dynamic Adaptive Streaming over HTTP (DASH) is a new streaming technology that uses streaming-rate adaptation to accommodate bandwidth variations and utilize the available bandwidth more efficiently. Multipath TCP (MPTCP) is a new transport protocol that enables the use of multiple paths for data transfer. In this thesis, we design and implement a framework for adaptive multimedia streaming using MPTCP, which in many scenarios provides a better streaming performance and robustness to path failures. At the core of our solution are efficient rate-adaptation algorithms that are applicable to both MPTCP and TCP streaming. With these algorithms, the performance of adaptive MPTCP streaming is evaluated under different network conditions to demonstrate the suitability of the proposed framework.

Acknowledgements

First and foremost, I would like to thank my advisor Goran Petrovic for his invaluable guidance. I feel deeply indebted to his strong advises, encouragements and instructions. He has not only assisted me in completing this thesis, they have also helped me to broaden my attitude towards research, and to develop my personality.

A special note of thanks to Prof. Thorsten Herfet for giving me the opportunity to pursue this thesis under his supervision and for his valuable inputs.

I am highly grateful to Prof. Philipp Slusallek for reviewing my thesis. I would like to thank Prof. Herfet for his inspiring personality which has prepared a very lively research environment at the Telecommunications Lab that allows for plenty of growth and learning. I would like to take this opportunity to also thank all the members of the department.

A special note of thanks to Zakaria and Manuel for their technical help on different occasions. Thanks to Manjunath, Mittul, Pooja, Manish, Bernd, Jochen, Tobias, Himangshu, Mahendiran and everyone else for their untiring support and being there during the crunch times. I am also very grateful to my friends and classmates for their company and moral support. I learnt a lot from them in last 2 years which is a very important factor for successful completion of my thesis.

Contents

Chapter 1

Introduction

In the beginning, the Internet was primarily designed and used for the textual data transmission. But, today the Internet is also being used for transmission of other types of data such as multimedia, telephony etc. Over the recent years, the multimedia transmission has become dominant as more and more users are watching multimedia over the Internet. It is estimated that multimedia transmission will account for more than 90% of the Internet traffic [5] in the next few years. This change of Internet usage can be accounted for the following reasons : (1) the availability of Internet Protocol Television (IPTV) services and open platforms like Youtube[1], enabled the users to share and watch the multimedia content globally. (2) New networking technologies like Content Distribution Networks (CDN's) enabled the efficient delivery of multimedia content to the end users over the dedicated networks. (3) Advancements in multimedia technologies like High Definition Television (HDTV), 3-D television and stereoscopic displays encouraged users to access and experience these technologies over the Internet. Even though the present Internet is being used for multimedia transmission, the long term support by the Internet is still an open question.

To begin with, in this chapter we introduce the state-of-the-art multimedia streaming and its limitations. Next, we define the scope of this thesis and discuss the related work. An overview of our contributions to overcome the multimedia streaming limitations is given in the end.

[1] http://www.youtube.com/

1.1 State-of-the-art

1.1.1 Multimedia Streaming

In general, 'Multimedia' represents a convergence of text, audio, still images, animation and video to a single form. It is used in many application domains such as advertising, education and entertainment. Multimedia streaming (transmission) over the Internet was started during the early 1990's and it represents the transmission of multimedia data over the Internet in compressed form, from a streaming sender to a streaming receiver and displaying it to the user at the receiver end.

Multimedia streaming differs from the textual data transfer in that it can tolerate certain amount of data loss and requires a real-time delivery (strict delay constraint) of the content. In addition to the usual streaming between a sender and a single receiver (unicast), scenarios with multiple clients need to be supported (multicast). To enable this, different multimedia transport protocols, coding, representation and transmission formats have been defined and developed over time. Diversity of these formats results in different types of media streaming systems with no standard solution. In general, media streaming systems can be classified based on the type of content being transferred, method of access, control algorithms (adaptation) and the underlying network protocols.

The main components of a multimedia streaming system are the encoder, streaming server, streaming client, media transfer protocol and the underlying physical network. Such a streaming system is shown in Figure 1.1.

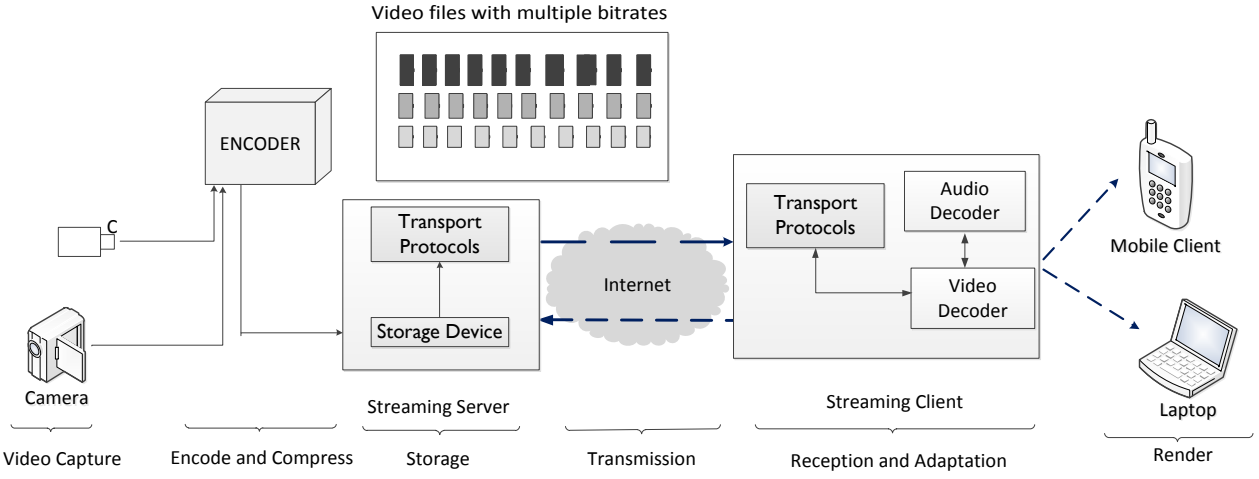


Figure 1.1: General multimedia streaming system.

Multimedia streaming system typically performs the functions such as capturing, encoding, storage, transmission, reception, decoding and rendering. Initially the camera captures real world events and produces the media content either as still images or a sequence of images (video). In some cases, output from the camera is in raw media format without any compression. This raw media cannot be transferred over the network as it requires a very large bandwidth because of its large size. To prepare the media for transmission, media is compressed using appropriate compression standards (e.g., MPEG) at the sending end. The compressed data are stored at the sending side (server) either as a single file or a set of files. The sending side also holds media-description files, which may contain various meta-data such as the location and timing information of the media files available on the server. Sender transmits the description files and media files when requested by the receiver. On the receiver end (client), the media is received in the form of packets and reassembled to the original compressed stream. Compressed stream is then input to the decoder, which decodes the media and subsequently passed on to the renderer for display. Generally, the above sequence of events represents the minimal set of operations performed by the streaming system.

1.1.1.1 Live and Stored Media Streaming

Based on whether the media is a live content or generated offline, streaming can be classified as live streaming and stored streaming. Live streaming has strict time delay constraints since capture, encoding, transmission, decoding and display are performed in real-time and no additional time delay is available for feedback and retransmission. Under such strict time delay constraints, if the data is delayed or corrupted, no corrective action could be performed. But in case of stored streaming, since data is stored on the sender (server), it can be retransmitted.

1.1.1.2 Adaptive and Non-adaptive Streaming

Adaptation refers to automatically choosing different encoded versions at different times based on the client's capabilities, the available network bandwidth and buffer conditions. Based on the presence (on the client and/or server side) or absence of adaptation mechanisms, video streaming can be categorized as adaptive or non-adaptive.

Non-adaptive streaming. In this mode, a single encoded version of the video is used. Therefore, no flexibility is available in automatically choosing among different versions of the media. As a result, the streamed content cannot be adapted to available conditions. In turn, if the user device is not having enough bandwidth, CPU resources or memory, the user experiences playout interruptions in the session. However, in some non-adaptive

systems, the user is provided with multiple versions and can choose different video qualities manually.

Adaptive streaming. In adaptive streaming, multiple versions of the media are available at the sender. In each version, the media has been partitioned into constituent elements called segments. This partitioning is performed along the time axis. Particular segments of the representation (version) can be selected automatically in response to instantaneous transmission conditions. The segments are typically of short duration (in the order of seconds) in order to support a quick adaptation to time-varying conditions. Ideally, the adaptation is performed in an optimized fashion such that the client always chooses the "best" segment based on variety of parameters based on the instantaneous availability of the resources such as the available bandwidth, display resolution, CPU power, state of the playout buffer, etc.

1.1.1.3 Transport Protocols in Media Streaming

Even though the present Internet is used for different kinds of data, the ability to support efficient transmission of different data types has remained a challenge till today. Many transport protocols have been proposed over time, some generic in their support for a variety of data types (e.g., TCP) and others specialized for certain kind of data (e.g., transmission of multimedia data). Multimedia transmission is characterized by the need to transfer large amounts of data and a real-time data consumption. To support this kind of transmission, specialized protocols for efficient broadcast, multicast and unicast have been proposed over the years, including RTP/RTCP [15], TFRC [23] and PRRT [13]. However, many of these specialized protocols are not yet deployed on a larger scale due to their novelty or inherent limitations. For example, RTP/RTCP works well in managed networks with its own packet structure and session management capabilities. But RTP has its own drawbacks. For example, RTP manages each streaming session on the sender, resulting in a resource intensive and non-scalable system. On the other hand, novel transport protocols such as PRRT may become widely deployed in the future.

Although TCP is a general-purpose transport protocol and widely used for file transfers, its suitability for efficient transport of multimedia data is still an open question. However, multimedia streaming over TCP in general and HTTP/TCP-based streaming in particular, have recently become popular due to the easiness of deployment in the current Internet. HTTP and TCP are foundation protocols of World Wide Web and are part of the protocol stack of each Internet-connected device. Further, they simplify connectivity, since a HTTP/TCP flow can traverse network firewalls and Network Address Translators (NATs) [35]. Finally, HTTP/TCP streaming can be managed without the need to maintain

a session state on the server, thus improving the system scalability. In this way, a large-scale deployment of a streaming system is made possible by making use of the already available Web infrastructure, including HTTP servers, proxies and caches.

1.1.2 Limitations of the State-of-the-art Streaming Technologies

Despite the above advantages, today's TCP-based streaming technologies have inherent limitations in supporting streaming applications. In our view, two majors limitations are the performance of adaptive streaming algorithms and the lack of support for multi-path streaming.

Performance of adaptive streaming algorithms. Previous studies on streaming using TCP have shown that TCP requires twice the bitrate of the video for uninterrupted streaming [37]. With HTTP/TCP becoming a popular protocol stack for large-scale video delivery, there is a number of proposals on adaptive HTTP/TCP video streaming. Several commercial video-streaming players employ adaptation [2] to improve streaming efficiency and a standardization in this area is ongoing [34]. However, a recent performance analysis suggests that current rate-adaptation algorithm leaves much room for improvement [2]. Previous studies on adaptive streaming have shown that the present day solutions achieve satisfactory performance only in specific environments [7]. For instance, Microsoft Smooth Streaming is very conservative when switching between available video versions and adapts slowly to the network conditions. As a result, it does not perform well under highly varying network conditions. Likewise, Adobe HTTP Streaming does not consider playout smoothness while adapting. Apple HTTP Live Streaming has been shown to have limitations in buffer-limited systems. Finally, a recent evaluation of the adaptation algorithm discussed in the context of the DASH standard shows the following limitations [7]:

1. Adaptation is done based only on measured bandwidth and does not consider the fullness of the playout buffer.
2. Switching between different video versions does not consider the impact of varying the video quality on user experience.
3. Erroneous and inefficient bandwidth estimates are common.

Lack of support for multi-path streaming. In many cases, the Internet provides multiple paths between endpoints. The use of multiple paths in multimedia streaming is attractive for several reasons. First, the multiple available paths can be used concurrently for streaming, thus improving the available bandwidth and streaming performance. Second,

the use of multiple paths may increase the robustness to failures in that the streaming can continue on other active paths after a path goes down. Third, a streaming application can perform load balancing of streaming data over multiple paths in order to improve network stability and performance. However, by design, TCP does not support the use of multiple paths. This limits TCP-based streaming applications to benefit from using multiple paths in the Internet.

1.2 Thesis Scope and Related Work

The focus of this thesis is on overcoming the above limitations by providing a streaming solution that: (1) employs an efficient streaming adaptation (2) exploits the availability of multiple paths in the Internet. Specifically, we follow the two main lines of work. First, to provide support for multi-path streaming, we consider a streaming system that employs MPTCP, a recently-proposed multi-path transport protocol [4]. Second, we implement algorithms for adaptive streaming over MPTCP. Our starting point for adaptive streaming is the open-source implementation of the DASH standard, available as an extension of the popular Videolan player (vlc-DASH plugin). We extend this implementation with an efficient streaming algorithm.

1.2.1 Related Work

To the best of our knowledge, the use of MPTCP for adaptive video streaming has not been studied before. Our work builds on previous efforts in the areas of multipath video streaming, adaptive video streaming and multipath data transport.

A number of research proposals have investigated multi-path data transport, but not specific to video streaming. Concurrent multipath transfer over SCTP [19] implements a multi-path congestion control by relying on independent TCP Reno congestion controls on each path. As such, it does not consider the bottleneck fairness. Multipath transport on layer 3 was considered in the context of network-layer protocols like shim6 [12], HIP [31] and ECMP [16]. However, as these solutions consider network layer in isolation, they may lead to negative interactions with the transport layer. Most notably, the use of multiple paths with largely different delays may lead to a significant amount of packet reordering and thus reduce the throughput achievable by TCP.

The potential of multipath transport to improve the performance of video streaming has been studied in the context of Content Delivery Networks [3] and wireless ad hoc networks [24]. These studies show that path diversity can improve video-streaming performance if

packet-loss and delay on different paths are uncorrelated [28]. However, these studies assume that the available bandwidth on a path is constant and do not analyze the impact of competing traffic. As such in the cases where the network resources are shared through congestion control, their applicability in the Internet is limited. More recently, the performance of multi-path streaming over TCP was studied in [37]. The authors show that a streaming application can achieve a higher throughput by utilizing several TCP connections in parallel. The impact on competing traffic on a share bottleneck (fairness), as well as a potential impact on network stability were not studied in the paper.

Recently, adaptive streaming was combined with multipath transport in [14]. The authors show the feasibility of using HTTP byte range-requests at the application layer to request video segments from several servers in parallel. Benefits in terms of throughput and quality are observed in a number of scenarios. Although this proposal is attractive in terms of utilizing existing protocol stacks for multi-path streaming, it is unclear whether this method can be applied on a global scale without considering its fairness on shared bottlenecks and the impact on network stability. In our case of using MPTCP, the fairness to competing flows and network stability are ensured by MPTCP design [37].

1.3 Contributions

In this thesis, we focus on the design and implementation of efficient adaptive streaming using the MPTCP. Our specific contributions are as follows.

1. We have designed and implemented an efficient bandwidth measurement and estimation mechanism, described in Section 3.2. We use pcap library for accurate bandwidth measurement and dynamic window-based harmonic averaging for estimation. This solution is applicable to adaptive streaming using both the TCP and the MPTCP.

2. We have developed a generic (protocol-independent) streaming rate-adaptation algorithm that allows to improve bandwidth utilization and user experience with an optimized trade-off between streaming efficiency and streaming-rate stability (Section 3.3). Our adaptation algorithm is flexible in that it can be configured to work in different modes, each of which achieves a given optimization objective. We also show how this generic algorithm can be extended to improve MPTCP streaming performance.

3. We have incorporated the algorithms for bandwidth estimation and adaptation in a streaming testbed that supports multimedia streaming over TCP and MPTCP.

We use this testbed for experimental evaluation of the proposed algorithms. Our experiments use datasets compliant with the DASH standard. As a part of our evaluation, we have experimentally found the best segment size in our dataset for streaming over MPTCP. Our dataset and experimental testbed are described in Sections 4.1 and 4.2, respectively.

4. We have validated the implemented components with respect to the required functionality and evaluated their performance. The evaluation results are given in Sections 4.3 and 4.4. We have also compared streaming performance over TCP and MPTCP under different network conditions in terms of delay and packet loss. The results of this comparison are given in Section 4.5.

1.4 Thesis Outline

The remainder of this report is structured as follows. In Chapter 2, we give an overview of the relevant technologies for multimedia streaming. We survey multimedia streaming protocols and discuss HTTP/TCP streaming in detail. Next, we provide a discussion on resource pooling principle and present an architectural overview of MPTCP and its congestion control implementation. Finally, we introduce the DASH standard and its main components. Chapter 3 introduces the proposed multi-path streaming and its components. Each component is discussed in detail, including the design motivation and the implementation details. Chapter 4 presents the evaluation of each component and the performance of MPTCP streaming under different network conditions. Finally, we conclude the thesis with our findings on the use of MPTCP for streaming in Chapter 5 and a discussion of opportunities for future work.

Chapter 2

Background

In this chapter we present an overview of the existing standard protocols for multimedia streaming. As our work involves the use of resource pooling for multimedia streaming, a general overview of the resource pooling principle is given along with some examples. In the later sections, we introduce the details of MPTCP, which implements the resource pooling principle. We overview its architecture, components and its implementation of the coupled congestion control. In the end, the new multimedia streaming standard DASH is introduced in more detail.

2.1 Streaming Protocols

Many protocols have evolved to support the multimedia transmission over the Internet. They belong to different layers of the OSI model[1] of the protocol stack and provide different kinds of services to the applications. Two of the most widely used protocol combinations are HyperText Transfer Protocol/ Transmission Control Protocol (HTTP/TCP) and Real Time Protocol/User Datagram Protocol (RTP/UDP). Each of these combinations provides different kinds of streaming services.

RTP/UDP based streaming uses RTP/RTCP, an application layer protocol and UDP, the transport layer protocol [35]. RTP/RTCP was specifically designed for media transfer and it is characterized with its own packet structure and session management capabilities. RTP/UDP combination supports both multicast and broadcast transmissions. Even though it works well in managed networks, it has its own drawbacks.

- RTP is resource intensive as it manages each streaming session on the sender.

[1] http://en.wikipedia.org/wiki/OSI_model

- RTP packets are blocked by firewalls and cannot pass through the current Internet.

In today's Internet, RTP/RTCP is limited from deployment on a large scale. In contrast, HTTP/TCP streaming has seen much deployment in today's Internet. However, it supports only unicast transmission and media is delivered in large segments. HTTP/TCP based streaming offers many advantages compared to RTP/RTCP, as mentioned earlier in Chapter 1. Next, we will give a brief overview of the TCP and the HTTP protocols, as only these are used in our framework.

2.2 HTTP/TCP Streaming

2.2.1 TCP

TCP[2] is the most widely used transport layer protocol. It provides logical end-to-end connection between the applications running on end hosts. Unlike other transport layer protocols, TCP is connection-oriented and provides reliable in order byte delivery of data over the unreliable underlying network. In addition, TCP implements the congestion control mechanism. TCP provides services to the upper application layers by implementing principles like error detection, retransmission, cumulative acknowledgement and transport layer headers. TCP identifies each application on an end host as Transport Service Access Point (TSAP). TSAP is a combination of 32-bit IP address and 16-bit port number, which uniquely identifies the applications running on any host over the Internet [35].

2.2.1.1 Congestion Control

The TCP is widely seen as being essential for the stability of today's Internet. It was introduced in the early ARPA-Net to alleviate the congestion collapse problem. 'Congestion Collapse' is a condition of a computer network, in which little or no useful communication is happening due to congestion. Congestion collapse generally occurs at choke points in the network, where the total incoming traffic to a node exceeds the outgoing bandwidth. In such a condition, network settles (under overload) into a stable state where traffic demand is high but little useful throughput is available and quality of service is extremely poor.

In order to avoid this condition, TCP implements end-to-end congestion control mechanism. TCP maintains a variable called 'Congestion window'(cwnd) on the sender side

[2] http://www.ietf.org/rfc/rfc0793.txt

and detects congestion based on the observed behavior of the network, such as packet loss or increase in delay. It also regulates the rate at which the sender sends the data into network based on 'Congestion window' parameter. Design of the TCP congestion control is based on two principles, namely

- A packet loss implies congestion, so TCP sender's sending rate should be decreased.
- An acknowledgement implies that packets are reaching the receiver and hence sender's sending rate can be increased.

2.2.1.2 TCP Congestion Control Algorithm

Congestion control algorithm [22] controls the behavior of the TCP sender in response to observed network conditions. Algorithm works in two phases namely, slow start and congestion avoidance.

Slow Start : Each TCP connection starts in slow start phase, with congestion window set to 1 MSS(Maximum Segment Size). For every acknowledgement, the congestion window increases by 1 MSS. TCP sends the first segment and on receipt of the acknowledgement it increases the congestion window by one MSS. Thus, congestion window grows exponentially after each round. The operational behavior of the slow start phase is shown in Figure 2.1. TCP sender in a slow start phase has the following behavior:

- On a segment loss event, cwnd is set to 1 and enters the slow start phase again. TCP sets the variable 'ssthresh' value to the half of the cwnd when the segment loss happens. The ssthresh is used by the TCP sender to move from the slow start phase to congestion window phase.
- If the congestion window reaches the ssthresh value, then TCP sender moves to the congestion avoidance phase.

Congestion Avoidance : In this phase, initially the cwnd value is set to half the value of the congestion window when congestion happened. TCP increases the congestion window linearly, 1 for every Round Trip Time(RTT). On a segment loss event, TCP sets the cwnd value to 1 and the ssthresh value is set to half the cwnd. Congestion window undergoes the 'Additive Increase and Multiplicative Decrease' behavior over the TCP connection. In addition, TCP always uses the min(cwnd, receiver window) in order to ensure that it is having enough space at the receiver buffer and not overloading it with data. Variation of the congestion window with time on loss events is shown in Figure 2.1.

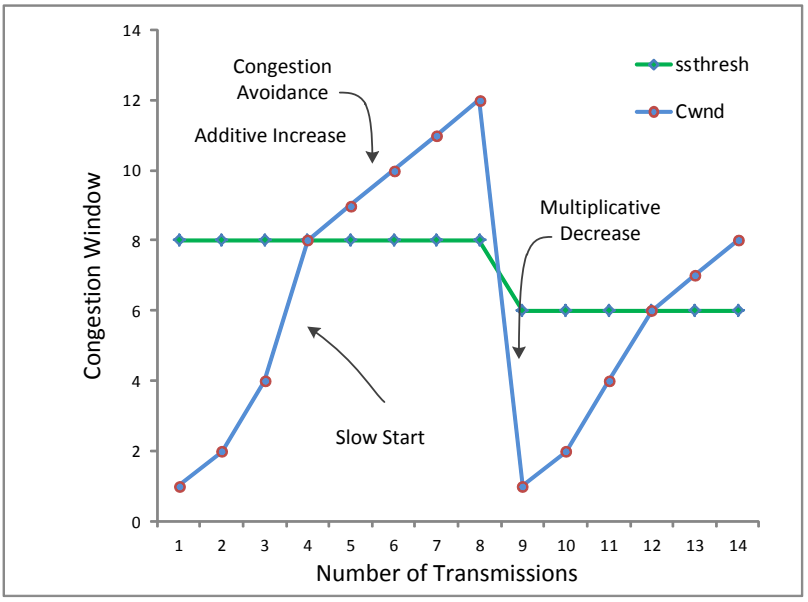


Figure 2.1: TCP congestion control behavior

2.2.2 HTTP

HTTP is the most widely used application layer protocol [18] for media transmission over TCP. It defines its own message formats and also how the client and the server exchange messages. HTTP is a stateless protocol, as it does not maintain any state information about the client and its requests. It supports two connection modes, persistent connection mode and non-persistent connection mode. Persistent connection mode corresponds to using the same TCP connection to serve multiple requests from the same client over a period of time. Non-persistent connection mode corresponds to using separate connection for serving each request.

HTTP defines two types of messages for its operation, request messages and response messages. A typical HTTP request message can be illustrated with the following example:

```
GET   /somedir/page.html   HTTP/1.1
Host: www.nt.uni-saarland.de
Connection: close
User-agent: Firefox
Accept-language: En
```

HTTP messages are ASCII text messages and consist of multiple lines. First line corresponds to the request line and the subsequent lines are called header lines.

- Request line has three fields, method, URL and the HTTP version field.

- Method corresponds to the action to be taken and possible options include GET, POST, HEAD, PUT and DELETE.
- GET method is important for our purposes, since it is used for accessing the object or file on the server.

HTTP Response Message can be illustrated with the following example:

```
HTTP\/1.1. 200 OK
Connection:  close
Date: Wed Dec 26 09:17:28 CET 2012
Server: Apache/2.2.3 (Debian)
Transfer-Encoding: chunked
Content-Type: text/html

(data data data ..............data)
```

The response message includes three subsections, namely: status line, header line and the data. Status line denotes the status of the server and includes protocol version, status code and status message. Each status code informs the client about the result of the request. For example, code '200 OK' states that the request has been successful. Similarly, status '404 Not Found' is sent if the requested document does not exist at the server.

2.3 Multipath TCP (MPTCP)

At the beginning of Internet communications, end hosts had a single interface and only routers were equipped with several physical interfaces. As a result, none of the end host-based Internet protocols were designed to use multiple interfaces. By design, today's dominant transport protocol - TCP - does not support the use of multiple interfaces simultaneously. This limits the present Internet to benefit from resource pooling on a global scale. However, today most of the end hosts have multiple interfaces, for example almost every notebook is equipped with Ethernet and Wifi Interfaces. Also, smart phones are provided with Wifi and 3G and are being used on a large scale to access the Internet. Extending the present Internet protocols or designing a new protocol to support the concurrent use of multiple interfaces is a possible and open option. Concurrent use of the multiple interfaces corresponds to using multiple available paths between the end points. This may result in a better availability or a better performance.

In this section, the resource pooling principle, existing localized mechanisms and its benefits are introduced. Pooling the capacity of the available paths and reliability

pooling are illustrated with the examples. Next, an end-to-end overview of MPTCP, its architecture, components and the congestion control algorithm is presented in detail.

2.3.1 Resource Pooling Principle

Internet is a network of networks. It consists of multiple paths between communicating hosts, with different characteristics like bandwidth and delay. Resource pooling corresponds to making a collection of these networked resources behave as though they make up a single pooled logical resource, with increasing reliability, flexibility and efficiency [38]. Resource pooling requires multihoming and multipath transport over the network by the end systems. Multihoming corresponds to end hosts having multiple IP addresses or interfaces. Multi-path transport refers to the ability of the end devices and the network to use multiple paths for transporting the data between two endpoints. In case of a path failure, the data traffic can be moved to another path shown in Figure 2.2, thus supporting the resilience to network failures. In addition, simultaneous use of multiple available paths may lead to a better throughput, lower packet loss and lower delay. Also, it enables moving traffic from congestion path to a less congested path, achieving load balancing as shown in Figure 2.3.

Benefits of resource pooling as a general concept can be summarized as [38]:

- increased robustness against component failures;
- better ability to handle localized surges in traffic;
- maximized utilization of the available resources;
- balancing load between various parts of the network.

For illustration, consider Figures 2.4 and 2.5 that represent the capacity pooling and reliability pooling, respectively. In Figure 2.4, the entire 36 Mbps capacity is shared fairly among the multiple paths, as a result of which each flow achieves more than that what can be achieved using a single “best” path. For instance, without resource pooling, Src A can use either 6 Mbps or 10 Mbps link any time. But with the capacity shared (resource pooled) between flows, path from Src A to Dst A uses both links and achieves a throughput of 12 Mbps. Figure 2.5 represents the partial pooling of reliability. Src A is more robust to failure than using single path among the four middle paths. If any of the four links goes down, traffic can be moved entirely to other three available paths. Each of the nodes in Figure 2.5 is more robust to link failures than in Figure 2.4.

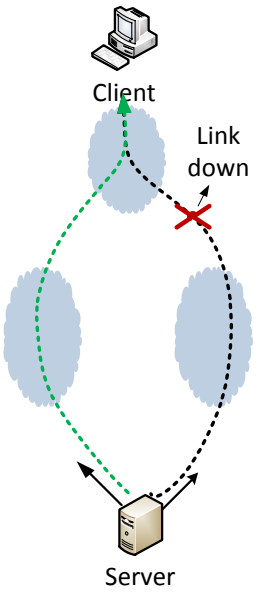


Figure 2.2: Failure resilience.

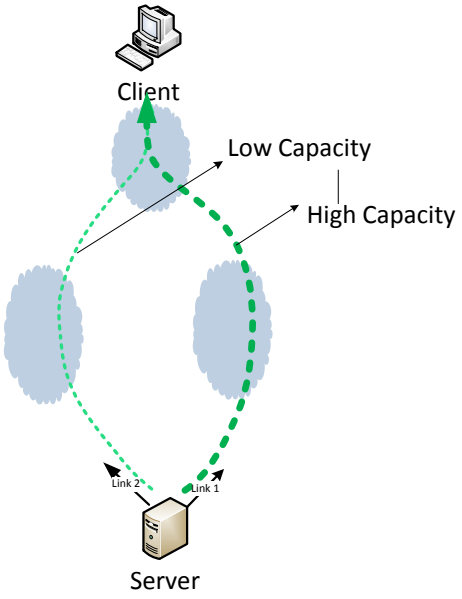


Figure 2.3: Load balancing.

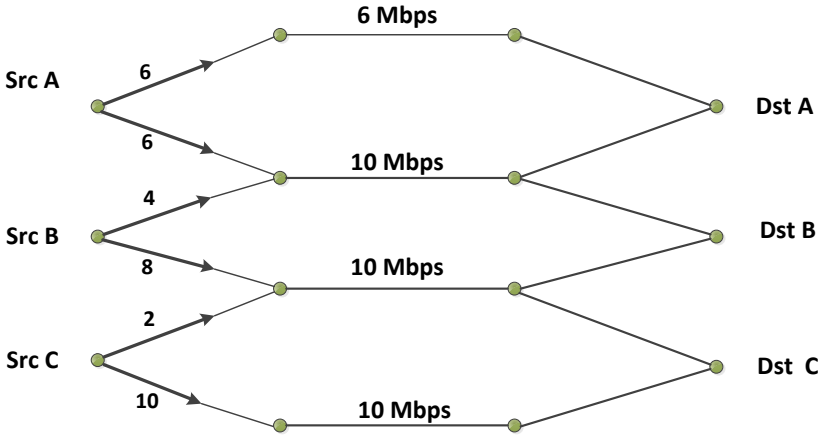


Figure 2.4: Capacity pooling [38].

The idea of resource pooling is not new and has existed since the beginning of the Internet [38]. Some of the examples are given in the following, Peer to peer networking is used to pool instantaneous upload capacity of multiple end nodes [38]. More recently, Content Distribution Networks (CDNs) pool the CPU cycles, bandwidth and reliability of the distributed servers.

In the papers of Iyengar *et al.* [19] and Wischik *et al.* [9], it was argued that among all the layers of the protocol stack, transport layer of end hosts is the most suitable for implementing resource pooling on a global scale. First, the transport layer has more accurate information about the performance of a specific path than other layers. Second, due to its interface to the application, the transport layer may customize path selection

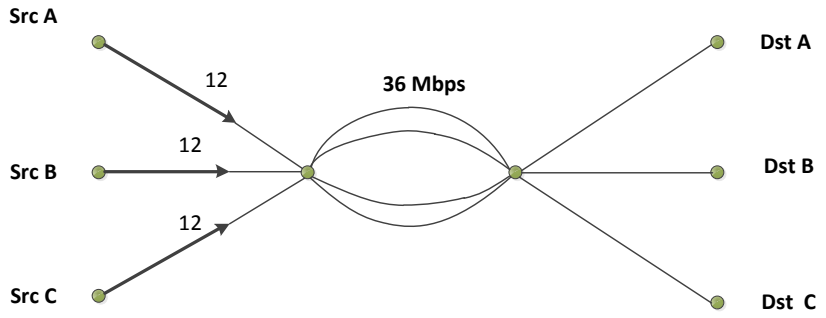


Figure 2.5: Partial reliability pooling [38].

for the application's specific needs. Third, the traffic balancing on the transport layer can be implemented in a distributed fashion and thus on a small timescale. As a result, path switching can be responsive to abrupt changes in path properties.

However, designing a resource pooling mechanism at the transport layer poses several technical difficulties [4]. First, although most of today's end systems fulfill the basic requirements for resource pooling in that they are equipped with multiple interfaces, the transport layer cannot use those interfaces in parallel. Second, a solution is needed to use those interfaces efficiently and transparently. Third, congestion control in multihomed and multi-path transmission scenarios is still an open problem. Fourth, in order to simplify the deployment, an implementation that extends TCP and its well-known application interface is desired.

There have been attempts to implement resource pooling at the transport layer. For example, extensions to TCP such as pTCP, R-MTCP have been developed. SCTP (Stream Control transmission protocol) [19] was designed for multihoming. However, although these solutions are standardized and several implementations exist, their current deployment is very limited.

Multipath TCP (MPTCP) [1] is an extension of the TCP [29], designed to overcome the limitations and implement the resource pooling on a global scale.

2.3.2 MPTCP Protocol Details

MPTCP is a set of extensions to regular TCP that allow the TCP connection to be spread across multiple paths. The main design goal of MPTCP is to "allow a pair of hosts to use several paths to exchange segments that carry data from a single connection" [1].

MPTCP uses multiple IP addresses in a single TCP session, which correspond to multiple interfaces. As a result, resource pooling is achieved through the use of multiple paths for the same connection. In addition, it provides compatibility with the application layer and the network layer. With network compatibility, MPTCP remains backward compatible with the Internet as it exists today. As a result, MPTCP segments can traverse through the Internet without being blocked by NATs, firewalls and performance enhancing proxies. Application compatibility refers to the appearance of MPTCP to the application, which is designed to be the same as TCP's, both in terms of API and the offered service model. This enables existing TCP applications to use MPTCP without modifications. Figures 2.6 and 2.7 shows the comparison of standard TCP and MPTCP protocol stacks.

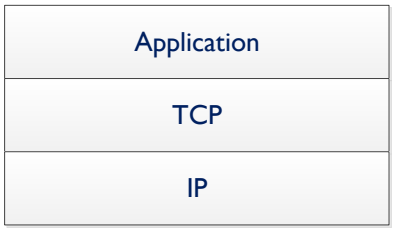


Figure 2.6: TCP protocol stack.

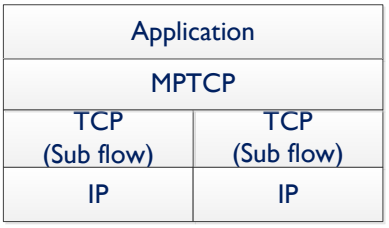


Figure 2.7: MPTCP protocol stack.

MPTCP has been been designed to fulfill the following goals:

1. **Improve throughput:** When using multiple paths, throughput should be at least equal to the throughput achievable on the best path using TCP.
2. **Do no harm:** Competing flows should not be affected more than if a single-path protocol was used.
3. **Balance congestion:** Automatically move the traffic from more congested paths to less congested paths.

2.3.2.1 MPTCP Architecture

Architecture of MPTCP is as shown in Figure 2.8. MPTCP architecture directly follows from the goals presented above. It consists of three main components namely,

- Master subsocket
- Multipath control block
- Slave subsocket

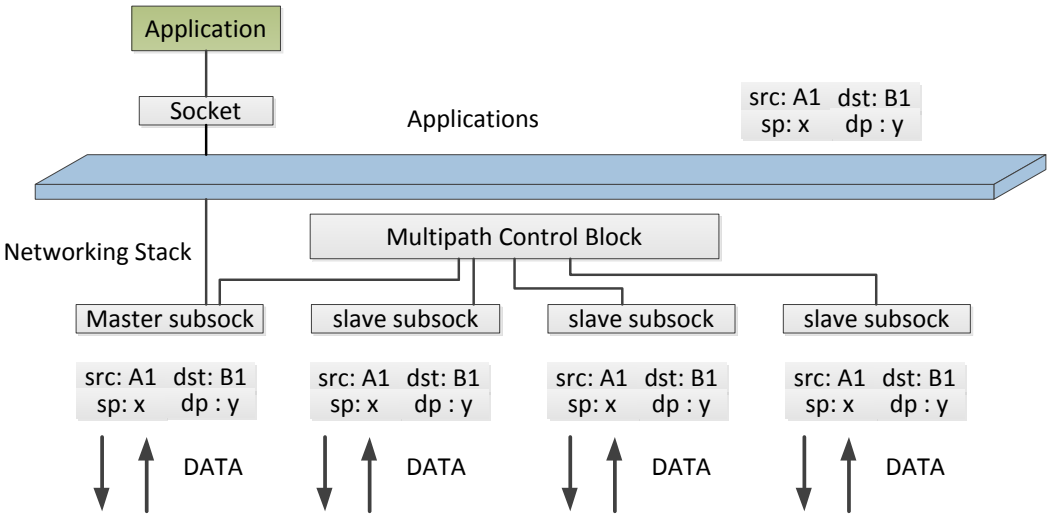


Figure 2.8: Overview of MPTCP architecture [4].

Every component implements one or multiple transport layer functions such as path management, packet scheduling, packet reordering, reliable delivery and congestion control. They are either derived from TCP or newly developed to implement existing as well as new transport layer functionalities. Master subsocket is a standard socket structure that provides the interface between application and kernel for TCP communication. If MPTCP is not supported by the other end system, then only master subsocket is used with regular TCP.

Multipath control block is the main component that runs the decision algorithm, the scheduling algorithm and the reordering algorithms. Decision algorithm decides when to start and stop the subflows. Scheduling algorithm is used to feed the application data to the available subflows. Reordering algorithm reorders the incoming bytes and provides in order delivery of data to the application. Reordering algorithm runs on both the connection level and the subflow level using connection-level sequence numbers and subflow-level sequence numbers. Slave subsockets are used for regular data transfer as done in TCP, and are not visible to the application. The master subsocket and the slave subsockets share the same functionality and form a pool of subflows, which are used by multipath control block.

2.3.2.2 Protocol Operation

To understand the operation of MPTCP, consider a case where a mobile client has two addresses, as shown in Figure 2.9. Initially it establishes a connection with a single-homed server (only one interface). This connection is setup using a normal handshake procedure

as done in TCP. Client sends a SYN segment from a single interface that contains the MP_CAPABLE option. MP_CAPABLE option is used for indication of support for MPTCP. Host on the other side replies with SYN+ACK segment with MP_CAPABLE option set. On receipt of this segment, initiator of the connection replies with ACK segment concluding the initial setup, as it is done in TCP. Once the initial connection has been established, additional addresses (if present) are advertised to the other end using ADD_ADDR option. With the additional address information, a new connection will be set up using the normal handshake procedure. Each of the connections including the initial connection uses a unique identification number. Using this unique identifier, additional connections (subflows) will be attached to the main subflow. More information on sequence of messages and options exchanged between the hosts during MPTCP operation can be found in [1].

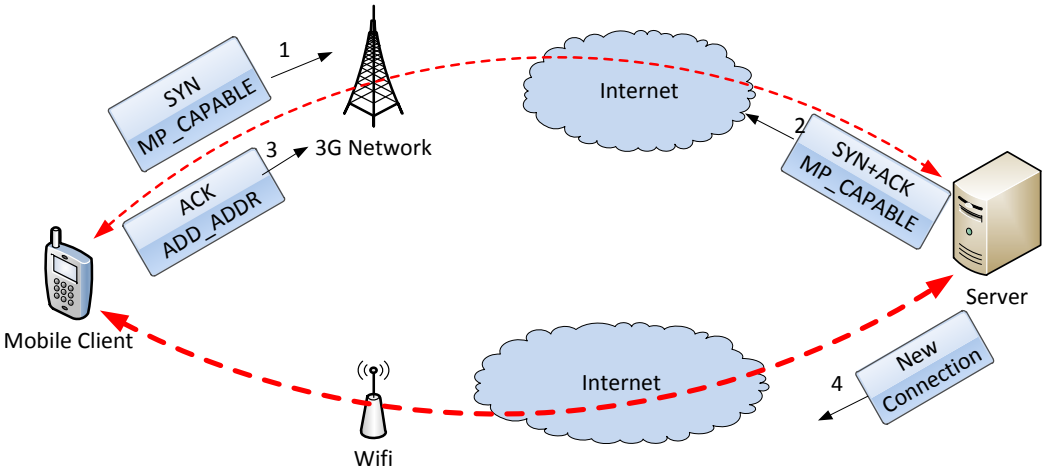


Figure 2.9: Mobile client with two interfaces [4].

Reordering Algorithm: As mentioned earlier, during the operation of MPTCP, if any of the subflow fails, data can be transferred over other subflow. To enable this, MPTCP uses sequence number spaces on subflow level and connection level separately. MPTCP treats each of the subflows as normal TCP connections with their own 32-bit sequence numbers. As a result, MPTCP segments are treated as normal TCP segments by middle boxes resulting in network compatibility. On the connection level, MPTCP maintains 64-bit sequence numbers that are used for reordering of the data received by different subflows. Each segment also carries the value for mapping between 32- and 64-bit sequence spaces. As soon as the data is received in order in a subflow, it is passed on to the connection-level buffer and then given to the application [32].

Scheduling Algorithm: With multiple subflows are in operation, MPTCP decides on which subflow to send data. In order to implement this, MPTCP maintains connection

level parameters like congestion window, round trip time (RTT) and packet loss rate. With this information, MPTCP data scheduler chooses the subflow with lowest RTT. Once the congestion window becomes full or a packet loss occurs on the lowest RTT path, scheduler chooses the next lower RTT path. However, this behavior of the scheduler can be modified based on the requirements in future.

Flow Control: MPTCP stores the incoming data in a single connection-level buffer. The data are delivered to the application on request. Additionally, each flow maintains a receive buffer. As soon as the data is received in order on the subflow, it is moved to connection level buffer. Like regular TCP, MPTCP implements the flow control for each subflow and also for the connection.

2.3.2.3 MPTCP Congestion Control Algorithm

In MPTCP, since multiple subflows are in operation, using the same congestion control algorithm as provided in TCP on each subflow independently does not ensure fairness. To illustrate this, consider the bottleneck link in Figure 2.11 and suppose for simplicity that all RTTs are equal. Multipath flow D gets twice as much throughput compared to the single-path flow E, thus being unfair to other flows. Secondly, it is also required that multipath congestion control has to shift the traffic from congested paths to uncongested paths as shown in Figure 2.10.

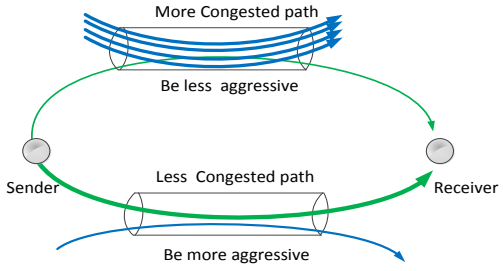


Figure 2.10: Coupled multi-path congestion control [9].

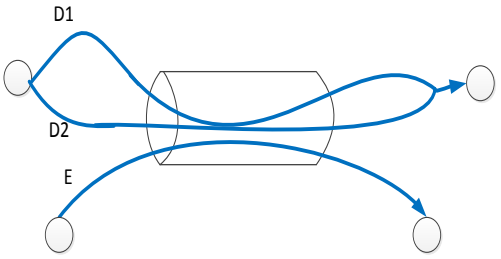


Figure 2.11: A bottleneck link with multiple flows [9].

MPTCP implements a coupled congestion control [6]. Coupled congestion algorithm does not modify the slow start and multiplicative decrease of the congestion-avoidance phase of the TCP. Instead, it only modifies the behavior in the increase phase of the congestion avoidance phase.

The MPTCP's coupled congestion control is implemented as follows. Let CW_i be the congestion window on the subflow i and CW_{total} be the sum of the congestion windows of all subflows in the connection. Let p_i, rtt_i, and MSS_i be the packet loss rate, round-trip time (i.e., smoothed round-trip time estimate as used by TCP), and maximum segment

size on subflow i, respectively. For each ACK received on subflow i, MPTCP increases CW_i by a value that represents the minimum of the computed value and the value of a standard TCP flow in the same scenario. This can be expressed as:

$$CW_i = CW_i + min(\frac{alpha * bytes\_acked * MSS_i}{CW_{total}}, \frac{bytes\_acked * MSS_i}{CW_i})$$

Taking the minimum of the two value ensures that the multipath flow cannot be more aggressive than a standard TCP flow. The *alpha* factor describes the aggressiveness of the multipath flow. It is computed dynamically based on the path properties as follows.

$$alpha = CW_{total} * \frac{MAX(CW_i/rtt_i^2)}{\sum CW_i/rtt_i)^2}$$

When multiple paths of equal RTT and MSS are available, for each acknowledgement, the total congestion window (over all subflows) will grow approximately by $alpha * MSS$ segments per RTT. On each subflow, the congestion window will increase by $\frac{alpha * CW_i}{CW_{total}}$ segments per RTT. When any of the paths is congested, its congestion window increases very slowly. On less congested paths, the increase is faster. In this way, the traffic can be shifted from congested paths to less congested paths. In the extreme case when a path goes down, the congestion window associated with this path becomes zero and no data is transferred on that path. Thus, MPTCP can also achieve resilience to path failures.

2.4 DASH Standard

HTTP/TCP based streaming is used as a de facto standard for the multimedia delivery in commercial Internet streaming deployments. For instance, major streaming platforms such as Apple's HTTP Live Streaming [27], Microsoft's Smooth Streaming [8], and Adobe's HTTP Dynamic Streaming [17] all use HTTP streaming as their underlying delivery method. However, each streaming solution is characterized by its own media presentation formats. As a result, no standard HTTP streaming solution exists today. Due to this, no interoperability between different streaming solutions can be achieved. For a client application to use all of the streaming formats, it has to implement and support all proprietary solutions. In order to overcome this limitation, a standard solution for media streaming is needed. A standard streaming solution would allow a standard-based client to stream content from any standard-based server, thereby enabling interoperability between servers and clients of different vendors.

MPEG-DASH standard is a new HTTP streaming standard in this regard. It provides interoperability between servers and clients of different streaming solutions. It specifies

the standard format for media segments and manifest files. Also, it specifies a normative definition of the delivery protocols and informative description of how a client can use the provided information to start and control a streaming session. DASH standard is specified as a set of profiles, where each profile supports different types of media streaming sessions. For example, on-demand, trick modes and live streaming are different types of profiles. DASH also supports many of the state-of-the-art video encoding standards like H.264. In the sequel, we introduce DASH standard in more detail.

2.4.1 DASH Scope

DASH standard consists of two main parts. First part of the standard defines the Media Presentation Description(MPD). MPD is an XML document that is provided by the server to describe the content information such as a "manifest" of the available content, its alternatives, URL addresses and other information. With this location and timing information of the media components, the client fetches and renders the media. Second part defines the format of the media segments, which contain the actual media bit streams in the form of segments, either in single or multiple files. The methods for the delivery of MPDs and segments, the algorithms for client adaptation and segment scheduling are not specified in the DASH standard. Figure 2.12 depicts a simple streaming session with DASH standard. Dark-shaded components such as MPD, segment formats, MPD parser and segment parser are specified by the DASH standard [33].

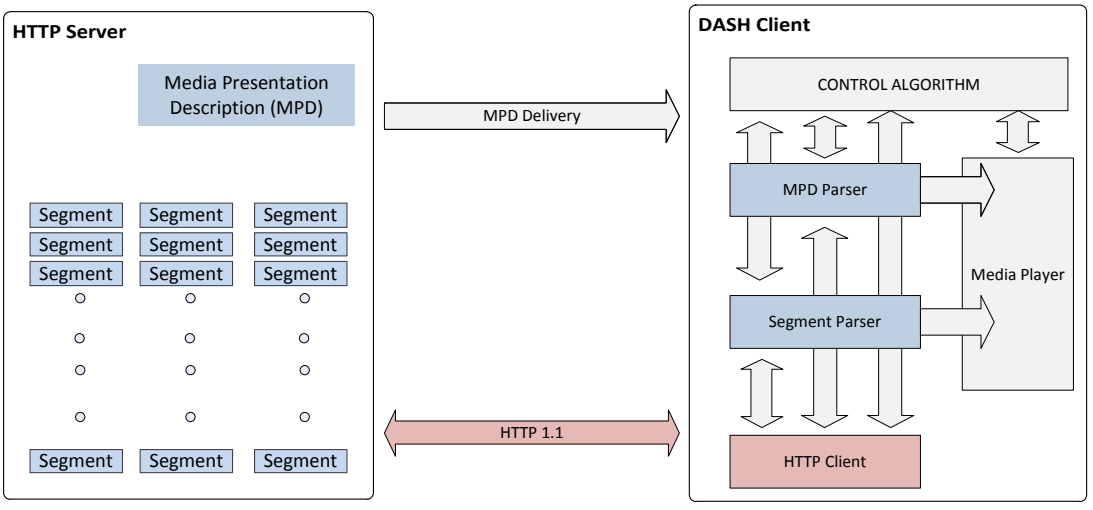


Figure 2.12: Scope of the DASH standard [33].

The two main components of the DASH standard are HTTP server and the DASH client as shown in Figure 2.12. HTTP server holds the media information and media data in the form of MPD and the segments, respectively. Media can exist in the form of a single

segment or multiple segments. In order to play the content, initially the client requests the MPD file from the server. When the MPD file is retrieved, the client parses the MPD file to learn about the program timing, media-content availability, media types and resolutions. In addition, it learns about the minimum and the maximum bandwidths, various encoded alternatives of media components, their accessibility features, locations on the network and other characteristics. Then the client selects the set of representations that it will use based on descriptive elements in the MPD, client's capabilities (e.g., resolution) and user's choices. Subsequently, the client builds a time line and starts playing the media content by requesting appropriate media segments. During the streaming session, based on the available network capacity and adaptation mechanism, the client may switch to the higher or lower encoded video versions if required.

2.4.2 Media Presentation Description (MPD) Model

An MPD represents information on a structured collection of encoded media data such as a movie or a program. It consists of components on multiple levels and is hierarchical in nature [33]. Three major components of MPD are Periods, Representations and Segments as shown in Figure 2.13. Periods are the top-level part of MPD and typically represent larger pieces of media. Each period is characterized with its start time, duration and one or more adaptation sets. An MPD may consist of one or multiple Periods. The adaptation set represents the next lower level and consists of one or more media components with multiple encodings in terms of bit rates, frame rates and video resolutions. For instance, an adaptation set might contain different bit rates of the video or audio components of the same multimedia content.

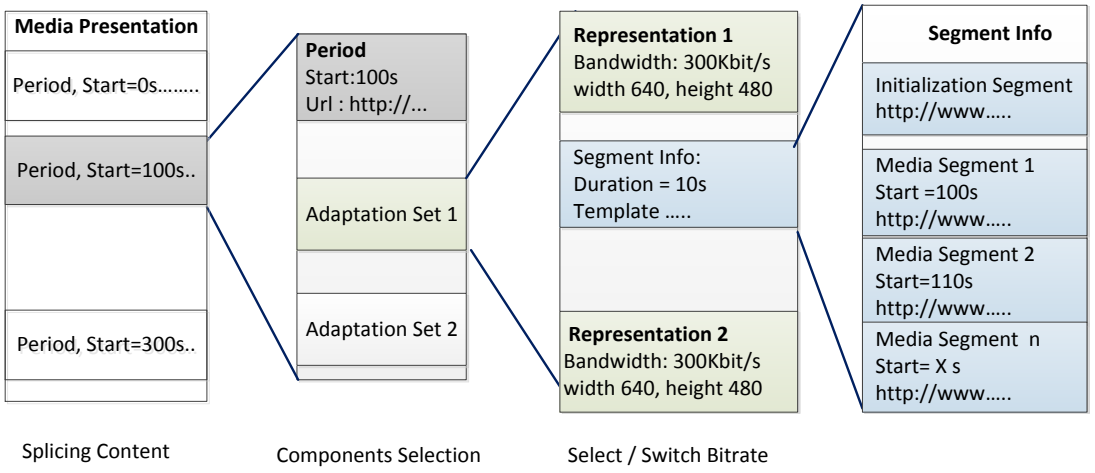


Figure 2.13: Hierarchical model of MPD [33].

On the next level, each adaptation set includes multiple representations. Each representation is an alternative encoding of the same media component, differing from other representations by bitrate, resolution, number of channels, or other characteristics. On the lowest level, each representation consists of one or multiple segments, which are the media stream chunks in temporal sequence. Each segment is associated with an addressable URL on a server that can be downloaded using HTTP GET or HTTP GET requests with byte ranges. These URL's are explicitly described in representation similar to play list.

2.4.3 VLC-DASH Plugin - an Implementation Prototype

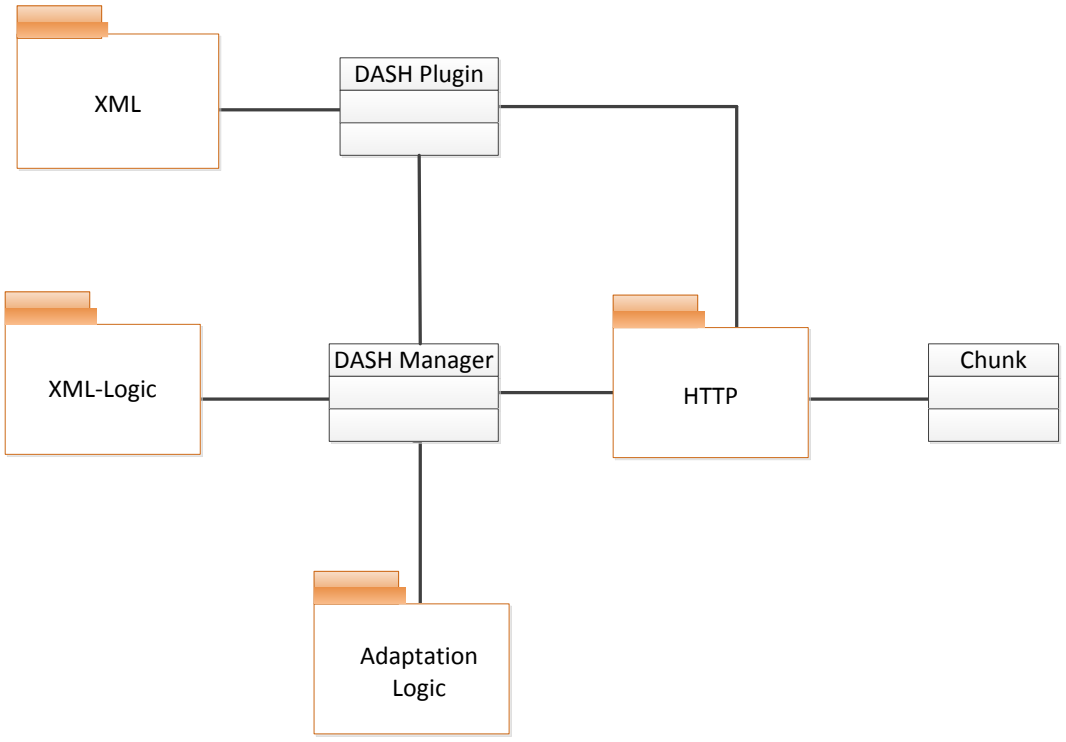


Figure 2.14: High level vlc-DASH plugin architecture [25].

Previous section gives an overview of the DASH standard, its components and scope. In this section we introduce the implementation of the DASH standard in Videolan media player, which is an open source project used in our work. The component of Videolan player that implements the DASH standard is called vlc-DASH plugin [25]. The vlc-DASH plugin implementation contains four major components and two controller classes. Figure 2.14 depicts each of the components and interdependence among them. The XML component is responsible for parsing the MPD file, which is an XML document.

It is invoked at the beginning of a streaming session so as to parse the MPD file and to provide support to other components during the streaming session.

All the functionalities of opening, maintaining, or closing of the HTTP connections are handled by the HTTP component. HTTP component provides both persistent and non-persistent connection mode. Adaptation of the streaming session to user preferences and device capabilities is handled by the adaptation-logic component. Adaptation logic is very flexible, enables easy change to the adaptation logic without affecting the other components. The vlc-DASH plugin implements on-demand streaming, but trick modes such as forward seek, backward seek and pause modes are not supported.

2.5 Summary

This chapter helped us to understand the media streaming protocols and their underlying principles. It also enabled us to understand the resource pooling principle in general and its benefits. It gives a brief overview of the new transport layer protocol - the MPTCP, which is an extension of TCP that implements the paradigm of using multiple available paths between end points. This chapter also introduced the MPTCP's coupled congestion control algorithm which ensures fairness and stability on a global scale. Finally, it enabled us to understand the DASH standard, its components and scope.

Chapter 3

Adaptive Multi-Path Streaming Algorithms

In this chapter, we give a technical overview of the proposed multi-path streaming. Our streaming implementation is based on the existing implementation of adaptive streaming available in the vlc-DASH plugin. We extend this implementation with improved versions of the `adaptation logic` and the `bandwidth estimator` components. Thus, both components are implemented at the streaming client. Therefore, only the client-side components with their architecture and implementation details are discussed in this chapter. We note that our components are designed to be compatible and experimentally shown to support, both multi-path streaming using MPTCP and single-path streaming using TCP.

3.1 Required Components

Figure 3.1 shows the main components for adaptive multi-path streaming. The main components are the `adaptation logic`, the `bandwidth estimator`, the `buffer` and the `scheduler`. Every component implements the required functionalities independently or in cooperation with other components. The `bandwidth estimator` measures the available bandwidth over the network, samples the measured values and averages them. Based on the present and the previously measured values, it estimates the bandwidth in the future. The `adaptation logic` component uses the estimates supplied by the `bandwidth estimator`, buffer status and a history of its adaptive decisions to select the bitrate of the next video segment to request. The `buffer` component stores the received video data and compensates for the jitter in the network behavior. Its capacity depends on the type of streaming performed. For instance, the buffer capacity during for stored

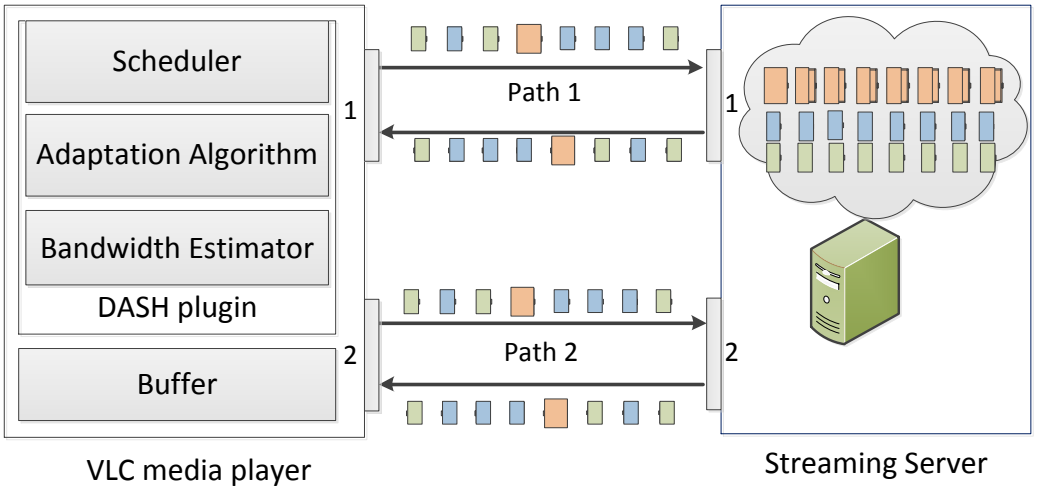


Figure 3.1: Components of the proposed adaptive multi-path streaming.

streaming is larger than during for live streaming. In our case is kept constant (30s) during the whole streaming session.

The `scheduler` component is responsible for scheduling requests for segments. It can run independently or be controlled by the `adaptation logic`. Once a requested segment is received completely, the `scheduler` may request the next segment immediately or delay the request based on the buffer status and network conditions. In our system, the `scheduler` delays the reading of the data from the socket buffer when the `buffer` reaches its full capacity. In this way, we rely on transport protocols' flow control to effectively delay requesting the next segment. The `scheduler` can also be configured to request the next segments based on the received fraction of segments already requested. For example, a new segment can be requested only after 75% of the present segment is received. In our system, we request new segments when 95% of the present segment is received, which is the default configuration of the vlc-DASH plugin. Since the `scheduler` component is not modified in our system, we do not further detail on its operation here.

3.1.1 Default Components in vlc-DASH Plugin

For a good adaptation performance, the bandwidth estimator needs to estimate the available bandwidth without overestimates and underestimates and also the adaptation logic has to use video bitrates based on the network as well as buffer conditions. In order to validate whether the vlc-DASH plugin satisfies this requirement, we performed a streaming experiment. Buffer capacity of the vlc-DASH client was 30s for the whole session. We varied the available bandwidth using a traffic shaper (Dummynet). More details on the dataset and the experimental testbed are given in Sections 4.1 and 4.2,

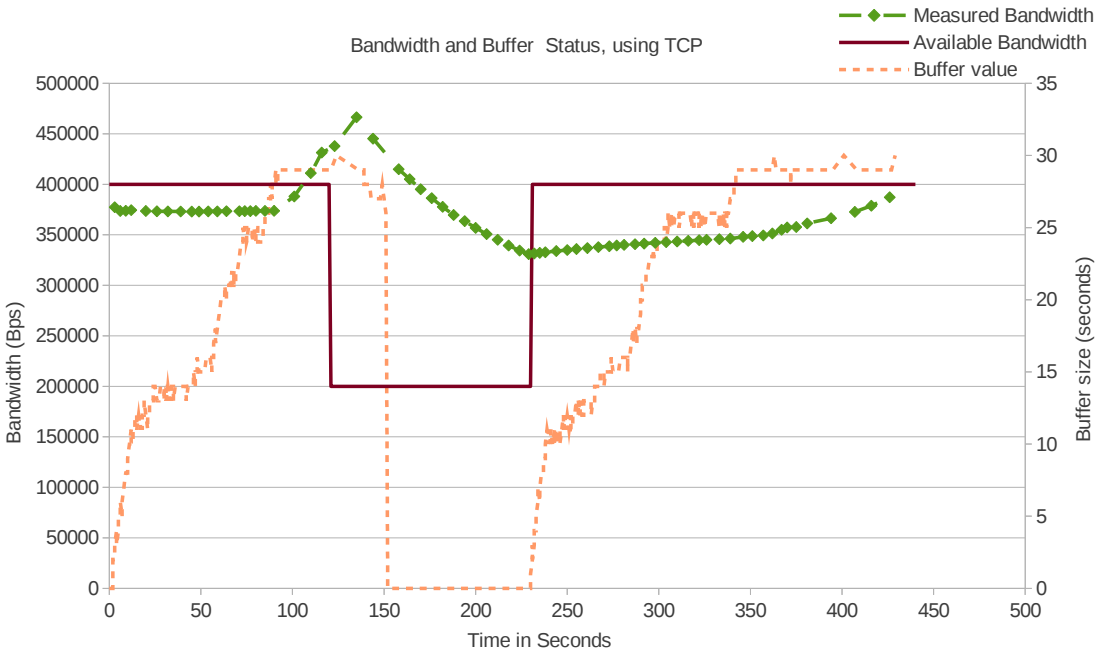


Figure 3.2: Bandwidth measurement by default vlc-DASH client.

respectively. We used Dummynet to limit the available bandwidth as follows. The round trip time is set to 60ms and packet loss rate to zero for the whole duration of the session. From the start of the session until 120s the available bandwidth is 400 Kbps. At time t=120s available bandwidth is reduced from 400 Kbps to 200 Kbps, until time t=240s. At time t=240s the available bandwidth is increased to 400 Kbps again.

As evident from the results in Figure 3.2, the bandwidth measured by the vlc-DASH client remains continuously below the actual available bandwidth, until the client buffer becomes full (roughly at t=90s). At these times, the adaptation uses the low video bitrates and moves to higher levels. After the client buffer is full, the measured bandwidth gradually increases. At time t=110s, the measured bandwidth approximately matches the available bandwidth and remains higher than the available bandwidth till time t=160s. At time t=120s, when the available bandwidth is abruptly reduced to 200 Kbps, the measured bandwidth only gradually decreases and remains larger than the available bandwidth for the next 120s, until t=240s. During this period, the client requests bitrates that are larger than the available bandwidth. These overestimates result in buffer drain from t=150s to t=240s. At time t=152 seconds, buffer becomes empty. The video playout first stalls for roughly 10s and then continues at a very low frame rate of around 1-2 frames per seconds. It is noted that, even though the buffer size becomes zero, the adaptation logic does not consider this and requests higher bitrate videos. At time t=240s, the available bandwidth is increased back to 400 Kbps. As the measured bandwidth is smaller than the requested bitrate, the video playout resumes at the original

frame rate.

To better understand the bandwidth measurement in the default vlc-DASH plugin, we analyzed the source code of the vlc-DASH plugin and found that the measured bandwidth is given as:

$$Measured\ bandwidth = \frac{Total\ no\ of\ bits\ received}{Total\ time\ taken} \tag{3.1}$$

This shows that the bandwidth measured is the long-term average of the bandwidth available from the beginning of the streaming session. This long-time averaging is not suited for adaptation in scenarios where the current available bandwidth significantly differs from the long-term average.

From the results we conclude that:

1. the rate-adaptation algorithm in vlc-DASH plugin does not consider the buffer status for its adaptation.
2. the bandwidth measurement in vlc-DASH plugin is inaccurate as it uses long-term averaging and fails to react to abrupt changes in the available bandwidth.

This motivates us for the design of a new bandwidth estimator and adaptation components. The design goals, implementation and the averaging mechanisms of the implemented components are given in the next sections.

3.2 Bandwidth Estimator

Available bandwidth measurements during a streaming session is one of the key functionalities in adaptive streaming. It refers to number of data bits that can be transferred over a link/path per unit time. Its value on a link is influenced by competing flows, underlying protocol in use, buffers at end devices (socket buffers at the transport layer and application buffers at the application layer) and the physical characteristics of the link such as capacity, delay, packet loss rate. The estimated bandwidth is used by the `adaptation logic` to select the bitrate of the next video segment to request. Therefore, in case of an inaccurate estimate, the bitrates of the requested video segments will not match the actual available bandwidth, resulting in streaming interruptions in case of an overestimate and unnecessarily low video quality (a degraded user experience) in case of an underestimate.

3.2.1 Design Requirements

One of the preliminary requirements for our `bandwidth estimator` is that it should be compatible and work with both TCP and MPTCP. Some of the additional aspects to be considered are:

- Bandwidth estimator should not alter the design or working of the Videolan player.
- The bandwidth estimator should be unaware of whether TCP or MPTCP is used during streaming.

In the course of the design exploration, we examined two potential solutions. The first solution was to use packet-pair techniques for bandwidth estimation during a streaming session. However, this approach would work only for TCP, as with MPTCP we cannot determine which paths will be selected to send the packets. The second solution we considered was to periodically use the TCP connection information from the `tcp_info` structure maintained for each TCP connection. This works well for TCP as Videolan player maintains the socket identifier for each connection. But with MPTCP, no API (at the time of this writing) exists to get the information of all the subsockets used by MPTCP.

Our bandwidth estimation is based on interface-level packet capture with pcap library on each interface. We describe the details of this solution in the sequel.

3.2.2 Bandwidth Measurement using Pcap

For our bandwidth measurement, we use libpcap[1], a portable C/C++ library for network traffic capture. It is a packet sniffer library which enables sniffing packets at different levels such as interface level, connection level, port level etc. This packet capture can be configured to work independently from the application (in a separate thread), which decouples the application functionality from packet capture. In our approach, we used interface level packet capture with a separate pcap thread on each interface. But among the captured packets, some packets may be from different hosts than the streaming server. In order to ensure the correct measurement, captured packets needs to be filtered. Filters are designed such that it filters the packets based on the source address, destination address and port number. This way of filtering is appropriate for our case, since each connection is identified by source address and destination address in the network layer and by port number in the application layer. Example filter expressions used in our implementation are given in Table 3.1.

[1] http://www.tcpdump.org/

Filter Expression	Performed Function
`host` media-server	Captures all packets arriving at or departing from media-server machine
TCP `and` src `and` dst net localnet	Selects all packets of each TCP connection that involves a localnet.
(src host A `or` src host B) `and` (port 80)	Captures the HTTP/TCP traffic on an interface from a multihomed host with addresses A and B.

Table 3.1: Example pcap filter expressions.

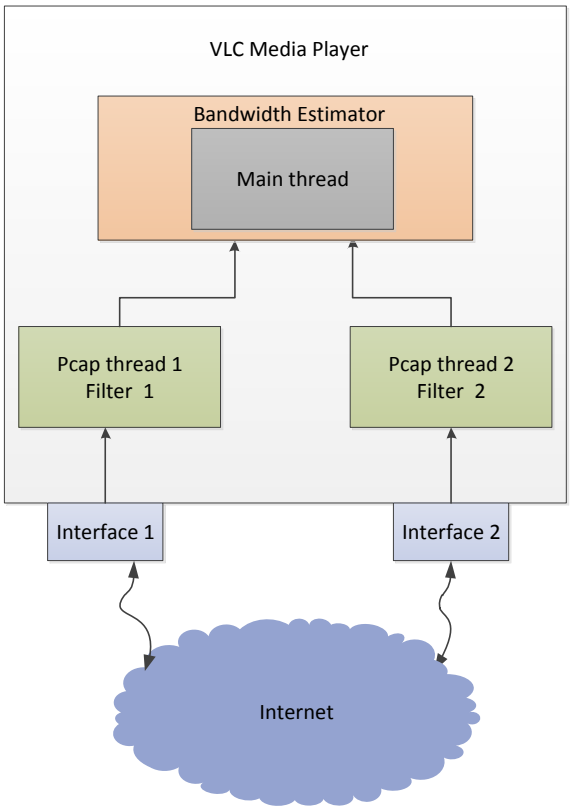


Figure 3.3: Bandwidth estimator architecture.

3.2.3 Implementation

To measure the bandwidth over multiple interfaces, the `bandwidth estimator` implementation consists of independent threads of pcap packet capture filters running per interface. Each thread captures packets that are destined only for the associated interface and uses the filter to process packets. As our testbed operates in a controlled lab setting, we ensure that no other TCP connections exists with same destination. The architecture of the estimator with three independent threads is shown in Figure 3.3.

Each thread captures the number of bits received per second on its underlying interface

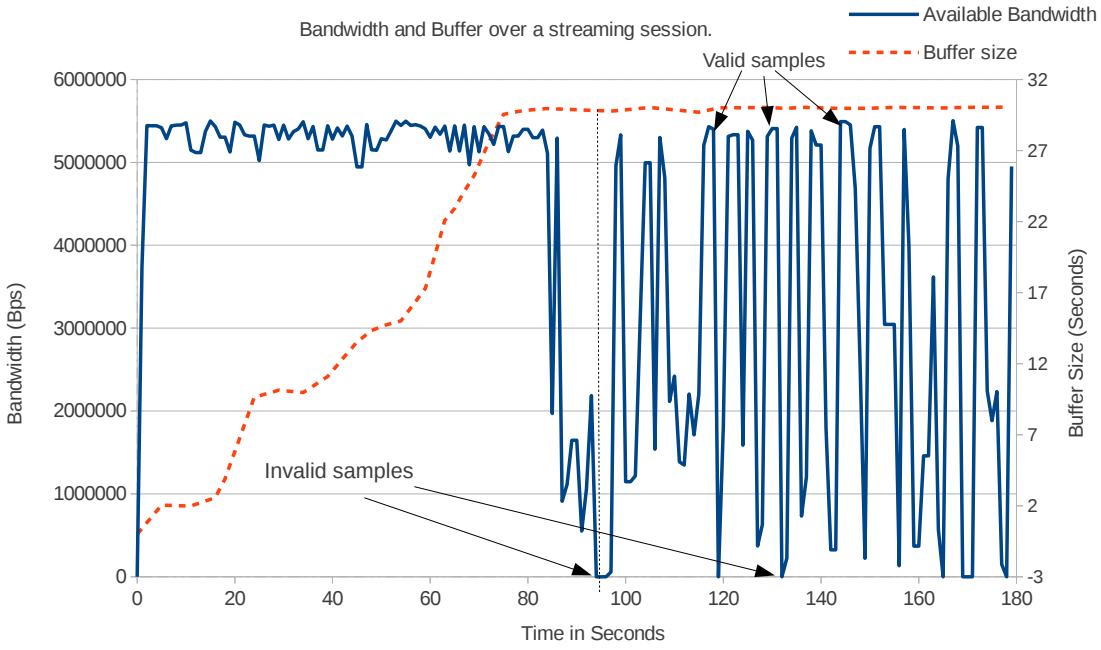


Figure 3.4: Streaming requests under different conditions.

(this sampling interval can be changed) and updates the corresponding data structures. Lock mechanisms using mutexes are implemented, so that the operations on shared variables are synchronized between all the threads. The main thread, which runs independently of other threads, reads the data from other threads and updates the total available bandwidth.

3.2.4 Sampling Mechanism

The sampling mechanism ensures that among all the measured bandwidth samples, only the valid ones are considered for averaging. We motivate the need to select the valid samples among all collected measurement samples as follows. During our preliminary experiments, we observed that when the client buffer is full, there are periods without data transfer between segment requests, as shown in Figure 3.4. This is due to the fact that the vlc-DASH plugin requests segments in discrete time intervals. During the idle periods (between two consecutive requests), no data transfer happens as no segments are requested, resulting in zero-valued samples.

These zero-valued samples are considered invalid for bandwidth estimation. Using zero-valued samples would result in erroneous average values. Therefore, in our case, valid samples correspond to the measured bandwidth values when the segment is requested and the channel is used to its full capacity (i.e., the available bandwidth). To ensure that we only consider valid samples in the estimation, in each segment period we only use the

maximum sampled value. This approach is guaranteed to give accurate estimates in case the segment transfer saturates the link. Otherwise, the sample is an approximation of the actual link capacity (i.e., the available bandwidth of the path).

3.2.5 Averaging Mechanism

The available bandwidth over time may have large amplitude fluctuations for short time periods. These fluctuations manifest themselves as positive and negative spikes in our valid sample set. Since in many case these spikes are caused by a transient congestion, they do not reflect average bandwidth conditions on a path (which are relevant for rate adaptation). However, due to their large amplitudes, the impact of spikes on the estimated bandwidth is significant. Our approach is to mitigate their impact by smoothing. To this end, our `bandwidth estimator` uses harmonic averaging over a dynamic window of the measured samples. Harmonic averaging is known to be more robust than arithmetic averaging [20], as shown in Figure 3.5. In this example, both the arithmetic and the harmonic averaging use a window of last 10 samples. It is evident that the harmonic averaging is less responsive to positive peak values and more responsive in case of negative peak values. This is a desired behavior for the averaging in streaming applications [10]. Harmonic averaging in our implementation uses a dynamic window with the minimum size of 2 and the maximum size of 10. More formally, averaging applied to a window of N most recent valid samples is given by as:

$$X(t) = \frac{N}{\sum_{i=1}^{N} x_i} \qquad 2 \leqslant N \leqslant 10, \tag{3.2}$$

where, $X(t)$ is the harmonic average of the bandwidth at time t, N is the size of the dynamic window and x_i is i^{th} valid sample in the dynamic window.

The dynamic window size is determined dynamically based on the previous values, their mean and the variance. The mean and the variance are given as:

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} \qquad 2 \leqslant N \leqslant 10 \tag{3.3}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N-1} \qquad 2 \leqslant N \leqslant 10 \tag{3.4}$$

Variance over a set of samples denotes how far the samples are spread out from the mean value. A large variance corresponds to larger fluctuations. Therefore, a larger

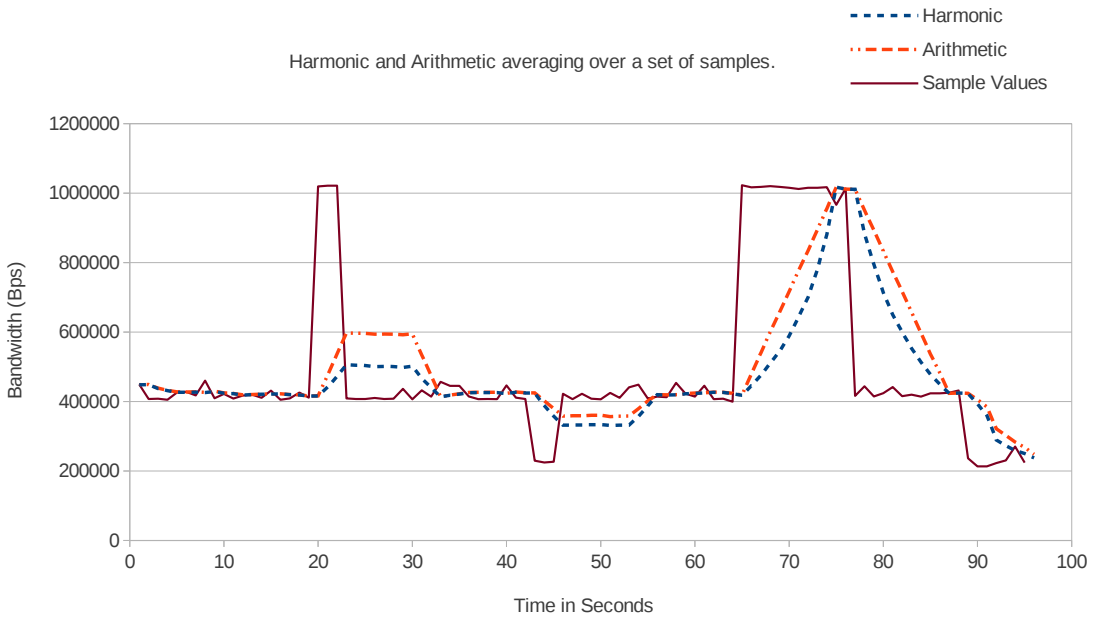


Figure 3.5: Arithmetic and harmonic averaging over a set of samples.

window has to be used to compensate for the highly varying bandwidth. In this case, our `bandwidth estimator` increases the dynamic window by 1 when the variance is higher than its previous value. Similarly, a small variance value corresponds to low variations from the mean value and hence smaller windows are suitable. In this case, our `bandwidth estimator` decreases the dynamic window by 1 when the variance is lower than its previous value.

As mentioned above, in our implementation the dynamic window size is limited to vary between a minimum value of 2 and a maximum value of 10. The intuition behind these values is as follows. The minimum window size of 2 is chosen to ensure that even on a stable channel, averaging is done to minimize the impact of spikes and still react to changes in the bandwidth. The maximum window size of 10 ensures that the averaging is long enough to filter out most of the impact of fast variations, while still being able to react to slow bandwidth changes in a reasonable amount of time.

As only one sample in each segment is considered to be a valid sample, the valid samples are typically placed a couple of seconds apart (the actual displacement depends on the used segment size). As a result, if the window size N is large, we may fail to detect abrupt changes in the bandwidth. To overcome this, our `bandwidth estimator` defines upper and lower bandwidth thresholds. We compute the thresholds using the current bandwidth estimate and the bitrate-step inherent in our dataset. If the available video bitrates are 300, 500, 800 and 1200kbps, and the current bandwidth is 2000 Kbps, then the bitrate-step (difference between the consecutive video bitrates) is 300kbps and the two thresholds are 2300 Kbps and 1700 Kbps. If two consecutive valid samples are larger

than the upper threshold, or two valid samples are below the lower threshold, the average bandwidth is set to the value of the last sample and the averaging window is reset to 2.

3.3 Adaptation Logic for TCP and MPTCP

In this section, the `adaptation logic` component is discussed in detail. It is the main component which accesses the data from other components, processes them and makes decisions for switching purposes. In addition, it defines and maintains quality metrics for decision purposes. Similar to `bandwidth estimator`, we design `adaptation logic` to be applicable to both TCP and MPTCP streaming. We first introduce quality metrics used by the `adaptation logic` and operational modes for adaptation. Then, we describe the design of the adaptation algorithms for both protocols.

3.3.1 Metrics

The metrics used by our `adaptation logic` to make bitrate switching decisions are loosely based on subjective and objective video quality metrics. The subjective quality metrics reflect the end user's experience when watching the video. Some of the example metrics are perceptual quality metrics based on modeling of the human visual system and user satisfaction [11]. In general, these metrics are hard to quantify and considered an open research topic [11]. On the other hand, objective metrics represent lower-level properties of a compressed video, such as frame rate, bitrate, video resolution and PSNR (Peak Signal to Noise Ratio). The objective metrics are simpler to quantify, compute and are often used for the design of adaptation algorithms in the literature [2]. The metrics used in our `adaptation logic` are detailed below. Tables 3.2 and 3.3 summarizes static and dynamic parameters used in our algorithm.

	Static Parameters	
Symbol	Description	Default Value
S_l	Segment length in seconds	6s
B_c	Buffer capacity	30s
α	Smoothing factor	0.5
B_δ	Negative threshold for moving to next lower video level	-0.50
$B_\bigtriangledown$	Threshold to determine the buffer end region	90%

Table 3.2: Summary of static parameter notations used in the algorithm.

	Dynamic Parameters	
Symbol	Description	Default Value
X(t)	Harmonic average of the bandwidth at time t	Dynamic
ϵ	Efficiency parameter	Dynamic
ω	Stability parameter	Dynamic
$B_s(t)$	Buffer size at time t	Dynamic
B_g	Buffer gain	Dynamic
B_{gt}	Buffer-gain threshold	Dynamic
$\triangle$B	Change in the buffer size	Dynamic
$\varphi_{total\ path}$	Total path-stability parameter (for MPTCP only)	Dynamic
φ_{thresh}	Path-stability threshold (for MPTCP only)	Dynamic
r_b	Presently-used video bitrate	Base Video bitrate

Table 3.3: Summary of dynamic parameter notations used in the algorithm.

3.3.1.1 Stability

In general, when the quality of the video changes often during a streaming session, it results in a degraded user experience [11]. To avoid this, our `adaptation logic` defines '*stability*', a subjective-quality related parameter that represents the stability of the video quality over a window of received video segments. Although the window size can be defined arbitrarily, for simplicity, we use a stability window size of 10 segment periods. The intuition behind the stability parameter is that the user tends to remember recent changes of video quality better than the older ones. The stability factor is computed as:

$$\omega = 1 - \sum_{i=1}^{10} A[\ i\ ] * 2^{-i}, \tag{3.5}$$

where, $A[\ i\ ] = 1$ indicates whether the video bitrate changed in the segment period $i-1$ and $A[\ i\ ] = 0$ indicates that there was no change in period $i-1$.

The formula implies that the stability factor is high when there are no recent changes and its value decreases exponentially with the every change of the video bitrate. For example, if the video quality has changed during the last segment request, then the stability factor is reduced by 1/2 of the previous value and if it has changed in last 2 segment periods, it is reduced by 3/4 of the previous value.

3.3.1.2 Efficiency

Efficiency is an objective-quality related parameter that determines how much of the available bandwidth is used by our streaming session. It is the ratio of the presently used video bitrate to the harmonic average of the available bandwidth and is given as:

$$\epsilon = \frac{r_b}{X(t)} \,, \tag{3.6}$$

where, r_b is the presently used video bitrate and $X(t)$ is the harmonic average of the bandwidth at time t.

A low value of efficiency means that the available bandwidth is underutilized and a high value corresponds to an optimal (*efficient*) utilization of the available bandwidth. The range of values for the efficiency parameter is as follows. At any instant in time, the used video bitrate cannot be higher than the available bandwidth. Therefore, our adaptation aims to limit the efficiency values to the range between 0 and 1. Low efficiency values indicate that the available bandwidth is not used optimally. In turn, efficiency values close to 1 indicate that the bandwidth is used efficiently. However, we note that during a streaming session, if the available bandwidth is reduced abruptly, the adaptation logic may take time to move to lower video bitrates. Thus, until the requested video bitrate is less than the available bandwidth, the efficiency may become greater than 1, which indicates over-utilization.

3.3.1.3 Buffer Gain

The amount of data stored in the streaming buffer represents a safety margin against streaming interruptions, which negatively affect the use experience. During a streaming session, due to variations of the available bandwidth, the buffer size varies over time. The *buffer gain* metric represents the relative gain or loss in buffer size during subsequent segment requests. A positive buffer gain represents that buffer size has increased between subsequent segment requests and a negative value indicates that the buffer size is reduced. Buffer gain is calculated with respect to the used segment duration, rather than the buffer capacity. It is given as:

$$B_g = \frac{\triangle B}{S_l}, \tag{3.7}$$

where B_g is the buffer gain, $\triangle B$ is the change in buffer size and S_l is the used segment length in seconds.

We explain this parameter with an example. Consider a scenario where a streaming session has a buffer size of 20s and the client requests the next segment of duration 6s. In an ideal case, if the available bandwidth matches the requested video bitrate, the buffer value should remain constant at 20s, as the transfer time of the segment is 6s and is identical to the amount of data read and displayed by the client. However, if the

available bandwidth is higher, it takes less time to transfer the segment, which would result in an increase in buffer size. For example, if the available bandwidth is such that it only takes 3.5s to download the 6s segment, we achieve a buffer gain of 2.5s. In practice, when the client buffer is near the capacity limit, the dynamic range of buffer gain is decreased, which makes the adaptation decisions more difficult. In this case, we apply the following heuristic - when the buffer size is more than 90% of the capacity, the buffer gain is increased tenfold.

3.3.1.4 Buffer-gain Threshold

In practice, the adaptation logic implements a trade-off between the stability and efficiency. The exact trade-off is application scenario-dependent. For the combined use of the stability and efficiency parameters, our `adaptation logic` defines a parameter called *buffer-gain threshold*. It is a weighted average of the stability and the efficiency and given as:

$$B_{gt} = \epsilon * \alpha + (1 - \alpha) * \omega, \qquad 0 \leqslant \alpha \leqslant 1 \tag{3.8}$$

where α is the *smoothing factor*.

The parameter α balances the importances of the stability and efficiency for adaptation during the streaming session. It is meant to be set after the exact application scenario is known. After the value of α is set at the beginning of the session, it remains constant till the end.

3.3.2 Operational Phases

Two design decisions guide our implementation of the adaptation logic. First, at any instant in time, the adaptation logic is limited to switch to adjacent video levels only (either the next higher or lower level, without skipping a level). Since the quality changes in adjacent video levels are gradual, the user experiences smooth transitions. Second, the adaptation logic operates in two phases based on the instantaneous buffer size. We refer to these phases as *startup phase* and *steady phase*. The adaptation switches between the two phases dynamically. These two phases are explained next.

Startup phase : The `adaptation logic` operates in the startup phase at the beginning of the streaming session or when the client buffer size is less than 50% of its capacity. If the available bandwidth is not sufficient to fill up 50% of the client buffer, the

client continues to request the base video level without considering to move to higher video levels. In all our experiments, we set the client buffer to 30s, so till the buffer size is 15s, our `adaptation logic` works in startup phase. When the client buffer size reaches more than 50% of its capacity, it moves to the steady phase.

Steady phase : The `adaptation logic` operates in the steady phase once the buffer size exceeds 50% of the buffer capacity. Using the 50% of the buffer capacity as a threshold ensures that after moving to the higher levels, if the available bandwidth drops, the `adaptation logic` has sufficient time to move to a lower video level without playout interruptions. If the buffer size goes below 50% of the capacity, the adaptation moves to the startup phase and switches to the base level. The exact adaptation logic in the steady phase depends on the *functional* mode. Our `adaptation logic` can operate in one of the three functional modes, which are detailed below.

3.3.3 Functional Modes

While operating in the steady phase, the `adaptation logic` can be configured to work in three different modes based on the value of parameter α, as given below. Since α remains constant over the session, the mode cannot change during the session. In all three modes, a switch to a higher video level is allowed only when the buffer gain is larger than buffer-gain threshold. If the buffer gain becomes less than the negative buffer-gain threshold B_δ, then the adaptation logic moves to the next lower video level. The negative buffer-gain threshold B_δ, is a constant value (set at the beginning) used by the adaptation logic to decide to lower video levels, when the buffer size reduces.

Stability mode : In this mode (α =0), the value of the buffer-gain threshold depends only on the stability factor. If the video bitrate has been changed in the last segment request, the stability parameter has a low value while the buffer-gain threshold has a high value. As a result, the `adaptation logic` requires higher buffer gains to move to the next higher video level. Additionally, it delays subsequent switching events by one or more segment periods (in all our experiments, we delay subsequent requests by one segment period). Intuitively, in this mode, the `adaptation logic` tries to minimize the number of consecutive video quality switches. In turn, this causes the switching events to be spread out wide on the time scale, thus giving a better user experience.

Efficiency mode : This mode (α =1) works exactly opposite to the stability mode, since the value of buffer-gain threshold depends only on the efficiency parameter. If

the efficiency parameter is low, the buffer-gain threshold is also low, thus favoring a move to a higher level. Intuitively, the `adaptation logic` is aggressive and tries to switch to higher levels without delays and does not consider the history of recent switches.

Mixed mode : In mixed mode, the `adaptation logic` operates in between the two extreme modes described above. The buffer-gain threshold is a weighted average of efficiency and stability, as α takes a value between 0 and 1.

3.3.4 TCP Adaptation Logic - Complete Algorithm

The Algorithm 3.1 presents the complete algorithm implemented in the `adaptation logic` for TCP protocol. Its main steps are summarized as follows.

- At the beginning, all the standard metrics like buffer-gain threshold, buffer gain, efficiency and stability are set to zero and client sets the requested video bitrate (r_b) to the lowest available bitrate. In addition, the values of other parameters like available bitrates and segment length in seconds are set by processing the MPD file. The value of the buffer end-region threshold $B_{\bigtriangledown}$ determines when the adaptation operates in the buffer end region. It is set at the beginning of the streaming session and it depends on the buffer capacity and the segment length. The $B_{\bigtriangledown}$ needs to be configured appropriately based on the client buffer capacity. For example, when the buffer capacity is very high, e.g., around 120 seconds, setting $B_{\bigtriangledown}$ to 0.95 would be appropriate. However, when the buffer capacity is very small, e.g., around 10 seconds, setting $B_{\bigtriangledown}$ to 0.9 is considered appropriate. In all our experiments, we set the buffer capacity to 30 seconds and $B_{\bigtriangledown}$ to 0.9. Before requesting a new segment, the `adaptation logic` makes decisions in sequence based on the metrics values as given in Figure 3.6. These decisions are given in the following.

- If $B_s(t) < 0.5 * B_c$, then `adaptation logic` operates in the startup phase and the r_b is set to the base video bitrate. It continues to request segments of the base bitrate until it moves to steady phase. When the buffer size becomes more than 50% of the buffer capacity ($B_s(t) > 0.5 * B_c$), the `adaptation logic` enters the steady phase and sets the next video bitrate depending on the buffer gain B_g and buffer-gain threshold B_{gt}. If the buffer gain is larger than the buffer-gain threshold, it indicates that the network conditions are favorable and higher video bitrates are supported. If the B_g is not sufficient, then B_g gets accumulated over subsequent segment requests and client continues to requests the video on the same level. If the B_g is less than negative threshold B_δ, it indicates that the buffer size is getting

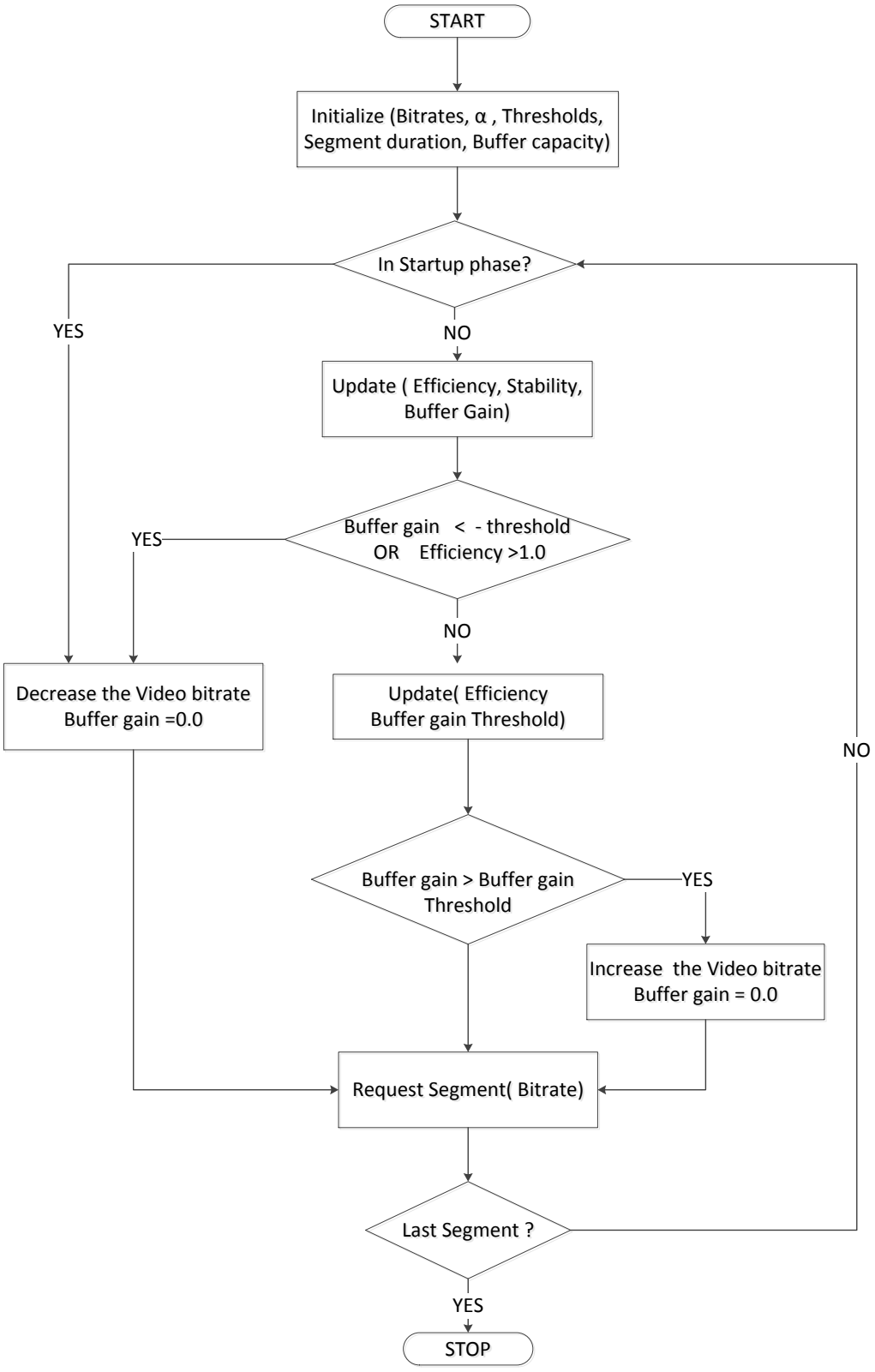


Figure 3.6: TCP adaptation logic decision control sequence.

Algorithm 3.1 Adaptation algorithm for TCP

1: $Adaptation\_TCP\ (\ r[b],\ B_c,\ \alpha,\ S_l\ , B_\delta\ , B_\bigtriangledown\ )$

2: **for** $i = 1 : N_{seg}$ **do**
3: **if** $(\ B_s(t)\ <\ \frac{B_c}{2}\ )$ **then** ▷ In startup phase?
4: $r_b\ =\ Decrease(\ r[b]\ )$
5: **go to** 33

6: **else** ▷ In steady phase ?
7: $\epsilon \leftarrow \frac{r_b}{X(t)}$
8: $Update\ (\omega)$
9: **if** $B_s(t)\ >\ B_c * B_\bigtriangledown$ **then**
10: $B_g \leftarrow\ 10 * \frac{\triangle B}{S_l}\ + B_g$
11: **else**
12: $B_g\ \leftarrow \frac{\triangle B}{S_l}\ + B_g$
13: **end if**

14: **if** $(\ \epsilon >\ 1.0\ \vee\ B_g\ <\ -B_\delta\ )$ **then**
15: $r_b\ =\ Decrease(\ r[b]\ )$
16: $B_g \leftarrow 0.0$
17: **end if**

18: **if** $(\ B_s(t)\ >\ B_c * B_\bigtriangledown\ \wedge\ B_g\ <\ -1.5 * B_\delta)$ **then**
19: $r_b\ =\ Decrease(\ r[b]\ )$
20: **end if**

21: **if** $(\ \epsilon < 1.0 \wedge\ \epsilon > 0.5\ )$ **then**
22: $\epsilon \leftarrow 1 - \epsilon$
23: **end if**

24: $B_{gt} \leftarrow (\epsilon\ *\ \alpha\ +\ (\ 1 - \alpha\ )\ *\ \omega)$

25: **if** $B_g\ >\ B_{gt}$ **then**
26: **if** $X(t)\ >\ Increase(\ r[b]\ )$ **then**
27: $r_b\ =\ Increase(\ r[b]\ )$
28: $B_g \leftarrow 0.0$
29: **end if**
30: **end if**

31: $Request(\ r_b\ , N_i)$
32: **end if**
33: **end for**

reduced during the subsequent requests due to unfavorable network conditions and the requested video bitrate needs to be decreased. As a result, the `adaptation logic` reduces the r_b to the next lower level.

- The efficiency factor ϵ determines the aggressiveness of the `adaptation logic` and

its value should always favor switching to higher levels in efficiency mode under sufficiently high buffer gains. For example, while in efficiency mode, when the efficiency is 0.75, in order to switch to the next higher level, buffer gain should be more than 0.75. While operating at higher efficiencies, achieving such higher buffer gains requires more time. To overcome this, the `adaptation logic` modifies the efficiency to lie always between 0.0 and 0.5. With small efficiency values, less buffer gain is required, thus favoring a switch to the next higher level.

- When the $B_s(t) > B_c * B_{\bigtriangledown}$, then the client is operating at the end region of the buffer. As a result, the dynamic range of the buffer gain is reduced at this instant. To overcome this, while operating in the end region of the buffer, our `adaptation logic` increases the buffer gain by ten-fold given as $B_g = B_g + 10 * \frac{\triangle B}{S_l}$.

Based on the above sequence of decisions, the `adaptation logic` updates the r_b and the used metrics and requests the next segment. This sequence of decisions are continued until the last segment is reached.

3.4 Path Stability Parameter in MPTCP Adaptation Logic

The proposed bandwidth estimation and adaptation logic can be readily applied to adaptive streaming over TCP. However, our preliminary experiments showed that the throughput over an MPTCP connection is more bursty. Namely, we have noticed that the burstiness increases with increase in the number of subflows, packet loss and delay. As multiple paths are used in an MPTCP connection, even a bandwidth variation on one of the paths results in a variation of total bandwidth. These variations become worse when all used paths are unstable, resulting in frequent variations. In turn, the adaptation experiences more switching events over time.

To mitigate this effect for MPTCP streaming, a modification of `adaptation logic` is needed. In the sequel, we first illustrate the influence of packet loss and delay on the measured bandwidth over an MPTCP connection. Then, we describe the adaptation logic for MPTCP streaming.

3.4.1 Experiments with Delay and Packet Loss

3.4.1.1 Influence of Delay on MPTCP Throughput

Generally, a TCP connection over a high delay and high packet-loss path experiences less throughput and more fluctuations in the available bandwidth, mainly due to the

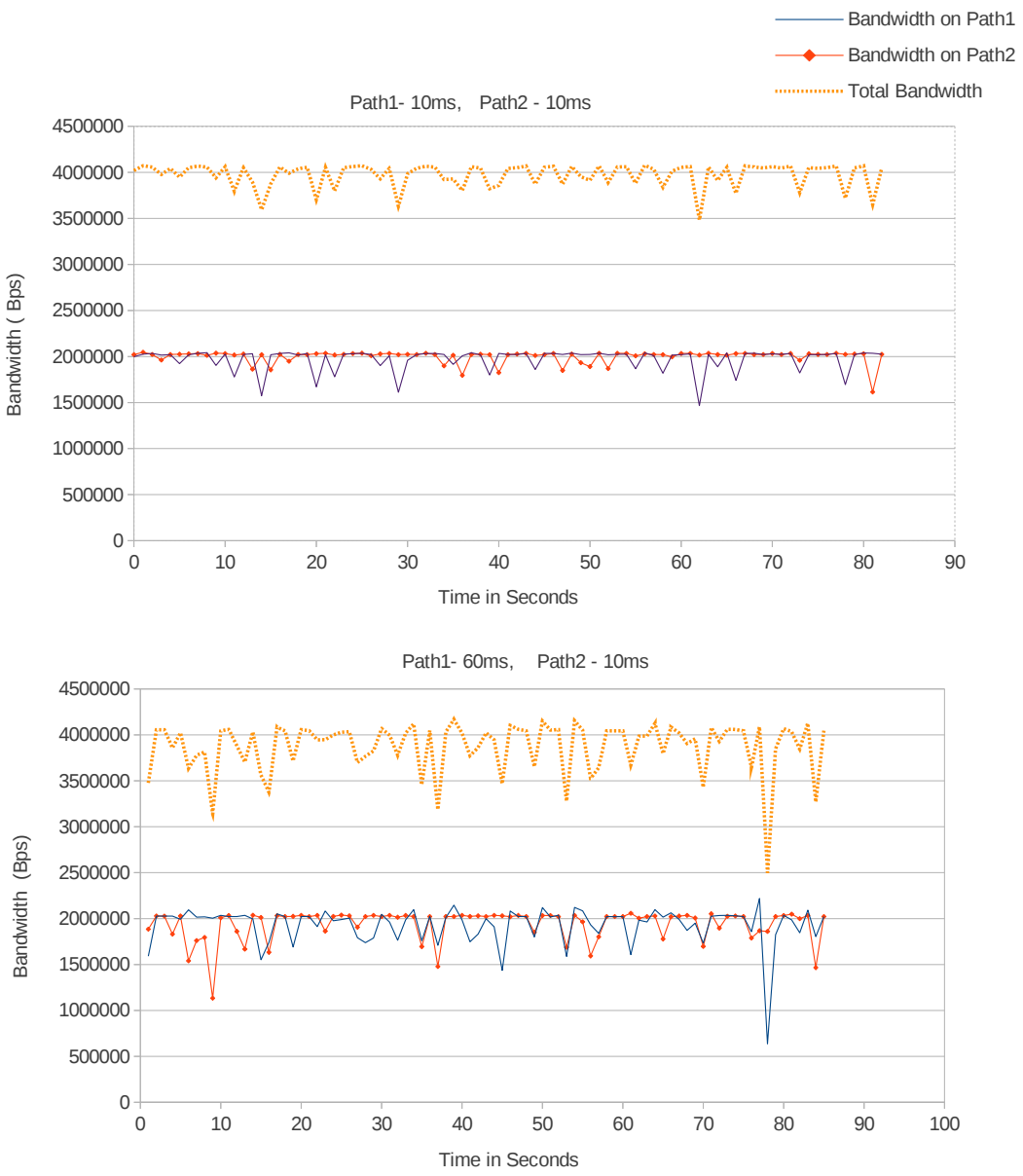


Figure 3.7: Influence of low delays on MPTCP throughput.

congestion control algorithm. TCP's congestion window experiences decrease events due to packet losses and grows slowly under high round trip times. As a result, TCP operates with small congestion windows and the amount of data transferred is less, resulting in a low measured throughput. Since MPTCP congestion control is derived from the TCP's, we can expect a similar effect on MPTCP throughput.

To study MPTCP behavior under high delays and high packet losses, we performed experiments involving a file transfer over an MPTCP connection. File transfer is chosen since it can saturate the link capacity such that the exact behavior of the congestion window and the bandwidth variations can be observed.

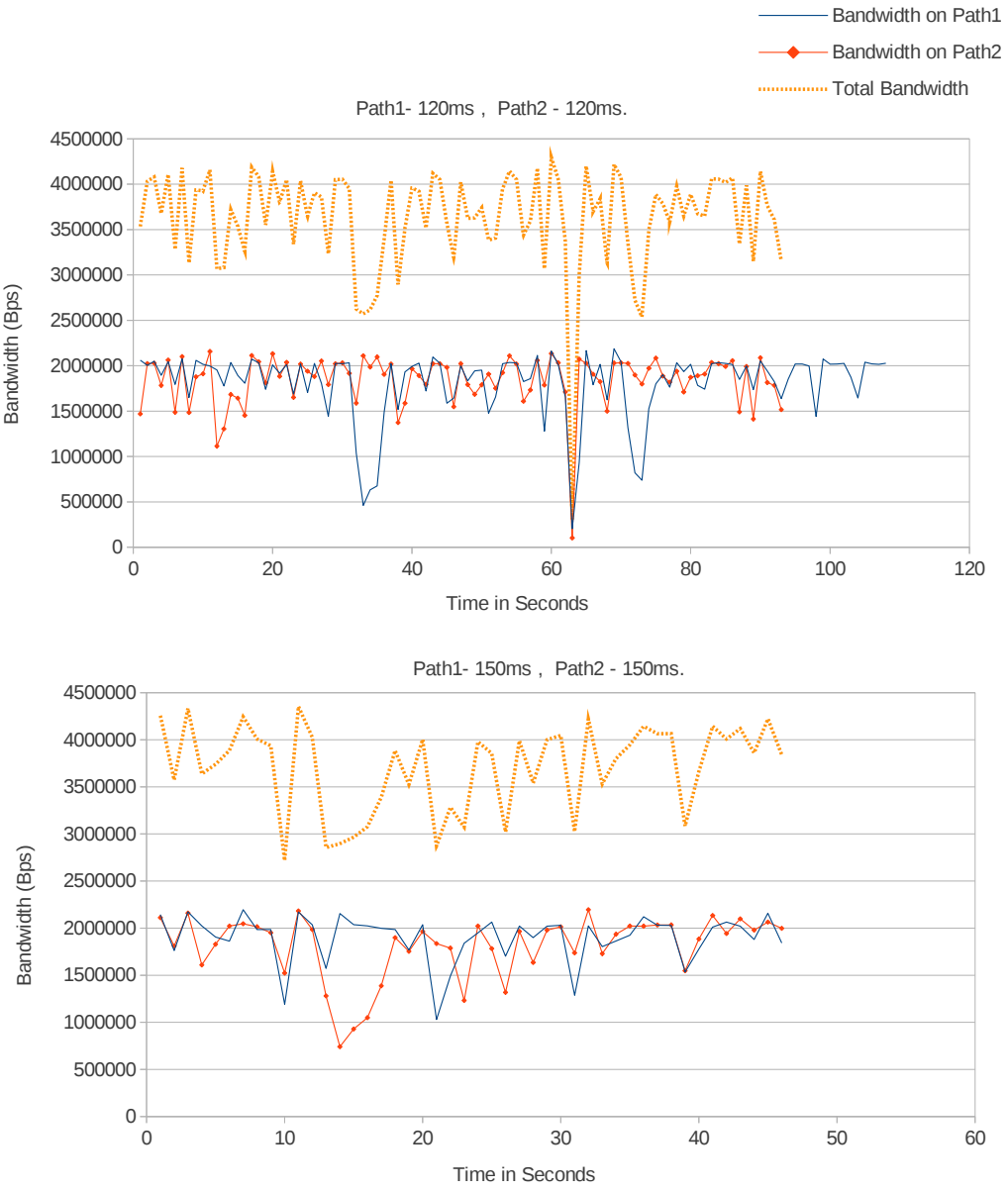


Figure 3.8: Influence of high delays on MPTCP throughput.

In our high-delay experiments, we use two paths and the available bandwidth is kept constant (2000 Kbps on each path) for the whole session. Results under different path conditions are shown in Figures 3.7 and 3.8.

- Under low delay (10ms) conditions, throughput over the paths remains constant with small variations. These small variations are a result of congestion window reaching the maximum value and experiencing decrease events. Both paths exhibit the same behavior.
- When the delay is increased to 60ms on one path, larger throughput variations are visible. Since the delay is increased, the congestion window takes more time to

grow after a decrease event. As a result, the congestion window remains at lower values for a longer time. Since the throughput on one path varies more, the total throughput also varies more.

- Next, when the delay is increased to very high values around 120ms on one path, throughput variations are large. The congestion window takes more time to move to higher values and wide negative spikes are seen. The average throughput is 0.25 Mbps smaller than the available bandwidth.
- In the last scenario, both paths are operating under high delay conditions (150ms). As seen, the available bandwidths on both paths are more unstable. As a result, total throughput is more unstable. The average throughput is 0.5 Mbps smaller than the available bandwidth. In general, this shows that the amplitude of throughput variations increases with increase in delay.

3.4.1.2 Influence of Delay and Packet Loss on MPTCP Throughput

In this section, we perform experiments to illustrate the joint impact of delay and packet loss on MPTCP throughput. The packet loss rate is 0.01 and the delay is varied in the experiments. The results are as shown in Figures 3.9 and 3.10.

- Under low delay conditions, the high packet loss rate causes frequent negative spikes. In these cases, packet losses cause the congestion window to be reset.
- With increase in delay, the congestion window grows slowly. Also, frequent packet losses cause the congestion window to reset often. This causes the congestion window to remain at lower values. As a result, the throughput remains smaller than the actual available bandwidth most of the time.
- The situation worsens when the delay along the path doubles and the throughput is less than the half of the available bandwidth. However, the differences between the neighboring samples are very small. As a result, the amplitude of variations are also small.
- The last scenario represents packet losses on the low delay path and the no packet loss on the high delay path. As can be seen from the graph, frequent fluctuations on high packet loss path cause the throughput to vary.

From the experiment, we conclude that frequent packet losses in any of the paths cause a low MPTCP throughput and high throughput variations. Since MPTCP uses multiple paths in parallel, the total throughput varies more in terms of amplitude and frequency.

Figure 3.9: Influence of delay and packet loss on MPTCP throughput.

3.4.2 MPTCP Adaptation Logic - Complete Algorithm

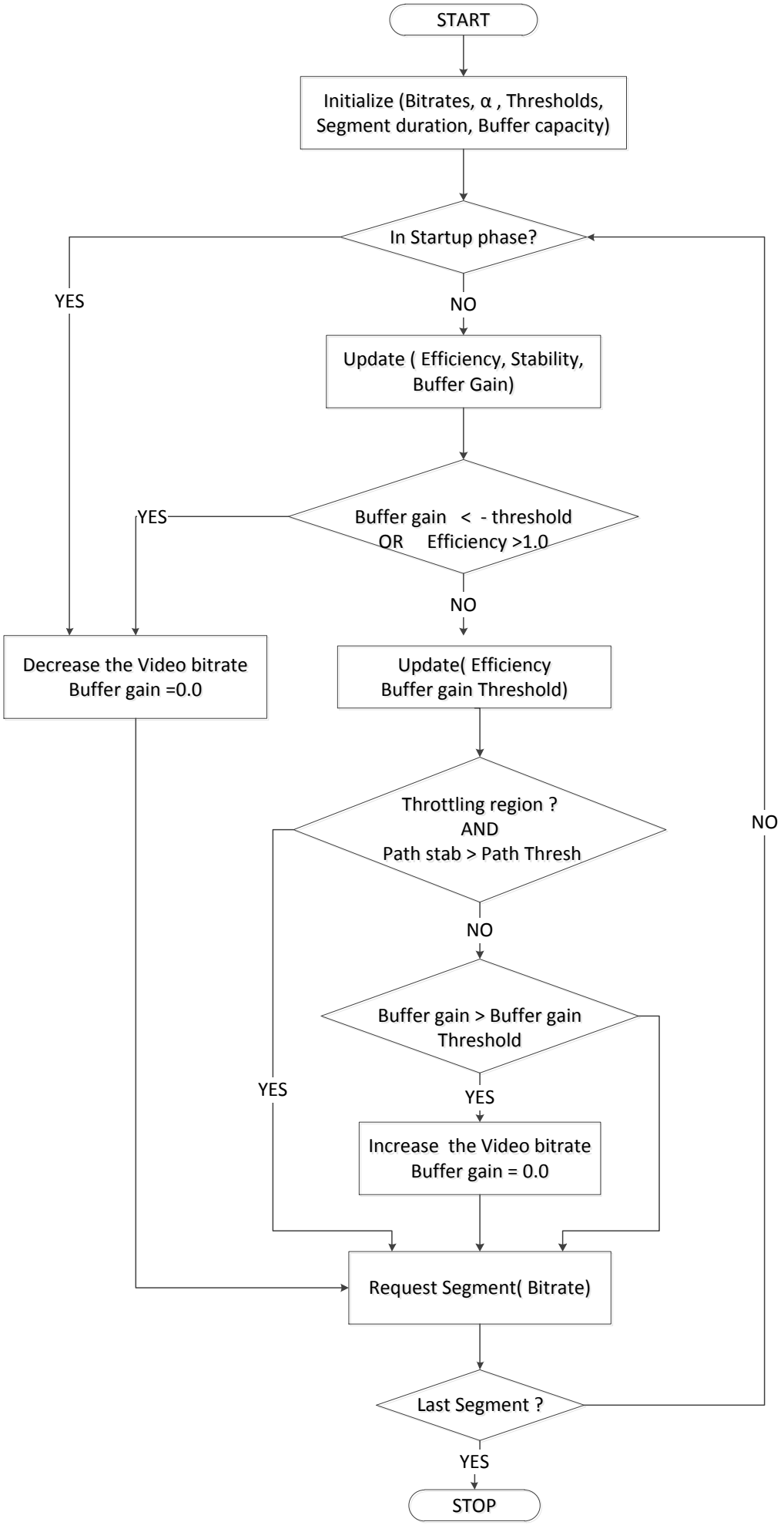
To overcome the burstiness of MPTCP, the `adaptation logic` for MPTCP uses an additional operational parameter called path-stability (φ_{path}). Path-stability on each path represents the instability (variations) of the bandwidth and the total path-stability ($\varphi_{total\ path}$) of all paths represents the instability of the whole MPTCP connection. Path-stability represents the average of the square differences between the consecutive measured bandwidth samples, as given in Equation 3.9.

$$\varphi_{path} = \frac{\sum_{i=1}^{n} (x_i - x_{i-1})^2}{n-1}, \qquad n = 10 \tag{3.9}$$

Figure 3.10: Influence of delay and packet loss on MPTCP throughput.

To effectively use the path-stability parameter for adaptation, the MPTCP `adaptation logic` first divides the operating bandwidth region into two regions:

1. **Stable region:** Stable region refers to the operating region where the requested video level is one level lower than the maximum level that can be achieved under a


START
Initialize (Bitrates, α , Thresholds, Segment duration, Buffer capacity)
In Startup phase?
YES
NO
Update (Efficiency, Stability, Buffer Gain)
Buffer gain < - threshold OR Efficiency >1.0
YES
NO
Decrease the Video bitrate Buffer gain =0.0
Update(Efficiency Buffer gain Threshold)
Throttling region ? AND Path stab > Path Thresh
NO
NO
Buffer gain > Buffer gain Threshold
YES
YES
Increase the Video bitrate Buffer gain = 0.0
Request Segment(Bitrate)
Last Segment ?
YES
STOP

Algorithm 3.2 Adaptation Algorithm for MPTCP

1: $Adaptation\_MPTCP\ (\ r[b], B_c, \alpha, S_l, B_\delta, B_\bigtriangledown, \varphi_{thresh})$

2: **for** $i = 1 : N_{seg}$ **do**
3: **if** $(\ B_s(t)\ <\ \frac{B_c}{2}\ )$ **then** ▷ In startup phase?
4: $r_b\ =\ Decrease(\ r[b]\ )$
5: **go to** 33

6: **else** ▷ In steady phase ?
7: $\epsilon \leftarrow \frac{r_b}{X(t)}$
8: $Update\ (\ \omega\ )$
9: **if** $B_s(t)\ >\ B_c * B_\bigtriangledown$ **then**
10: $B_g \leftarrow\ 10 * \frac{\triangle B}{S_l}\ + B_g$
11: **else**
12: $B_g\ \leftarrow \frac{\triangle B}{S_l}\ + B_g$
13: **end if**

14: $Update\ (\ \varphi_{total\ path}\ )$
15: **if** $(\ \epsilon >\ 1.0\ \vee\ B_g\ <\ -B_\delta\ )$ **then**
16: $r_b\ =\ Decrease(\ r[b]\ )$
17: $B_g \leftarrow 0.0$
18: **end if**

19: **if** $(\ B_s(t)\ >\ B_c * B_\bigtriangledown\ \wedge\ B_g\ <\ -1.5 * B_\delta)$ **then**
20: $r_b\ =\ Decrease(\ r[b]\ )$
21: **end if**

22: **if** $(\ \epsilon < 1.0 \wedge\ \epsilon > 0.5\ )$ **then**
23: $\epsilon \leftarrow 1 - \epsilon$
24: **end if**

25: $B_{gt} \leftarrow (\epsilon\ *\ \alpha\ +\ (\ 1 - \alpha\ )\ *\ \omega)$

26: **if** $(\ B_g\ >\ B_{gt})\ \vee\ (\ R_l == 1\ \wedge\ \varphi_{total\ path}\ < \varphi_{thresh}\ )$ **then**
27: **if** $X(t)\ >\ Increase(\ r[b]\ )$ **then**
28: $r_b\ =\ Increase(\ r[b]\ )$
29: **end if**
30: **else**
31: $B_g \leftarrow 0.0$
32: **end if**

33: $Request(\ r_b\ , N_i)$
34: **end if**
35: **end for**

given available bandwidth. For example, if the maximum achievable level is L_n, then the region from L_1 to L_{n-1} is considered as the stable region.

2. **Throttling region:** This region refers to the region between levels L_{n-1} and L_n. When operating in this region, large variations of available bandwidth cause frequent level switches (between levels L_{n-1} and L_n).

$$\varphi_{total\ path} = \sum_{i=1}^{no\ of\ paths} \varphi^2_{path}[i], \tag{3.10}$$

To avoid frequent level switches in the throttling region, the `adaptation logic` moves to higher video levels only if the total path-stability is smaller than path-stability threshold (φ_{thresh}). Smaller path-stability values minimize the number of switching events. If the total path-stability exceeds the path-stability threshold, switching to higher levels is disabled. The path-stability threshold is defined at the beginning of the session and depends on the average of the differences between available video bitrates. In our `adaptation logic`, the φ_{thresh} is set to four times the average difference between available video bitrates.

The algorithm that implements MPTCP `adaptation logic` is shown in Figures 3.11 and 3.2. As seen, the algorithm includes the path-stability parameter. If the streaming system is operating in the throttling region, the decision to use the next higher level is taken through a separate branch.

3.5 Summary

In this chapter, each component of the proposed multi-path streaming was introduced in detail. In the next chapter, we conduct experiments to evaluate each of the components with respect to their functionality and performance.

Chapter 4

Experiments and Results

In this chapter, we present the experimental testbed, the dataset and the experimental evaluation of each of the components of our system. Our evaluation studies the effectiveness of the components under different network conditions in terms of packet loss, delay and available bandwidth. Later, we introduce the evaluation of streaming over MPTCP compared to over TCP under different network conditions. We begin this chapter with a description of the datasets and experimental setup in Section 4.1 and 4.2. We present the experiments and their results in later sections.

4.1 DASH Dataset

For an effective evaluation of each of the components of the system, we used the DASH dataset complying to DASH standard. The dataset used in our experiment is the video sequence 'Big Buck Bunny', which is an animation movie with horizontal resolution of 1080 pixels and 574 seconds duration, available at [36]. Sequence duration is long enough to test our `adaptation logic` efficiently, without replaying the video. Video is segmented with durations of 1, 2, 4, 6, 10, 15 seconds and coding rate ranges from the minimum of 50 Kbps to the maximum of 3.87 Mbps, as shown in Table 4.1. The sequence is encoded and segmented using the DASH encoder available at [21]. For most of our experiments we used video profiles with bitrate more than 2 Mbps as given in Table 4.2, to reflect the real streaming rates over the Internet. The dataset also includes other video sequences of different genres like sports and movies [36]. These can be used to test the performance of our `adaptation logic` for different video content types. However, to maintain uniformity of the dataset in our experiments, their use is limited in this report.

Bitrate	Dimension
50 Kbps	320x240
100 Kbps	320x240
.....	.....
1.2 Mbps	1280x720
1.5 Mbps	1280x720
.....	.....
3.55 Mbps	1980x1080
3.85 Mbps	1980x1080

Table 4.1: Available bitrates of the dataset.

Bitrate	Dimension
2.07 Mbps	1980x1080
2.39 Mpbs	1980x1080
2.91 Mbps	1980x1080
3.29 Mbps	1980x1080
3.55 Mbps	1980x1080
3.85 Mbps	1980x1080

Table 4.2: Multiple bitrates used in the experiments.

4.2 Experimental Setup

Our testbed is implemented to support experiments under both TCP and MPTCP. The implemented testbed is shown in Figure 4.1. Its main components are as follows:

- *Server* is installed with Ubuntu 12.04 Linux (64 bit) system, as MPTCP is supported only on 64-bit Linux operating systems [26]. Apache web-server is installed on sender machine to support HTTP requests over TCP and MPTCP. Web server can be configured to use both persistent and non-persistent connections. Server stores videos of different segment durations, each encoded at multiple bitrates. In addition, server stores the MPD files of all the videos.
- Videolan player with DASH plugin (vlc-DASH client [36]) is installed on the *client* machine running Ubuntu 12.04 Linux (64 bit) system with MPTCP installed. The vlc-DASH client is configured to log system parameters periodically, which will be used for performance evaluation.

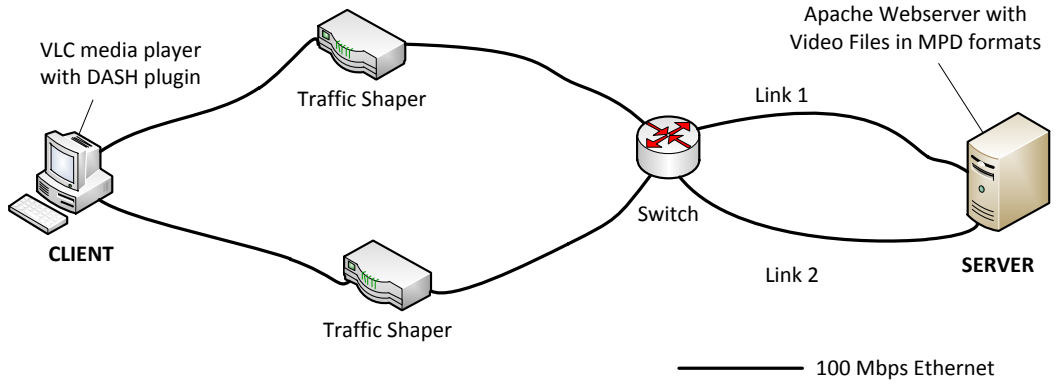


Figure 4.1: Experimental setup for adaptive video streaming.

- Client and server are connected using two links, to enable emulation of two different paths. Capacity of each link is 100 Mbps. On each link, a traffic shaper is present (Dummynet installed on a separate Linux machine), which is used to vary the path properties like bandwidth, delay and packet-loss rate.

4.3 Validation of Basic MPTCP Functionalities

In this section we perform two sets of preliminary experiments. First, we validate the basic properties of MPTCP, such as the protocol's ability to balance the streaming load over multiple available paths. Second, we evaluate the impact of segment duration in MPTCP streaming. These two experiments serve the purpose of demonstrating the suitability of our testbed for MPTCP streaming and empirically determining the suitable segment duration for streaming, respectively.

4.3.1 MPTCP Multipath Support

As explained in Chapter 2, MPTCP is designed to use multiple paths and balance the application load over those paths. As a result, it also supports failure resilience by continuing traffic on the active paths when any of the paths goes down.

Correspondingly, our evaluation tests the video streaming over MPTCP, the load balancing and failure resilience. In this experiment, we perform bandwidth measurements on the client side. The client implements `adaptation logic` introduced in Section 3.4.2. Results of measured bandwidth values are shown in Figure 4.2. Segments of size 6s were used during the whole session. Buffer capacity at the vlc-DASH client was 30s for the whole session. The round trip time is set to 60ms and packet loss rate is zero for the whole duration of the session. From the start of the session until 100s the available bandwidth on path 1 is 3500 Kbps and on path 2 is 1500 Kbps. At time t=100s, the available bandwidth is abruptly decreased to 2000 Kbps on path 1 and increased to 2000 Kbps on path 2. At t=160s, path 1 goes down and the bandwidth over path 2 is increased to 2500 Kbps. At t=220s, path 1 is restored and both paths continue with a bandwidth of 2500 Kbps till time t=280s. At time t=280s, path 2 goes down and path 1 continues with bandwidth of 2500 Kbps till the end of the session. At time t=340s, path 2 is restored and continues with bandwidths of 2000 Kbps, 1500 Kbps, 1750 Kbps and 1250 Kbps for intervals of 60s till the end of the session.

Takeaway

From the experiment we can conclude that:

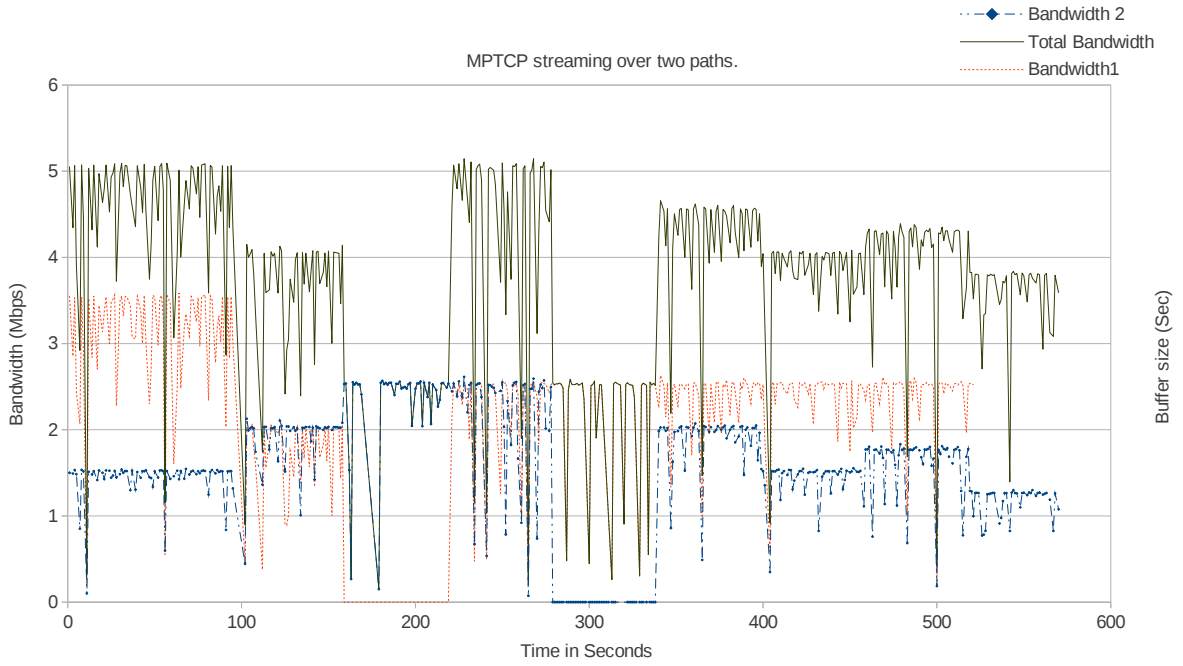


Figure 4.2: Capacity increase and failure resilience using MPTCP.

1. Our testbed supports video streaming over MPTCP without any changes to the streaming application.

2. MPTCP supports the failure resilience as expected and the streaming continues on active paths without stopping the session.

3. With the use of multiple paths, the total capacity of the connection has been increased compared to use of single path and is equal to the sum of capacities of the used paths (as evidenced by the Total Bandwidth plot in Fig. 4.2).

4.3.2 Selection of Segment Duration for MPTCP Streaming

The segment size is an important parameter for a streaming system. For adaptive streaming that uses individual segments, the average bitrate achieved during the session is proportional to the sum of data transferred in the lifetime of session.

To test the impact of different segment sizes on MPTCP streaming, our dataset consists of segments of 1, 2, 4, 6, 10, 15 seconds in duration. Server and client are connected with two links - link 1 and link 2 - with fixed bandwidth of 2000 Kbps. Buffer capacity of the vlc-DASH client is set to 30s for the whole session. We used the persistent connection mode, such that a single HTTP connection is maintained and used for every new segment request during the session. We use the default `adaptation logic` available in the vlc-DASH plugin.

Figure 4.3: Impact of segment duration on MPTCP streaming.

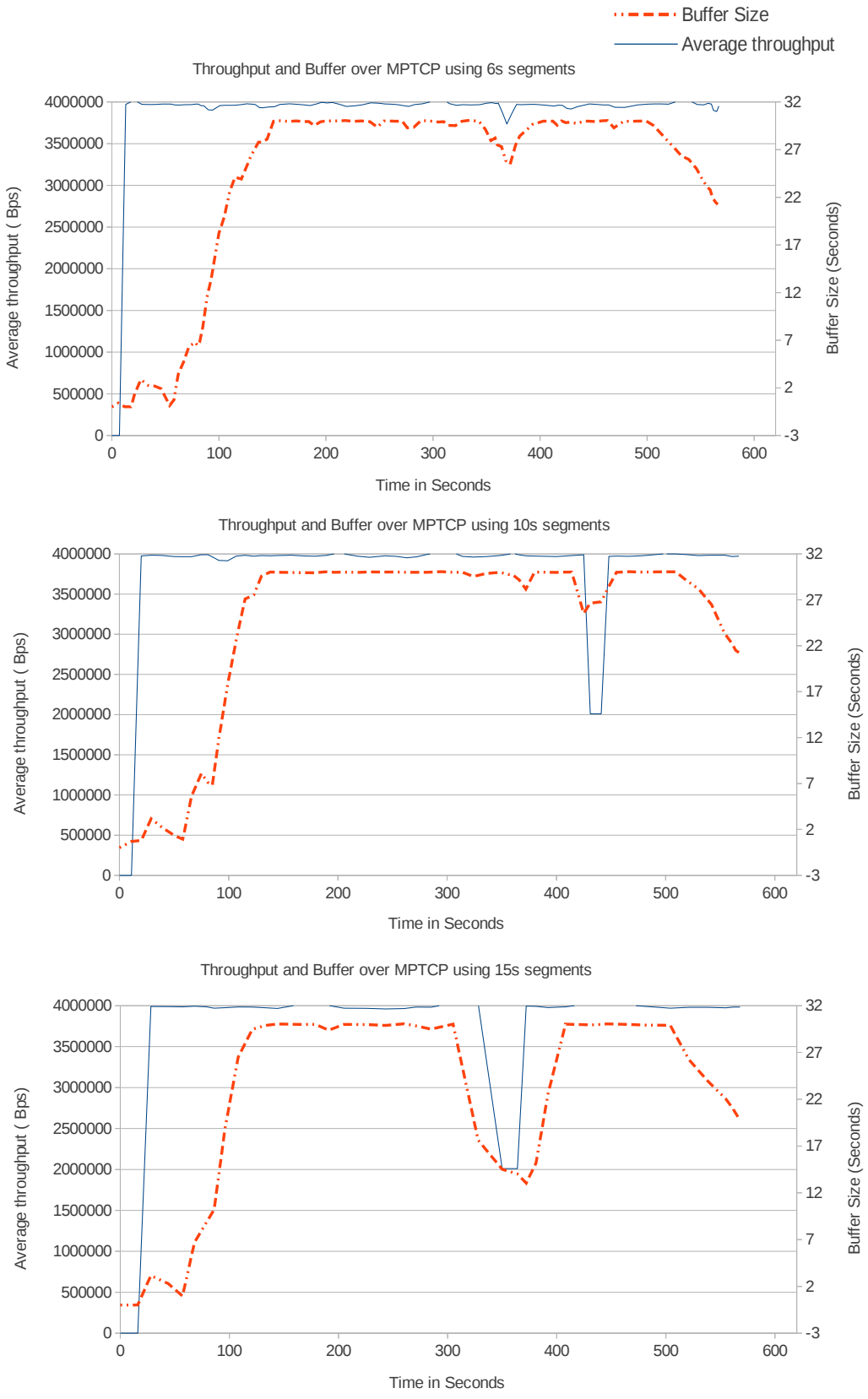


Figure 4.4: Impact of segment duration on MPTCP streaming.

Results are shown in Figure 4.4. Use of 1s segments results in a low and highly varying throughput. The buffer size remains below 2 seconds. With 2s segments, the achieved throughput is higher and the buffer reaches the full capacity. However, the average throughput still varies. Similar behavior can be observed for segments sizes of 4s. In contrast, segments of size 6s achieve a nearly constant average throughput. The buffer size using 6s segments is also higher. By using yet larger segment sizes (10s and 15s), we can achieve a very high average throughput. However, as the adaptation is performed at longer timescales, the `adaptation logic` can no longer closely match the available bandwidth. This occasionally results in large throughput drops, as shown in Figure 4.4.

The results in Figure 4.4 demonstrate that the segment size of 6s strikes a good balance between the average throughput and the throughput variability. As a result, we consider 6s segments as optimal for streaming over MPTCP. For the remainder of this report, in all the experiments we use 6s segments. Further, our experiments will use 6s segments with persistent connections, since the use of a non-persistent connection would require the congestion window to start from low values with each new connection.

4.4 Evaluation of Implemented System Components

4.4.1 Bandwidth Estimator

In this section, we evaluate performance of the `bandwidth estimator` described in Section 3.2.3. To this end, we implement the `bandwidth estimator` as a part of the vlc-DASH client application. For brevity, we refer to this implementation as "modified vlc-DASH client". In this section, we first evaluate the performance of the bandwidth measurement described in Section 3.2.2. We present results for both the TCP streaming and the MPTCP streaming. Next, we evaluate the harmonic averaging method for bandwidth estimation, as proposed in Section 3.2.5. We perform this evaluation for MPTCP streaming only. Finally, we evaluate the averaging performance in the presence of bandwidth spikes.

4.4.1.1 Bandwidth Measurement over a TCP Connection

We test the bandwidth measurement on the client side using the modified vlc-DASH plugin over a TCP connection. Similar to our earlier experiments, we use the segment size of 6s and the buffer capacity in vlc-DASH client is set to 30s. We vary the available bandwidth using the dummynet traffic shaper. We use dummynet to limit the available bandwidth as follows. The round trip time is set to 60ms and packet loss rate to zero for

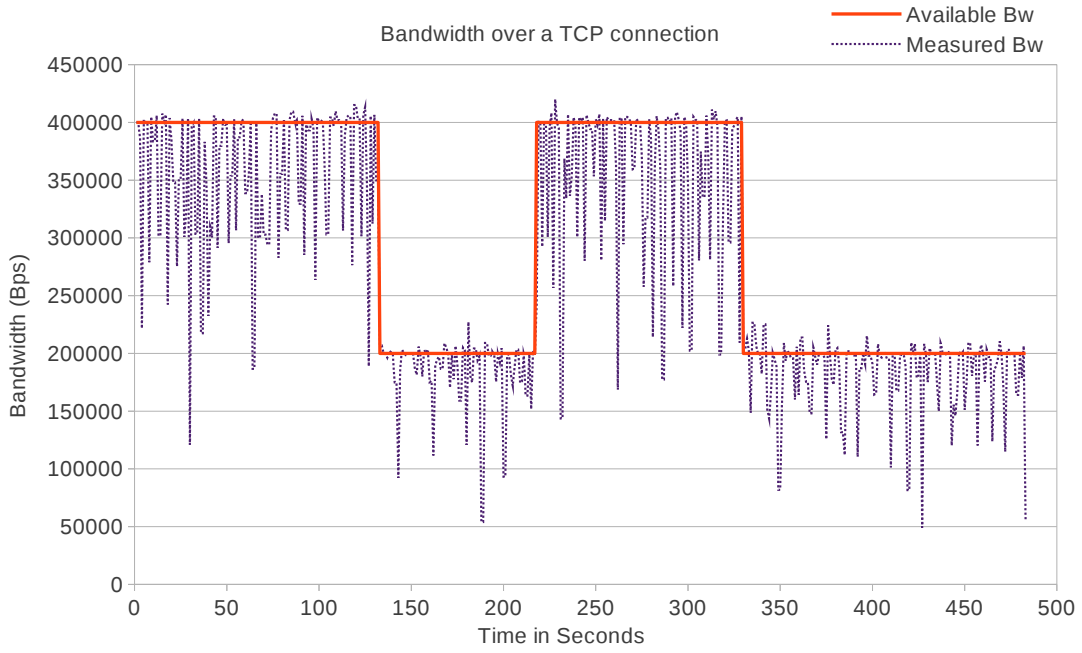


Figure 4.5: Bandwidth measurement using modified vlc-DASH client over TCP.

the whole duration of the session. From the start of the session until time t=125s the available bandwidth is 400 Kbps. At time t=125s, the available bandwidth is abruptly decreased to 200 Kbps followed by increase back to 400 Kbps at time t=225s. Finally, at time t=330s the bandwidth is decreased back to 200 Kbps till the end of the session.

The obtained measured bandwidth values are shown in Figure 4.5, alongside the actual available bandwidth (ground truth). We see that at any instant of time, the measured bandwidth samples approximately match the actual available bandwidth. The large variations in the measured bandwidth visible in Figure 4.5 do not affect the accuracy of our measurement. As explained in Section 3.2.4, the vlc-DASH plugin requests the segments in discrete time intervals such that the drops in the measured bandwidth reflect the idle periods during which no data is transferred. Figure 4.5 also shows that some measured samples are 1-2% above the actual bandwidth values. This is due to the unpredictable timing of the threads in the operating system that affects the sampling instants in practice. The effect of this inaccuracy on the `adaptation logic` is negligible, as they will be overruled by the sampling and averaging mechanism, as it will be shown in our subsequent evaluations.

Takeaway

From the above experiment we conclude that our bandwidth measurement implementation based on Pcap gives accurate results.

4.4.1.2 Bandwidth Measurement over an MPTCP Connection

Our MPTCP evaluation tests the bandwidth measurement under varying bandwidth conditions over multiple paths. In this context, multiple paths correspond to multiple interfaces of the client machine. Our experiment is similar to the one explained in Section 4.4.1.1. We have used 6s segments during the whole session and the round trip time is set to 60ms. On each path the available bandwidth is set to 2000 Kbps from the beginning of the session till time t=130s. At time t=131s the available bandwidth is reduced to 1000 Kbps for the next 125 seconds. Then, at time t=255s available bandwidth is increased to 2000 Kbps till the end.

The measured bandwidth samples are shown in Figure 4.6. Our measurement approach uses Pcap to capture packets arriving at two network interfaces in parallel. Therefore, Figure 4.6 shows two sets of samples, one for each network interface. For completeness, we perform a summation of these samples and represent the total available bandwidth of the MPTCP connection with the "Total Bandwidth" curve in Figure 4.6.

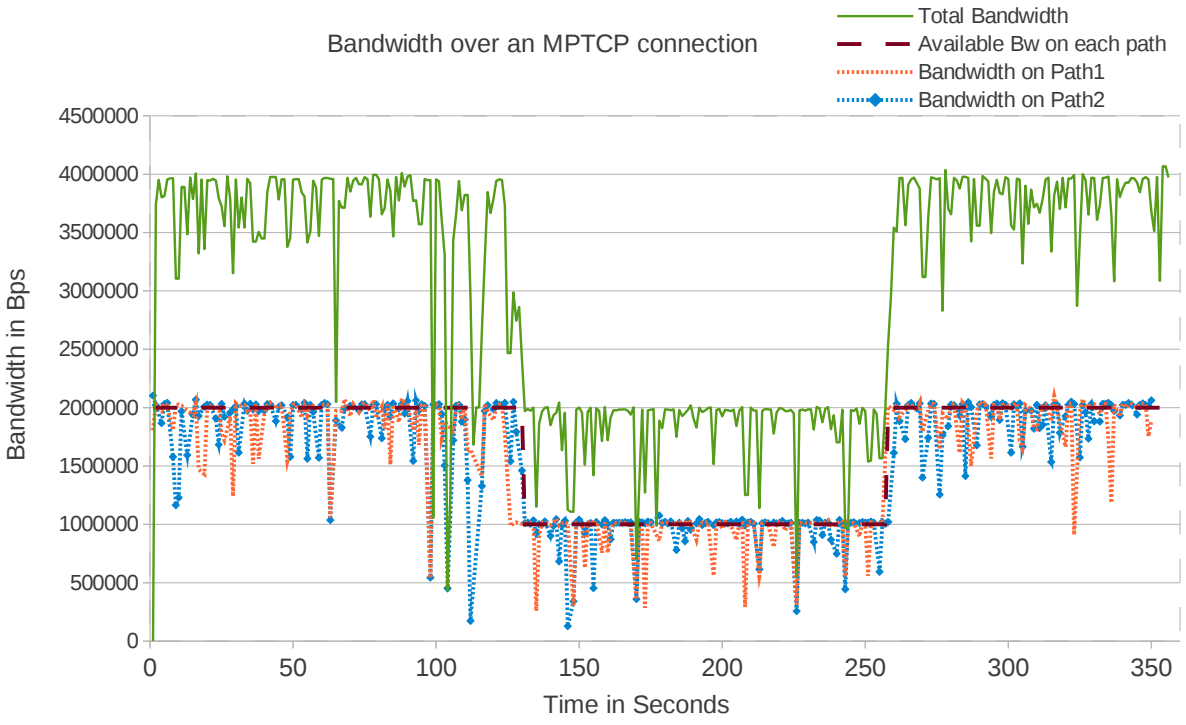


Figure 4.6: Bandwidth measurement using the modified vlc-DASH over MPTCP connection with multiple interfaces.

Takeaway

Some conclusions following from the experiment are:

- The measured samples on each path are approximately equal to the available bandwidth, thus representing the correctness of our bandwidth measurement.

- The interface-level bandwidth measurement implemented by our `bandwidth estimator` is suited for MPTCP connection and works as expected.

4.4.1.3 Harmonic Averaging over a Dynamic Window

We perform harmonic averaging over the measured samples using a dynamic window as described in Section 3.2.5. To evaluate the harmonic averaging for MPTCP streaming, we use an experimental setup similar to that in Section 4.4.1.2. During the whole session, 6s segments are used and the round trip time is set to 60ms. The total available bandwidth is set to 4000 Kbps from the beginning of the session till 80 seconds. At time t=81s the available bandwidth is reduced to 3000 Kbps for the next 60 seconds. Then, at time t=140s the available bandwidth is increased to 4000 Kbps for the next 80 seconds. After time t=220s available bandwidth is set to 3000 Kbps till the end.

Figure 4.7 shows the results. We plot three different set of data points. The measured bandwidth points ("Measured Bandwidth" curve) refer to bandwidth samples obtained using the Pcap library (evaluated in Section 4.4.1.2). The valid samples are obtained by selecting only the maximum measured samples, as described in Section 3.2.4 (illustrated as "Considered valid samples" in Fig. 4.7)). The "Harmonic Averages" curve shows the results of applying harmonic averaging to the valid samples. We see that the proposed bandwidth estimation is able to effectively smooth the variations inherent in the measured bandwidth and provide accurate bandwidth estimates. We also see that the averaging based using the dynamic window is able to closely track the available bandwidth over time. For example, at times t=80s and t=220s, when the bandwidth is decreased abruptly, we dynamically adjust the averaging window within next two samples (two segment periods). As the result, the estimator produces accurate results despite the large and abrupt bandwidth drops t=80s and t=220s. The advantage of dynamic window adjustments can be clearly seen by focusing on time instant t=141s. When the available bandwidth is increased to 4000 Kbps, we are able to filter out the sample that significantly differs from the long-term average and obtain a correct estimate.

Takeaway

Harmonic averaging allows to obtain accurate bandwidth estimates and to closely track the available bandwidth over time. This property makes it attractive for use in streaming adaptation.

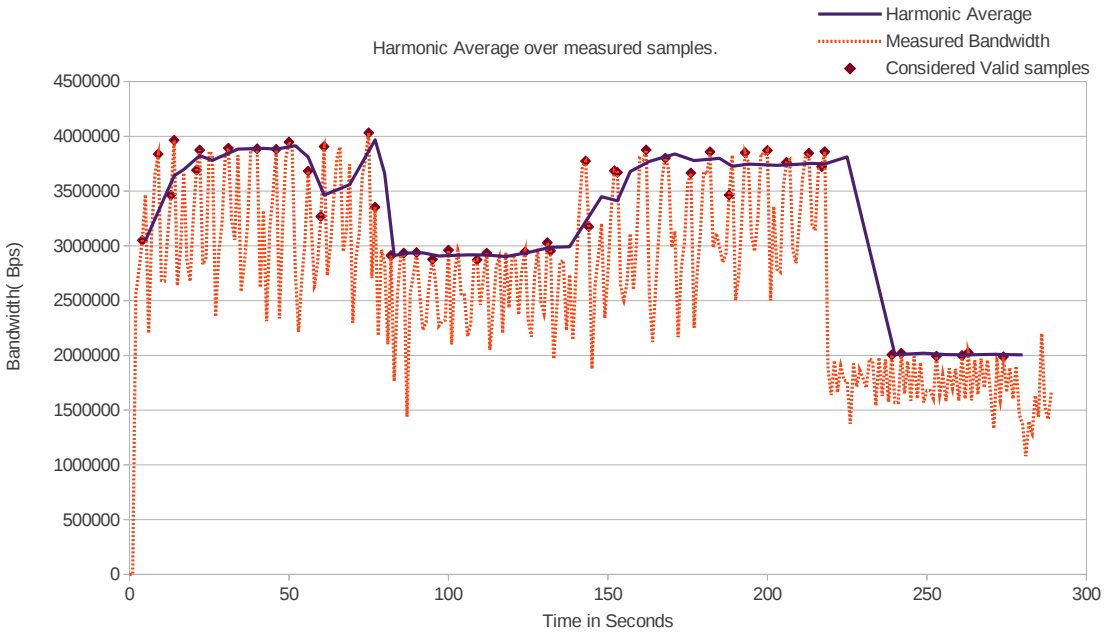


Figure 4.7: Harmonic averaging over the measured bandwidth values.

4.4.1.4 Bandwidth Measurement in Presence of Spikes

As decribed in Sections 3.2.4 and 3.2.5, our harmonic averaging has to remove outliers among the valid samples. In this section, we perform experiments to demonstrate how the outliers are removed prior to averaging. Our experimental setup is similar to that in Section 4.4.1.2.

The total available bandwidth is set to 3000 Kbps from the beginning of the session till 180 seconds. At time t=181s the available bandwidth is increased to 4000 Kbps for the next 5 seconds. Then, at time t=185s the available bandwidth is decreased to 3000 Kbps for the next 65 seconds. After time t=250s the available bandwidth is decreased to 2000 Kbps for next 5 seconds. Finally at time t=255s the available bandwidth is set to 3000 Kbps till the end of the session.

Measured bandwidth samples, valid samples and averaged values after the outliers removal are shown in Figure 4.8.

As seen from the graph, eventhough the measured samples contain both large positive and large negative spikes of the available bandwidth, they are not considered for averaging, since at least two consecutive samples larger than the threshold are required to trigger a change in the available bandwidth.

Takeaway

This shows that our harmonic averaging does not respond to the positive and negative spikes in the measured values. As such, it gives a good performance under abrupt drops

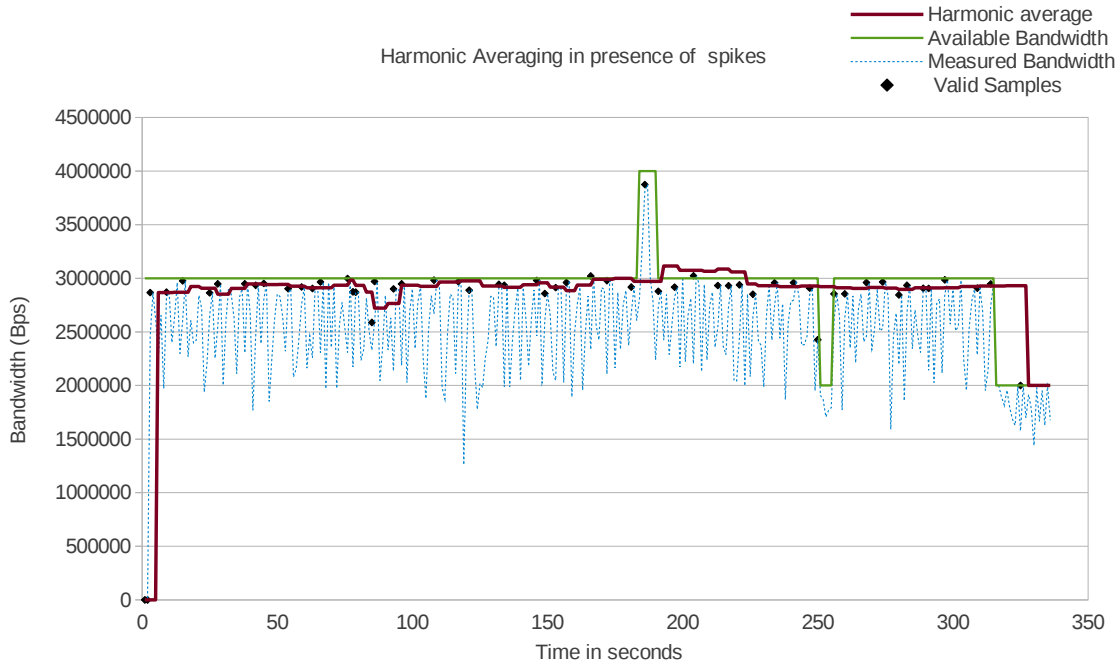


Figure 4.8: Harmonic averaging in presence of spikes.

or increases in the available bandwidth. Its use in an adaptive-streaming algorithm will be described next.

4.4.2 Validation of Adaptation Logic

In this section, we perform experiments to evaluate the `adaptation logic` component proposed in Section 3.3. As explained in Section 3.3, the `adaptation logic` operates in different functional modes, depending on the setting of the α parameter when switching to higher video levels. Our evaluations for the MPTCP and the TCP are given in Sections 4.4.2.1 and 4.4.2.3, respectively. In order to demonstrate the effectiveness of the path-stability parameter in MPTCP streaming (Section 3.4.2), we perform an experimental evaluation of MPTCP adaptation with and without path-stability in Section 4.4.2.2. The switching to lower video levels is indifferent to the choice of a functional mode, and is evaluated in Section 4.4.2.4.

4.4.2.1 TCP Adaptation Logic - Switching to Higher Video Levels

As detailed in Section 3.3.4, our adaptation logic moves to higher video level only if: (1) operating in steady phase and (2) buffer gain B_g (defined in Section 3.3.1.3) accumulated over multiple segment periods is more than the buffer-gain threshold. The buffer-grain threshold is computed dynamically, based on efficiency and stability parameters, as

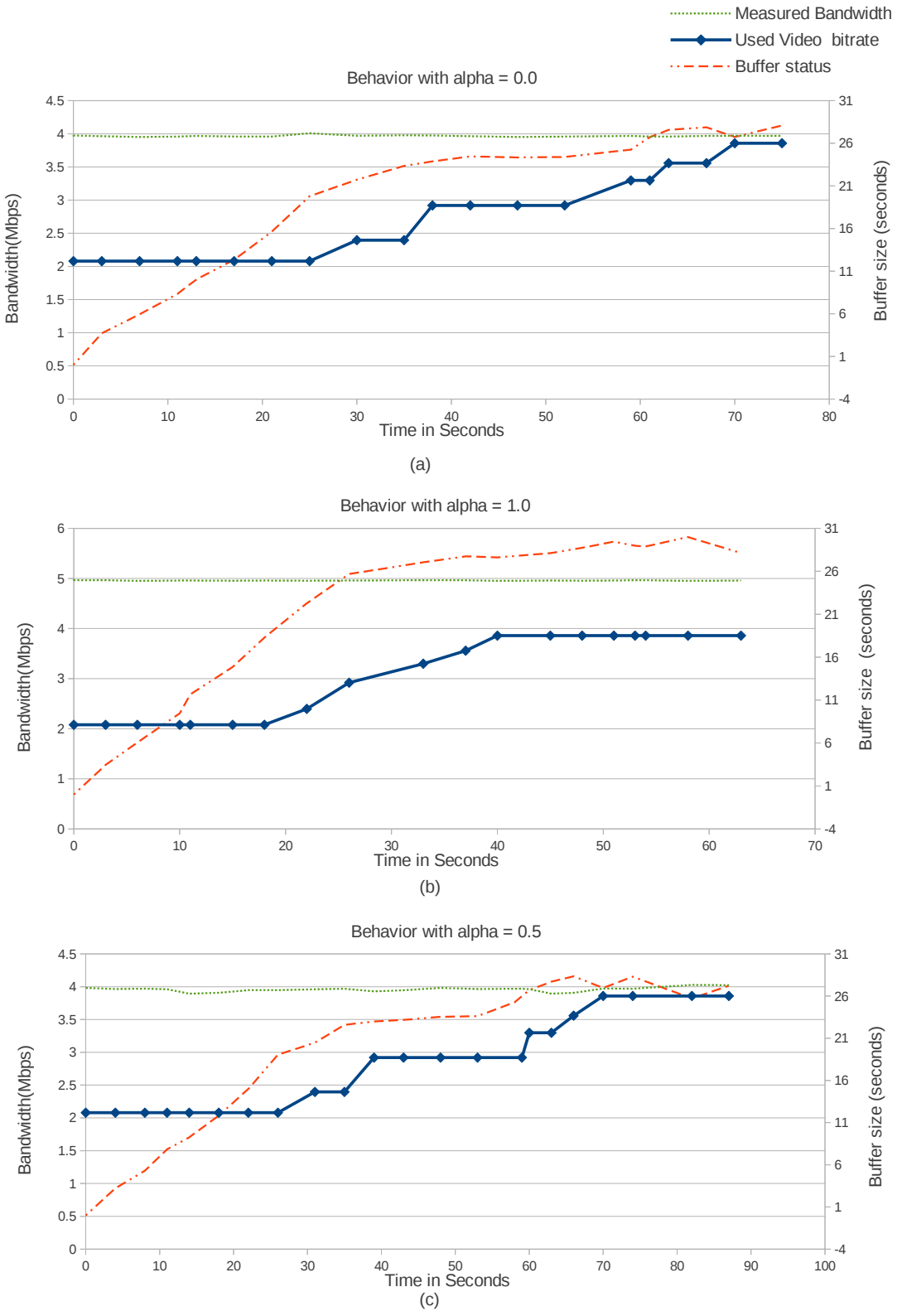


Figure 4.9: Adaptation logic for TCP under different modes.

given in Section 3.3.1.4. To validate the proposed adaptation, we perform the following experiment. Our dataset has the minimum and the maximum video bitrates of 2.07 Mbps and 3.85 Mbps respectively. We use two settings for the path capacity - 4 Mbps and 5 Mbps, depending on th functional mode. We apply these capacity settings during the entire streaming session. We consider this bandwidth over-provisioning appropriate, as the goal of this experiment is to validate the correctness of the implemented adaptation algorithm and not to evaluate its performance in time-varying bandwidth conditions. The path is configured with a round trip time of 60ms and the packet loss is set to zero for the entire session. We assume a buffer size of 30s.

Stability mode with TCP. We select the stability mode by setting the parameter α to zero and depict the behavior of our adaptation logic in Figure 4.9(a). The path capacity is 4 Mbps in this experiment. The curve with dotted markers in Figure 4.9(a) represents the bitrates of the video levels requested over time. The marker positions refer to time instants at which a segment is requested. As seen in the graph, the base video bitrate is used until t=26s. At this time instant, the buffer reached 50% of the buffer capacity and the B_g is larger than the buffer-gain threshold. Thus, our algorithm enters the steady phase at t=26 and starts to request higher video levels. According to the defined behavior in stability mode, after moving to a higher level, the adaptation logic requests the same video level for a number of subsequent segment periods. In this experiment, the algorithm requests the same video level for one subsequent segment period. As a result, the switching instants are separated by at least one segment period in Figure 4.9(a).

Efficiency mode with TCP. The behavior of the adaptation logic in efficiency mode can be observed with the parameter α set to 1, as shown in Figure 4.9(b). The path capacity is 5 Mbps in this experiment. The steady phase is reached at time instant t=19s. The adaptation enters the steady phase earlier compared to the stability mode described above, which is due to the higher path capacity setting (5 Mbps). According to the defined behavior in this mode, the adaptation logic requests higher video levels in every segment request. This is shown in Figure 4.9(b) as video levels increase from one segment period to the next, until t=40s. As a result, the adaptation exhibits aggressive behavior relative to the stability mode. Note that the bitrate growth saturates at t=40s, since we reach the highest video level available in the dataset.

Mixed mode with TCP. Figure 4.9(c) shows the behavior of the adaptation logic in mixed mode with parameter α set to 0.5. The path capacity is 4 Mbps. At time t=26s, the adaptation logic enters the steady phase. From time t=26 till t=40s, the adaptation logic is less aggressive and it waits for one segment period before moving to a higher level. In this period, the behavior of the adaptation logic is similar to its stability-mode behavior described above. At time t=40s, the buffer size remains constant with a low

buffer gain. As a result, the adaptation remains at the same video level till time t=65s. After time t=66s, the adaptation logic requests higher levels in every segment period, resulting in a behavior similar to that in efficiency mode.

Takeaway

From these results, we conclude that our adaptation logic for TCP works as expected in all three functional modes. This enables the user to set the desired adaptation behavior, based on his requirements, by setting parameter α.

4.4.2.2 MPTCP Adaptation Logic - Effectiveness of the Path-stability Parameter

As defined in Section 3.4.2, an important parameter for the MPTCP adaptation is the path-stability parameter φ_{path}. MPTCP adaptation uses the path-stability parameter to move to a higher video level only if: (1) the adaptation logic is operating in the throttling region (Section 3.4.2) and (2) the computed path-stability φ_{path} is less than the path-stability threshold (φ_{thresh}).

To evaluate the proposed MPTCP adaptation behavior with the use of the path-stability parameter, we use two paths and configure them to operate under the same network conditions. On each path, the capacity, the round trip time and the packet-loss rate are set to 1850 Kbps, 60ms and zero, respectively. The adaptation logic behavior is as shown in Figure 4.10. We focus our discussion on time instants where the behavior in the throttling region can be best observed: t=185s and t=405s. As seen in the graph, the adaptation logic first enters the throttling region at time t=110s. While in this region, the adaptation logic does not move to higher video levels, since the path-stability φ_{path} is larger than the path-stability threshold (φ_{thresh}). After time instant t=185s, less bandwidth variations are present. As a result, φ_{path} becomes smaller than the threshold. Thus, the adaptation logic moves to a higher video level and remains at this level until time t=280s. At time t=405s, the adaptation logic reaches the throttling region again. However, since much more variation is present, the adaptation logic does not move to higher levels.

We also evaluate MPTCP adaptation logic without the path-stability parameter. The results are shown in Figure 4.11. As seen in the graph, at time t=100s, the adaptation logic enters the throttling region and requests a higher video level until time t=190s. At time t=190s, the buffer size varies with varying bandwidth and hence the buffer gain also varies often. As a result, the adaptation logic changes the video bitrate between adjacent levels often. Similarly, at time t=480s, the adaptation logic enters the throttling region again. The requested video bitrate oscillates for a short period of time.

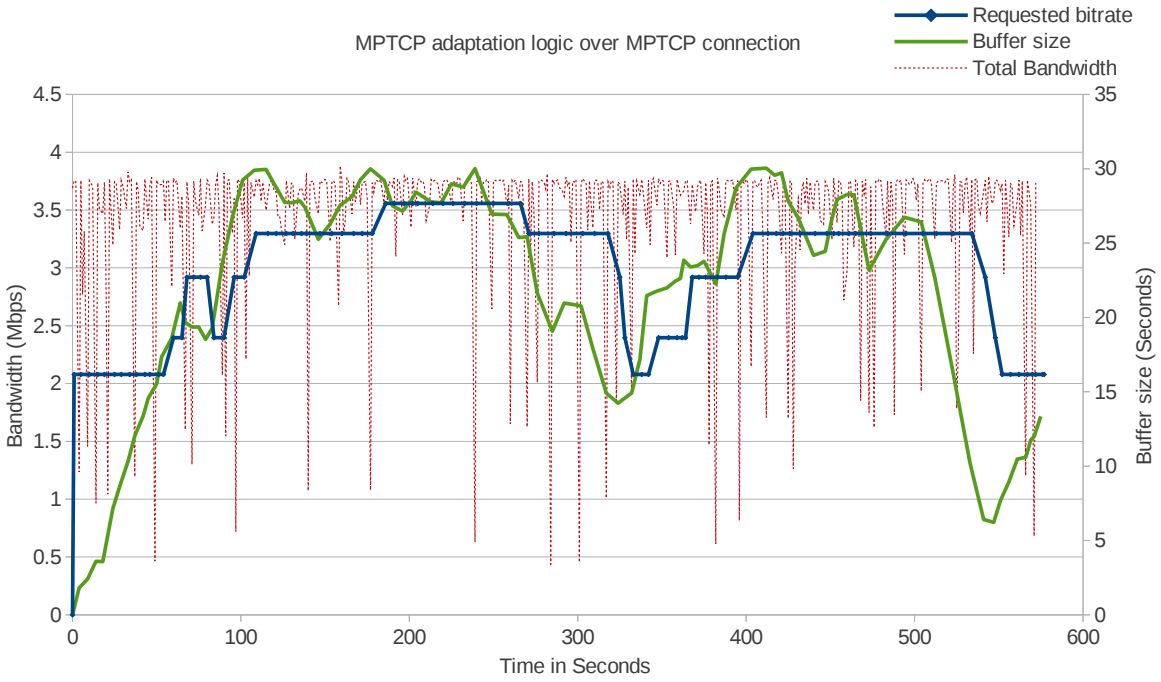


Figure 4.10: MPTCP adaptation logic with path-stability parameter φ_{path}.

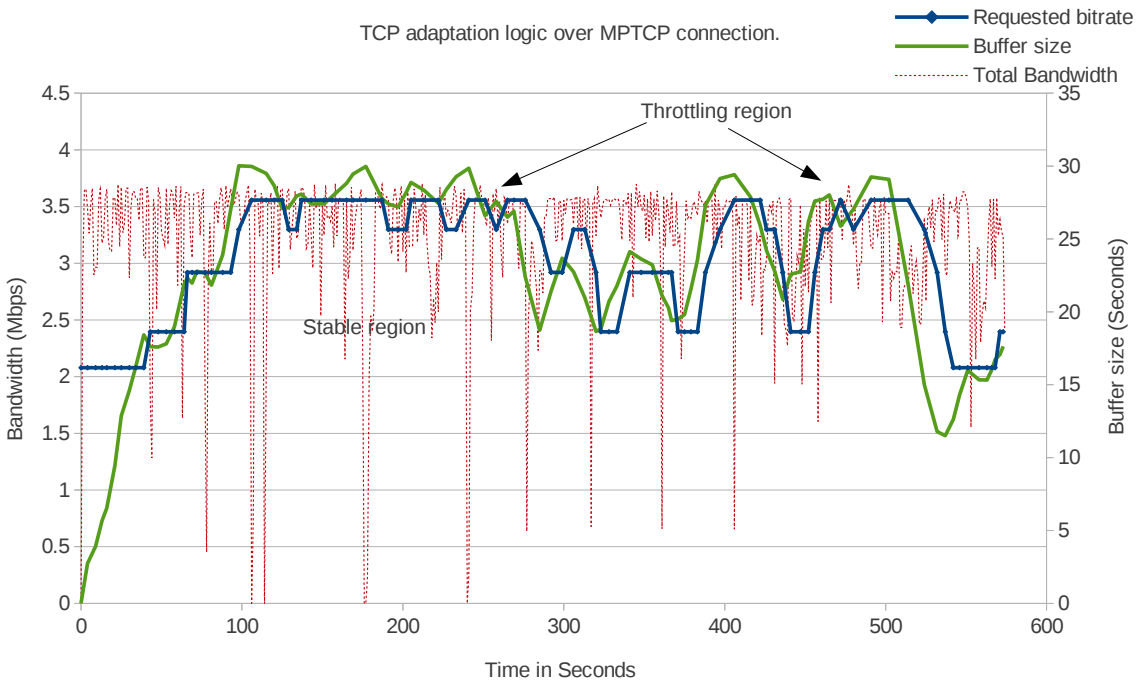


Figure 4.11: MPTCP adaptation logic without path-stability parameter φ_{path}.

Takeaway

From the above experiment, we conclude that the MPTCP adaptation logic accommodates large variations and avoids frequent switching. We achieve this by means of the path-stability parameter, which allows to minimize the number of switching events in the throttling region.

4.4.2.3 MPTCP Adaptation Logic - Switching to Higher Video Levels

Our experiment in this section is similar to that in Section 4.4.2.1. However, we found that if the path capacity is set to 4 Mbps, MPTCP adaptation logic achieves too low buffer gains. Operating in a low buffer-gain regime would make our validation difficult. To be able to properly evaluate the adaptation logic, we use higher levels of over-provisioning than in the case of TCP. We employ path capacities of 5 Mbps and 6 Mbps, depending on the functional mode. Each path is configured with a round trip time of 60ms, the packet loss is set to zero for the entire session and we assume a buffer size of 30s.

Stability mode with MPTCP. The adaptation logic is configured to operate in stability more by setting parameter α to 0. The path capacity is 5 Mbps in this experiment. The adaptation logic behavior is shown in Figure 4.12(a). The dotted markers on the used bitrate line refer to time instants at which a segment is requested. As seen in the graph, till time t=25s, the adaptation logic operates in startup phase and uses the lowest video bitrate. At time t=25s, the adaptation logic enters the steady phase and requests a higher video level. After moving to the higher level, the adaptation logic requests the same video level for one subsequent segment period. As a result, the switching instants are separated by at least one segment period in Figure 4.12(a).

Efficiency mode with MPTCP. The behavior of the adaptation logic in efficiency mode with $\alpha = 1$ is as shown in Figure 4.12(b). The path capacity is 5 Mbps in this experiment. At time t=16s, the adaptation logic moves to steady phase and requests a higher video level. The adaptation logic continues to move to higher levels in subsequent segment requests. At time t=37s, the adaptation logic enters the throttling region and moves to the next higher level after one segment request. Compared to adaptation logic for TCP, the MPTCP adaptation logic is less aggressive in this case, due to our defined throttling-region behavior.

Mixed mode with MPTCP. Figure 4.12(c) shows the behavior of the adaptation logic in mixed mode with $\alpha = 0.5$. The path capacity is 5 Mbps in this experiment. At time t=18s adaptation logic enters the steady phase and moves to a higher video level. From time t=18s till t=26s, the adaptation logic is more aggressive and moves to higher levels with each segment request. In this period, the behavior of the adaptation logic

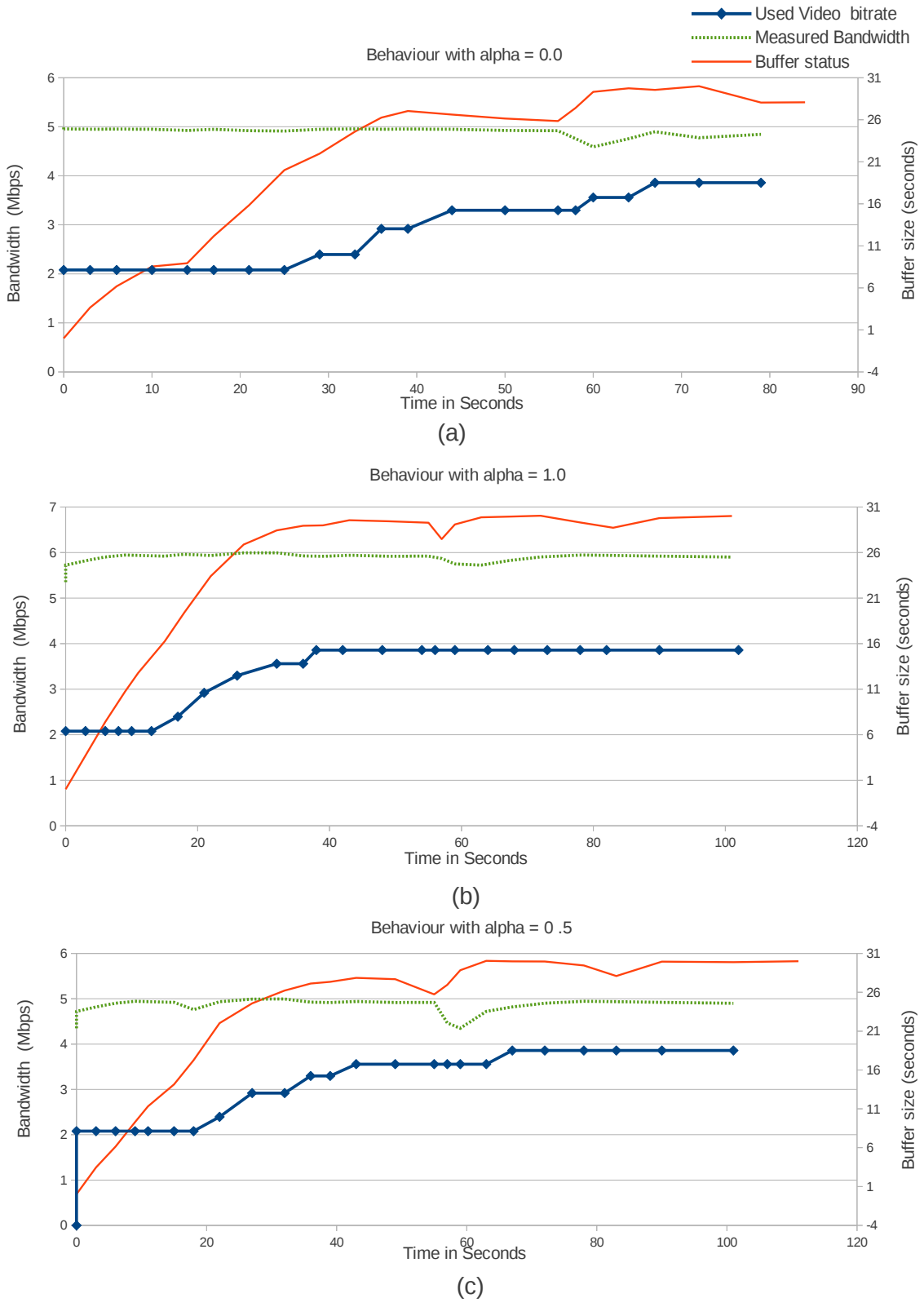


Figure 4.12: MPTCP adaptation logic under different modes.

is similar to its efficiency-mode behavior described above. From time t=26 till t=40s, the adaptation logic is less aggressive and it waits for one segment period before moving to a higher level. In this period, the behavior of the adaptation logic is similar to its stability-mode behavior. At time t=43s, the adaptation logic enters the throttling region and requests the same video level. At time t=66s, the next higher video level is requested, since the bandwidth variations are reduced (as tracked by the path-stability parameter).

Takeaway

From the above experiment, it is evident that the MPTCP adaptation logic works as expected in all three functional modes and optimizes the corresponding metrics. In addition, the throttling-region behavior and the path-stability parameter are correctly accounted for in the algorithm. To better understand the inability of MPTCP streaming to achieve the same buffer gains as TCP streaming, we perform additional experiments in Section 4.5.1. As per the stated design goals of MPTCP, its throughput performance should not be inferior to TCP's. It is therefore important to explain this unexpected behavior, as we will do in the later section.

4.4.2.4 Adaptation Logic - Switching to Lower Video Levels

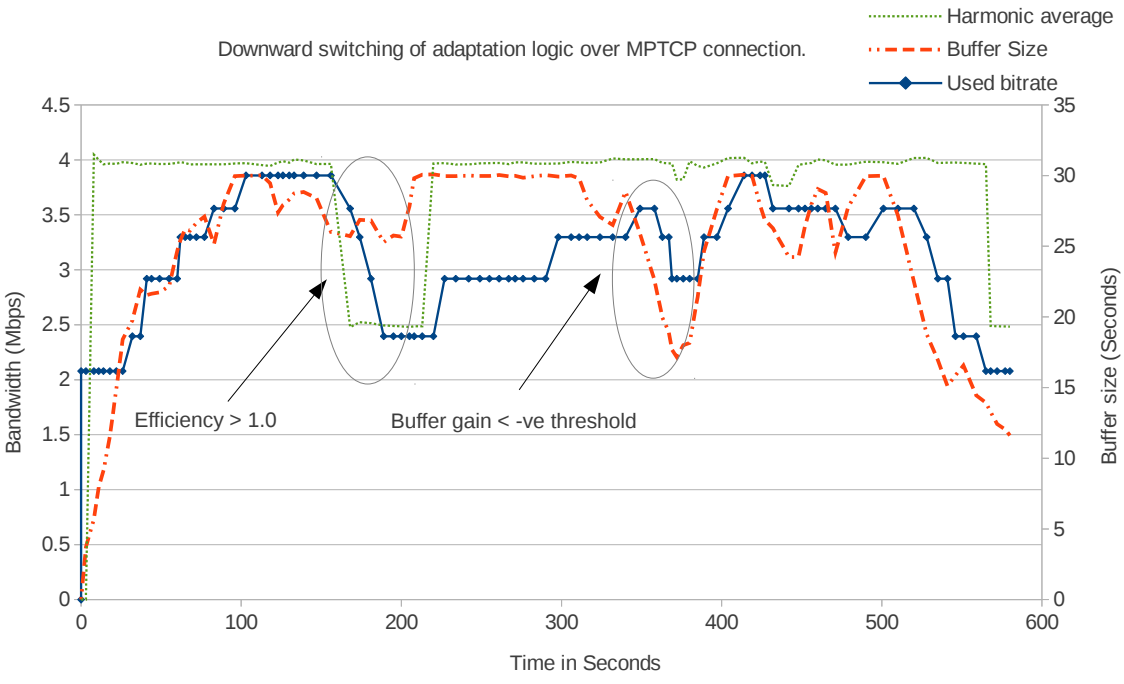


Figure 4.13: Adaptation logic downward switching behavior.

Previous experiments validated our adaptation logic when moving to higher video levels. However, during a streaming session, the bandwidth or the buffer size may get reduced.

Under these conditions, the adaptation logic has to move to lower video levels to compensate for the deteriorated network conditions. In contrast to the previously evaluated behavior when requesting higher video levels, our adaptation logic when switching to lower levels does not differentiate between the functional modes (Section 3.3.4).

Our experimental setup is as follows. For the entire session, the delay on each path is set to 60ms and the packet loss rate is zero. From the beginning till time t=150s, the total available bandwidth is set to 4 Mbps. At time t=150s, the bandwidth is reduced to 2.5 Mbps. After time t=210s, the bandwidth is reset to 4 Mbps till the end of the session. The average bandwidth, the buffer size and the used video bitrate are as shown in Figure 4.13.

We validate two aspects of the level switching: (1) efficiency is greater than one and (2) buffer gain is smaller than the negative threshold B_δ. As seen in the graph, at time t=150s when the bandwidth is reduced, the used video bitrate is larger than the available bandwidth. As a result, the efficiency becomes greater than 1. The efficiency value greater than 1 represents an over-utilization of the network, so the adaptation logic moves to lower video levels till the used video rate becomes less than the average available bandwidth (computed as a harmonic average). The adaptation logic switches to adjacent lower video levels immediately and does not wait between switching events. At time t=380s, the buffer size reduces even though there is enough bandwidth. As a result, the buffer-gain becomes smaller than the negative threshold and the adaptation logic moves to lower levels.

Takeaway
Adaptation logic moves to lower video levels under deteriorated network conditions (drop in the available bandwidth) in order to avoid playout interruptions. Unlike the behavior when switching to higher levels, we switch to lower levels immediately, without considering the recent history of switching events.

4.4.2.5 MPTCP Adaptation Logic - Overall Performance

Summarizing, the performance of our adaptation depends on appropriately configuring the required parameters. We illustrate this with the following experiment. For the whole session, the round trip time on each path is set to 150ms and the packet loss rate to zero. From the start of the session till time t=100s, the available bandwidth on each path is set to 2000 Kbps for MPTCP and 4000 Kbps for the single-path with TCP. At time t=100s, the available bandwidth is abruptly decreased to 1500 Kbps for MPTCP and 3000 Kbps for TCP. At t=160s, the available bandwidth is changed to 1750 Kbps for MPTCP and 3500 Kbps on TCP. At t=220s, available bandwidth is again decreased to

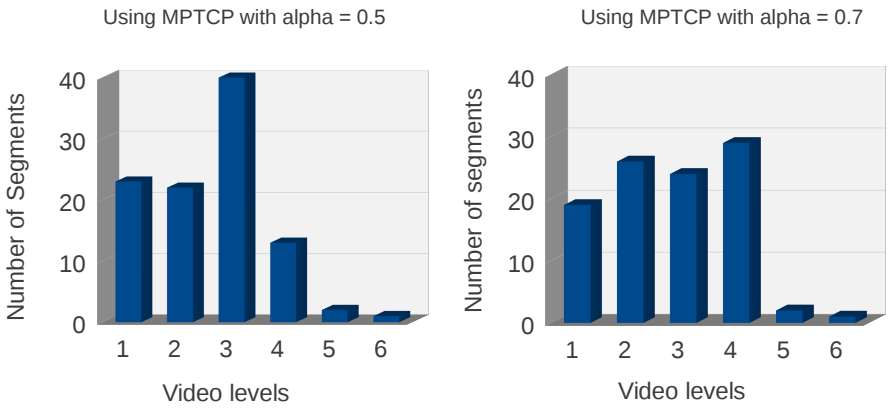


Figure 4.14: Number of segments requested at different levels.

1250 Kbps on MPTCP and 2500 Kbps on TCP. The same pattern of bandwidth changes is instantiated till the end of the session.

In our case, the streaming adaptation can be tuned with two parameters: (1) lowering the buffer-gain threshold (B_{gt}) to move to higher levels under low buffer gains and (2) running the adaptation under different modes by setting parameter α. We present experiments under different values of α and the buffer-gain threshold. The results of the MPTCP streaming with (1) α=0.5, B_{gt} and (2) α=0.7, $B_{gt} = B_{gt}/2$ are shown in Figure 4.14. A bar chart with the number of segments requested at each level gives the clear indication of the performance differences. Even though the performance is improved by using higher video levels, the number of switching events also increases from 25 to 29.

4.5 Comparison of Adaptive MPTCP and TCP Streaming

In Section 4.3.1, we demonstrated the MPTCP's ability to pool the capacities of multiple available paths. More specifically, with the use of multiple paths, the total capacity of a MPTCP connection is equal to the sum of capacities of the used paths. This means that the performance of a streaming application implemented over MPTCP will be superior to the same application's performance over TCP, since TCP lacks support for multi-path transport and is unable to pool path's capacities. However, this provides little insight into the performance difference when using the two protocols for video streaming in scenarios where multiple paths have shared bottlenecks, or on lossy paths where the quality of a path changes significantly over time. For example, the coupled congestion control in MPTCP is designed such that in shared bottleneck scenarios the MPTCP connection on aggregate remains fair to competing TCP flows [6]. It is not clear if the

MPTCP streaming would outperform TCP streaming in this case. The performance comparison of MPTCP and TCP streaming in such scenarios is the focus of this section.

When comparing the performance of a streaming application over MPTCP and TCP, we have to take into account the differences in terms of protocol support for multi-path streaming. Importantly, the question is raised as to how to ensure that the protocols are compared under *equivalent* conditions. To this end, we propose the following experimental methodology.

- *Equivalent conditions.* Our experiments are conducted under the "equivalent" network conditions. We define the "equivalent" network conditions as: (1) the total aggregate available bandwidth of the multiple paths with MPTCP is equal to that of TCP's single path, (2) the round trip times (delays) of each path are equal to that of TCP path and (3) the packet loss rate on each path is equal to that of TCP. For comparison purposes, we define the term "equivalent performance". It is defined in terms of the number of segments requested at each level. A streaming system is said to have an "equivalent performance" to another system when it uses approximately the same number of segments at each video level.
- *Parameter configuration* of the streaming algorithms. Our streaming application is implemented with the previously proposed bandwidth estimation and adaptation algorithms. During each experiment, the parameters of the adaptation algorithms are set to same values for both TCP and MPTCP streaming.
- *Scenarios.* We present comparisons of the MPTCP and TCP streaming under a range of conditions. These include high network delays, high packet losses and streaming using non-persistent connections.

4.5.1 MPTCP Streaming Performance to Fill Client Buffer

In general, MPTCP should perform similarly to TCP under equivalent conditions. However, our experimental results in Section 4.4.2.3 show that under conditions of low delay and no packet loss, the MPTCP streaming requires more bandwidth compared to TCP streaming to achieve the "equivalent" performance. To better understand this, we evaluate the buffer growth rate during MPTCP and TCP streaming. As a part of this evaluation, we also show the variation of congestion windows during streaming for both protocols. Our experimental setup is as follows. Both paths are configured with a round trip time of 60ms and the packet loss is set to zero for the entire session. The path capacity of a single path is set to 4 Mbps. For the case of MPTCP streaming, the two paths have a capacity of 2 Mbps each.

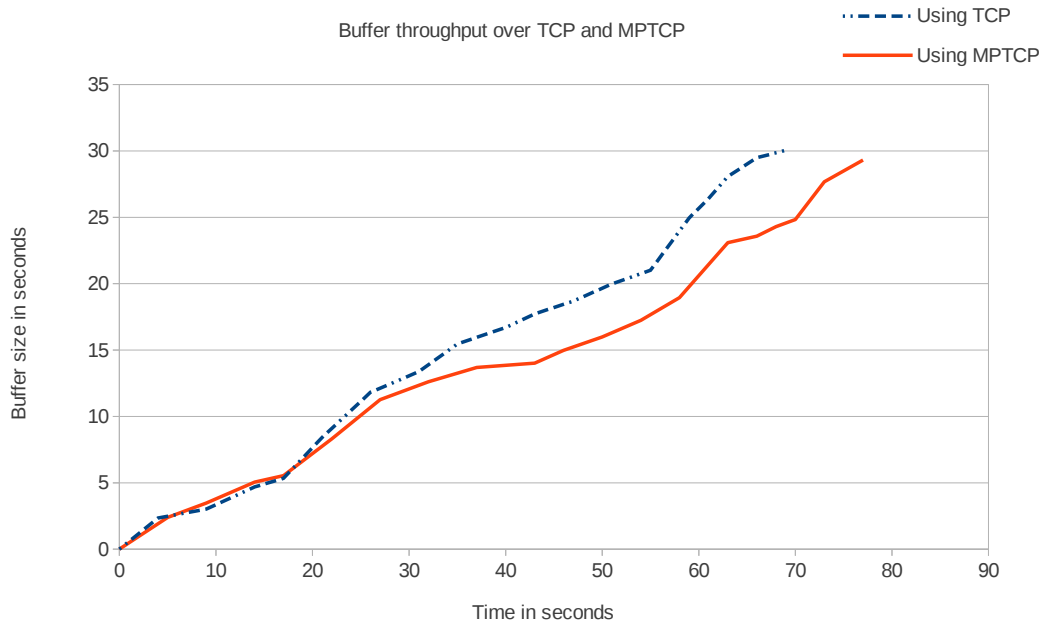


Figure 4.15: Buffer growth rate using TCP and MPTCP.

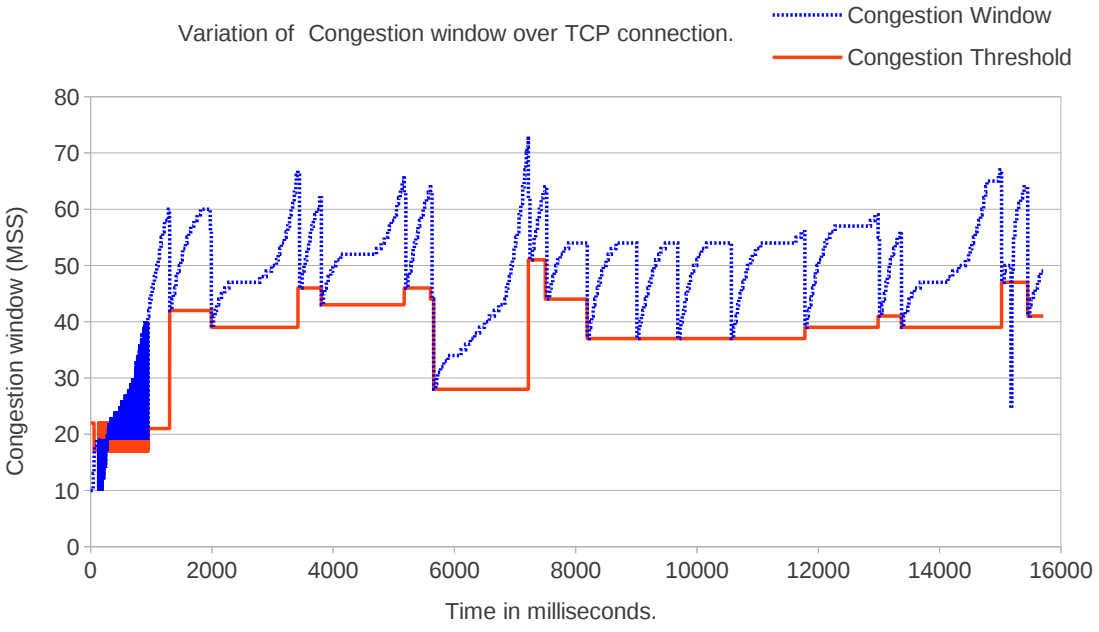


Figure 4.16: Variation of congestion window over TCP.

We measure the time from the beginning of the streaming session till the time the buffer reaches its full capacity. The results are as shown in the Figure 4.15. As seen from the graph, till time t=20s, the buffer size increases at the same rate for both the TCP and MPTCP. After time t=20s, the buffer-increase-rate in TCP is higher compared to MPTCP and this behavior continues till the buffer reaches its full capacity. As a result, TCP streaming takes less time (68s) to reach the full buffer capacity compared to MPTCP (78s).

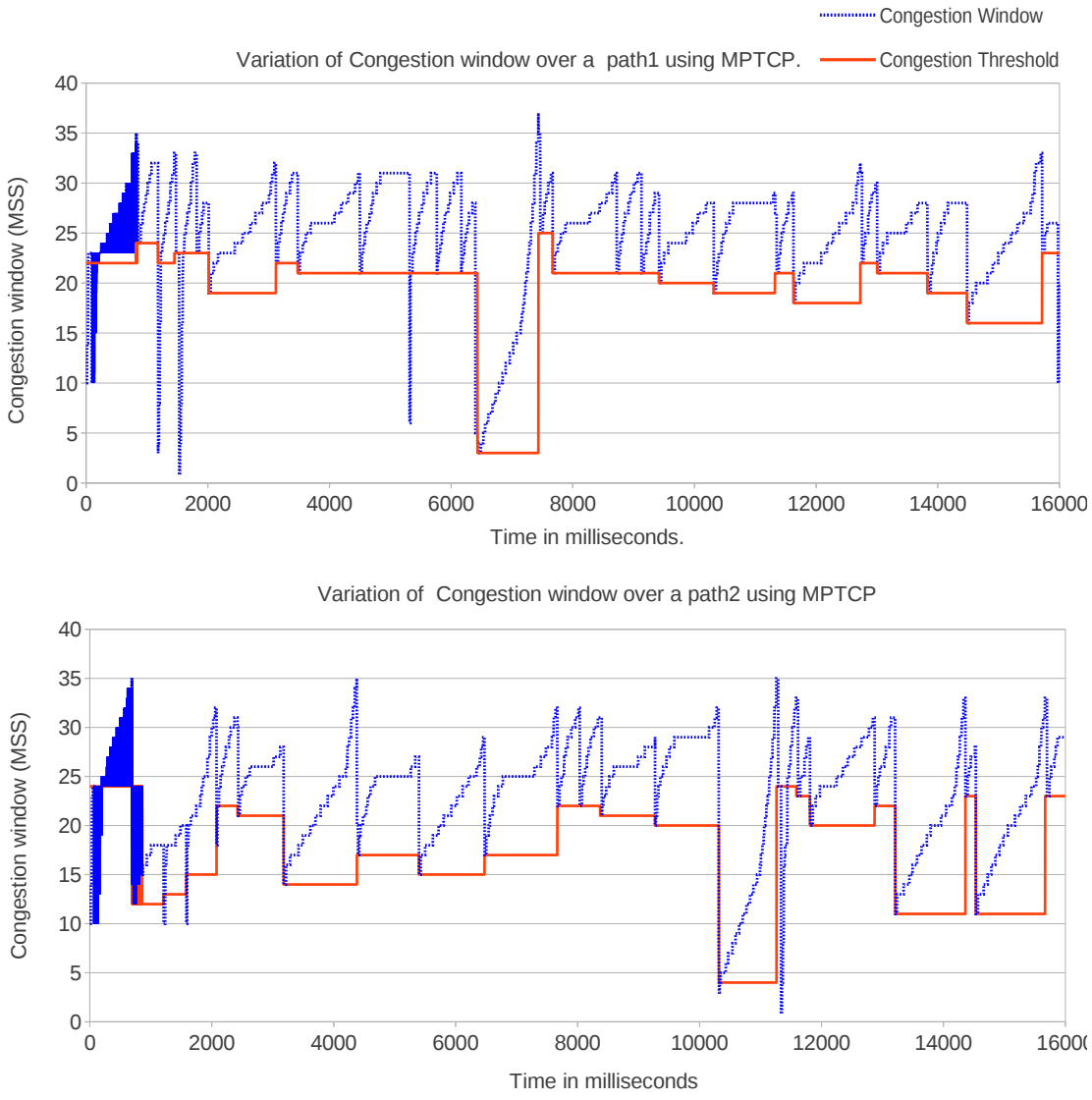


Figure 4.17: Variation of congestion windows (two paths) over MPTCP.

To understand this behavior, we trace the congestion window growth in TCP and MPTCP for the entire session. Figures 4.16 and 4.17 show the variation of congestion windows and the congestion window thresholds (ssthresh) using TCP and MPTCP, respectively. As seen in the graph, the TCP congestion window experiences fewer multiplicative decrease events compared to individual subflows of MPTCP. As a result, the average value of the

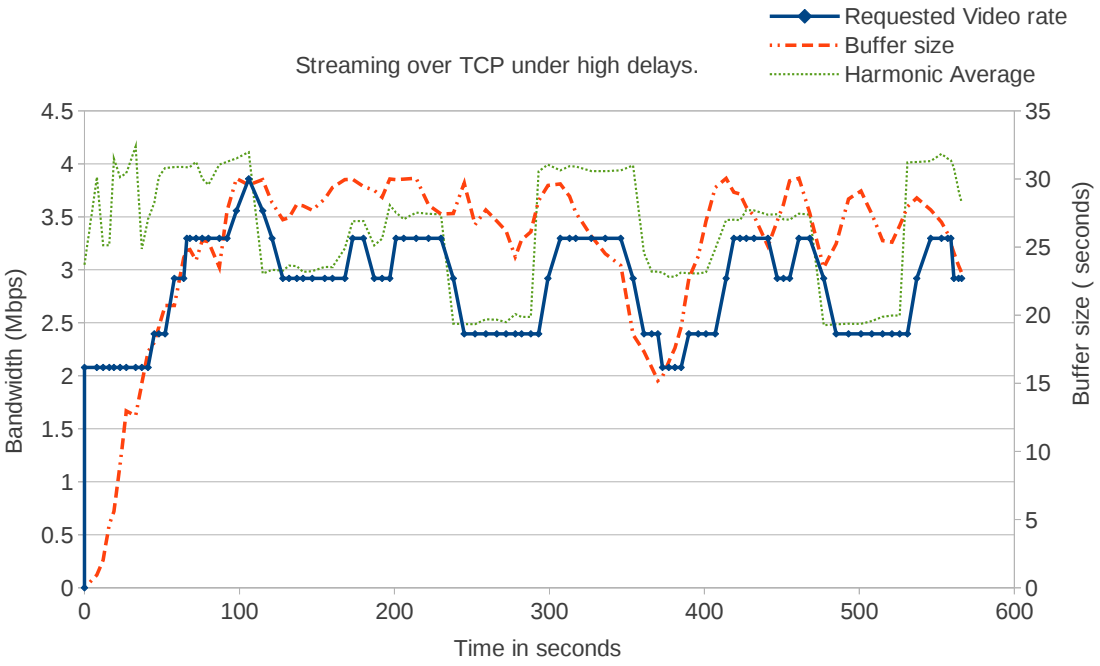


Figure 4.18: Streaming over TCP under high delay of RTT=150ms.

congestion window over the entire session is higher in TCP compared to MPTCP. We also see that the congestion window's multiplicative events become periodic over a TCP connection. But in case of MPTCP, the individual congestion windows on each path experience more frequent variations. We explain the reason for this behavior as follows. The available bandwidth on each path is smaller than that of an equivalent single path (half of the total bandwidth). As a result, the maximum value of the congestion window is also smaller. Eventhough the MPTCP congestion window grows at the same rate as that of TCP, it reaches th maximum value (limited by the path capacity) in less time and experiences multiplicative decrease events earlier compared to TCP. We believe that the frequent variations of the individual congestion windows in MPTCP subflows cause the increase of the time required to fill the buffer.

4.5.2 MPTCP vs. TCP under High Delays

In Section 3.4.1.1, we have found that under high delay conditions (more than 100ms of RTT), the total bandwidth achieved over an MPTCP reduces. As a result, low buffer gains are achieved. In this section, we evaluate the performance of the MPTCP and the TCP streaming under the "equivalent" high delay conditions.

Our experimental setup is as follows. For the whole session, the round trip time on each path is set to 150ms and the packet loss rate to zero. The adaptation logic is configured to operate in mixed mode by setting the parameter α to 0.5. From the start of the

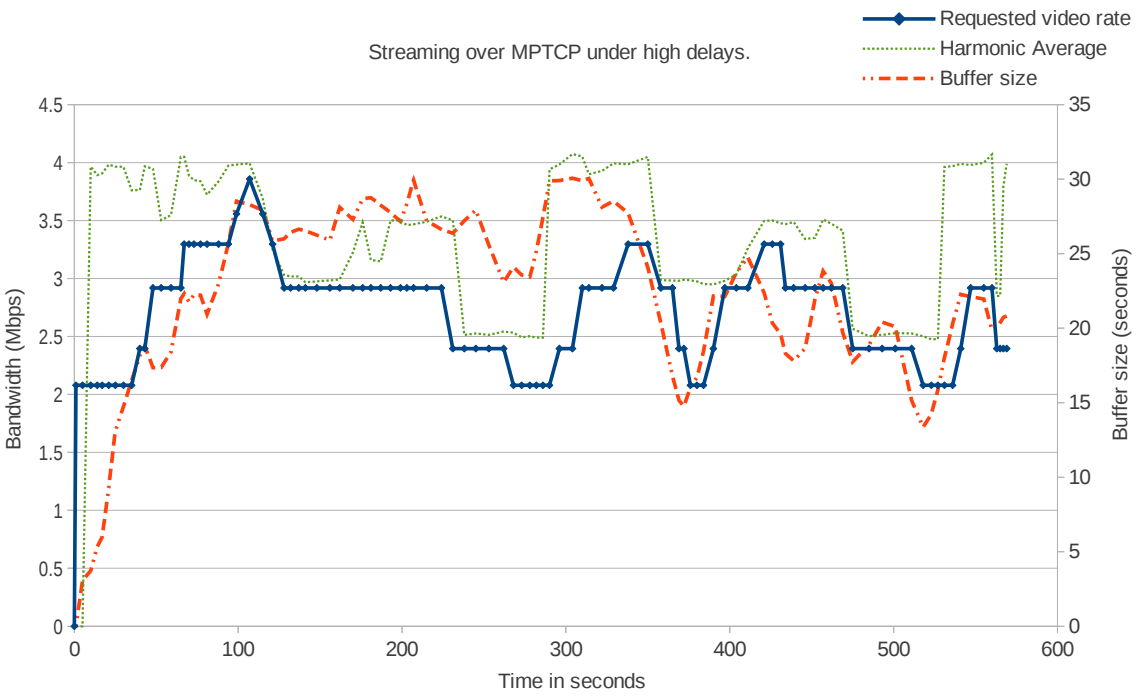


Figure 4.19: Streaming over MPTCP under high delays on both paths (RTT=150ms).

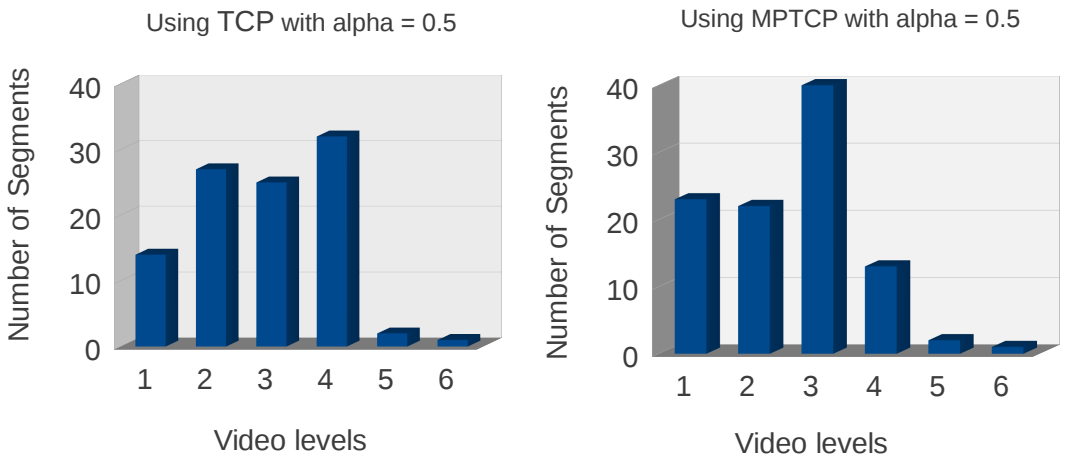


Figure 4.20: Number of segments requested at different levels in TCP and MPTCP under High delays.

session till time t=100s, the available bandwidth on each path is set to 2000 Kbps for MPTCP and 4000 Kbps for the single-path with TCP. At time t=100s, the available bandwidth is abruptly decreased to 1500 Kbps for MPTCP and 3000 Kbps for TCP. At t=160s, the available bandwidth is changed to 1750 Kbps for MPTCP and 3500 Kbps on TCP. At t=220s, available bandwidth is again decreased to 1250 Kbps on MPTCP and 2500 Kbps on TCP. The same pattern of bandwidth changes is instantiated till the end of the session.

Results are as shown in Figures 4.18 and 4.19. As seen in the graph, the buffer size remains at higher levels during TCP streaming compared to MPTCP streaming. As a result, the TCP streaming achieves higher buffer-gains and gives better performance by requesting higher video bitrates compared to MPTCP streaming.

In order to understand this performance difference, we perform an analysis of the congestion window behavior similar to that in Section 4.5.1 (not shown here). We found that, the individual congestion windows of the MPTCP connection experiences more number of decrease events and grows slowly due to high RTT's compared to that of the TCP congestion window. Also, the MPTCP connection experiences large decrease in the congestion window values compared to that of TCP.

Takeaway
From this experiment, we conclude that under the same conditions, TCP performs better compared to MPTCP by operating at the higher video levels.

4.5.3 MPTCP vs. TCP under High Packet Losses

A TCP connection over a high packet-loss path experiences high bandwidth variations with abrupt changes, mainly due to TCP congestion control. The performance of adaptive streaming under these conditions demonstrates the suitability of the protocol for streaming. In this section, we evaluate the performance of the MPTCP and TCP streaming under the "equivalent" packet loss conditions on all paths. Our experimental setup is similar to that in Section 4.5.2. The packet loss on all the paths is set to 0.005 and round trip times to 60ms.

The results of the MPTCP and the TCP streaming are as shown in Figures 4.21 and 4.22 , respectively. As shown in the graph, high packet losses in MPTCP streaming have less impact on the achieved throughput compared to TCP. The TCP streaming achieves a low throughput (65% of the total capacity) over the entire session, with larger variations. With these low achieved throughputs, TCP streaming uses only the lowest two video levels over the entire session as shown in Figure 4.22. The MPTCP streaming performs better compared to TCP streaming since the achieved throughput is larger and more stable. As a result, the MPTCP uses higher video levels. A bar chart with the number of segments requested under TCP and MPTCP streaming is as shown in Figure 4.23.

In order to understand this performance difference, we perform an analysis of the congestion window behavior similar to that in Section 4.5.1 (not shown here). We have concluded that:

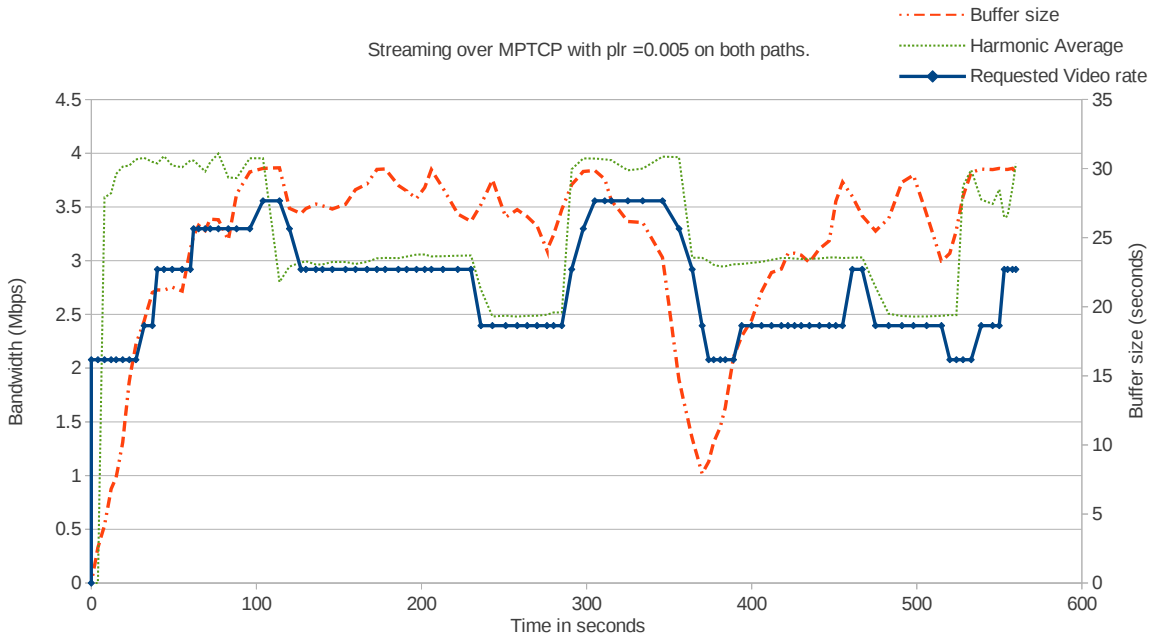


Figure 4.21: Streaming over MPTCP under packet loss conditions on both paths.

1. With MPTCP, the bandwidth on each path is lower than on an "equivalent" single path (half of that of TCP). As a result, the highest value of the congestion window is smaller. Thus, the amount by which the congestion window size decreases in each multiplicative decrease event is smaller than in case of TCP. As a result, the average congestion window over the MPTCP session is larger than that of TCP.

2. The packet loss events on each path are not correlated and hence the congestion window behaviors are not correlated. For this reason, MPTCP achieves a higher average throughput.

Takeaway

From the experiment we can conclude that MPTCP streaming performance is better compared to TCP streaming under high packet loss conditions.

4.5.4 MPTCP vs. TCP under Persistent and Non-Persistent Connections

Most of today's state-of-the-art streaming solutions use non-persistent connections. However, at the time of building our testbed, the vlc-DASH plugin only supported persistent connections. For this reason, all our experiments so far involve the use of persistent connections. In the meantime, vlc-DASH has obtained support for non-persistent connections. When using non-persistent connections, it is important to note that: (1) some latency is always required for connection setup (2) for a fresh connection, the congestion window starts from low values, which may result in a decrease of overall

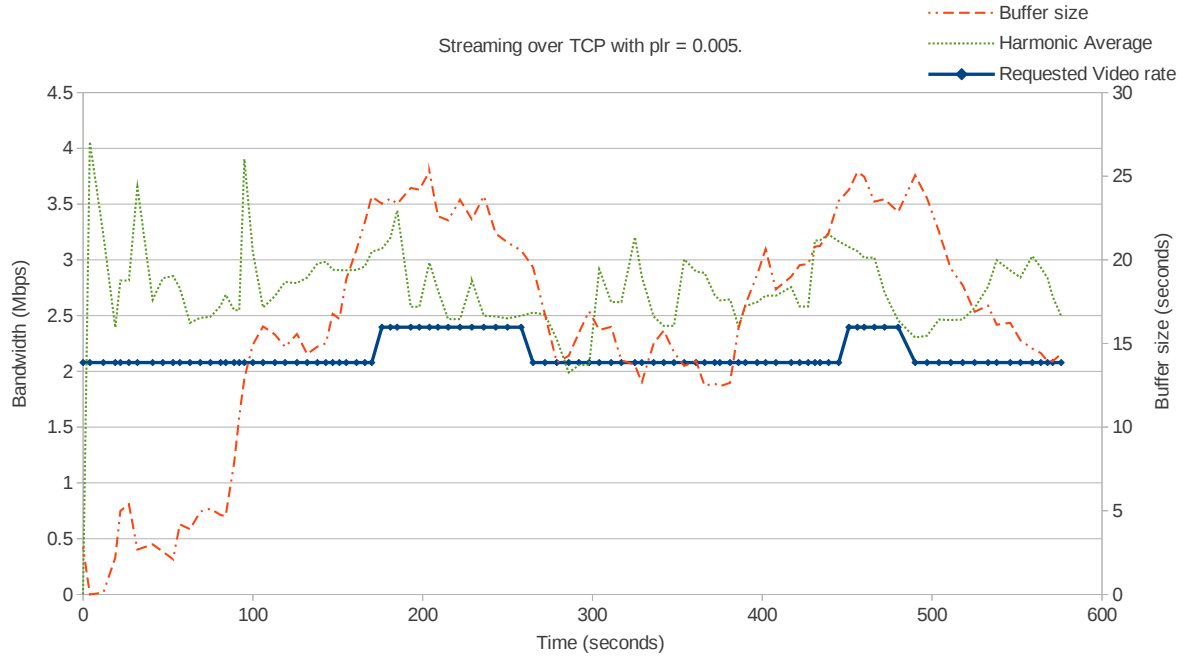


Figure 4.22: Streaming over TCP under packet loss conditions.

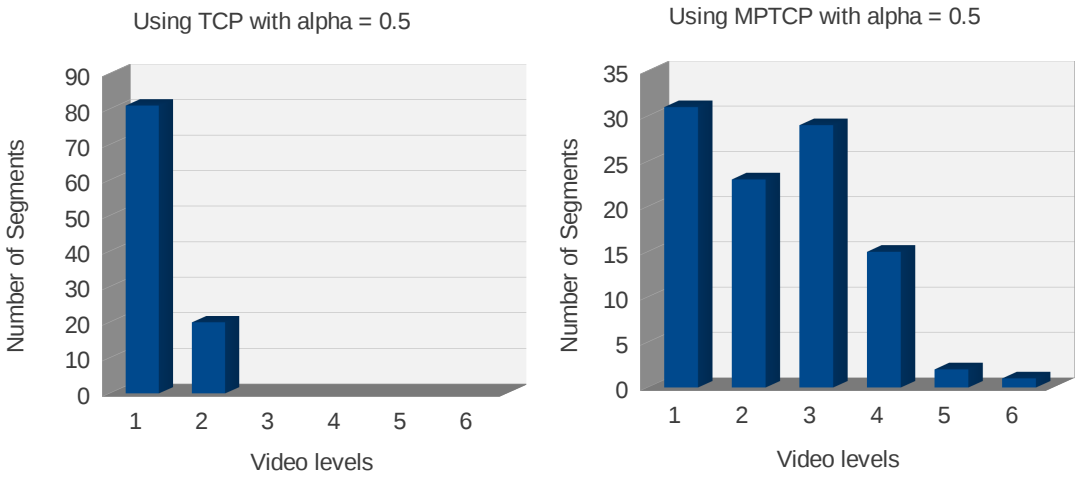


Figure 4.23: Number of segments requested at different levels in TCP and MPTCP.

throughput compared to using an already open ('warm') connection. The performance of such connections is particularly important for MPTCP streaming, since more latency is required when a connection needs to be setup over all the available paths, as discussed in Section 2.3.2.2. In this section, we evaluate the performance of MPTCP using non-persistent connection mode compare it to TCP streaming.

Our experiment setup is similar to that in Section 4.5.2. Each path is configured with 60ms of round trip time and packet loss of zero. The performance results of the TCP streaming in non-persistent connection mode, MPTCP streaming in persistent and non-persistent connection modes are shown in Figures 4.25, 4.26 and 4.27, respectively.

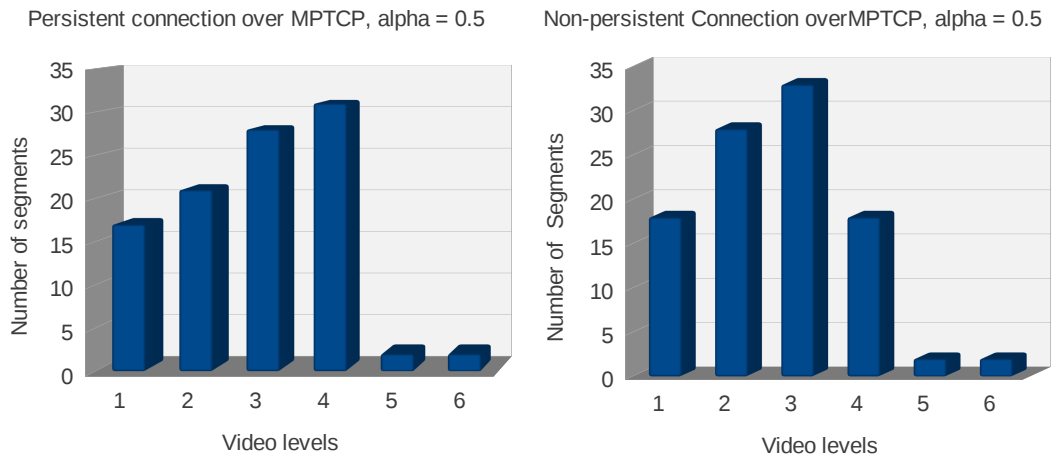


Figure 4.24: Number of segments requested at different levels in MPTCP using persistent and non-persistent connections.

First, we see that the MPTCP streaming performs under performs compared to that of its persistent connection mode (Figure 4.24). Second, we see that the MPTCP streaming performs better compared to TCP. A bar chart with the number of segments requested at each video level is shown in Figure 4.28. As seen, TCP never uses the highest video bitrates and MPTCP uses the higher video bitrates for at least two segment periods. The highest video bitrates used by the TCP streaming and the MPTCP streaming are 3.29 Mbps and 3.85 Mbps, respectively. The number of switching events are also reduced for MPTCP(22) compared to TCP(23).

In order to understand this difference, we have examined the congestion window behavior, similar as in previous sections. For every new TCP connection, the congestion window starts with the value of 10 MSS. For MPTCP, a new connection creates two subflows with congestion windows starting with values of 10 MSS (total effective congestion window=20 MSS). As a result, at the beginning of the each connection, more data transfer happens over MPTCP streaming compared to TCP streaming.

Takeaway

From the above experiment, we conclude that MPTCP streaming performs better compared to TCP streaming in the non-persistent mode since it achieves a higher throughput in the beginning of the connection. The use of non-persistent connections has little impact on MPTCP streaming performance, while TCP streaming under-performs compared to its persistent-connection mode behavior.

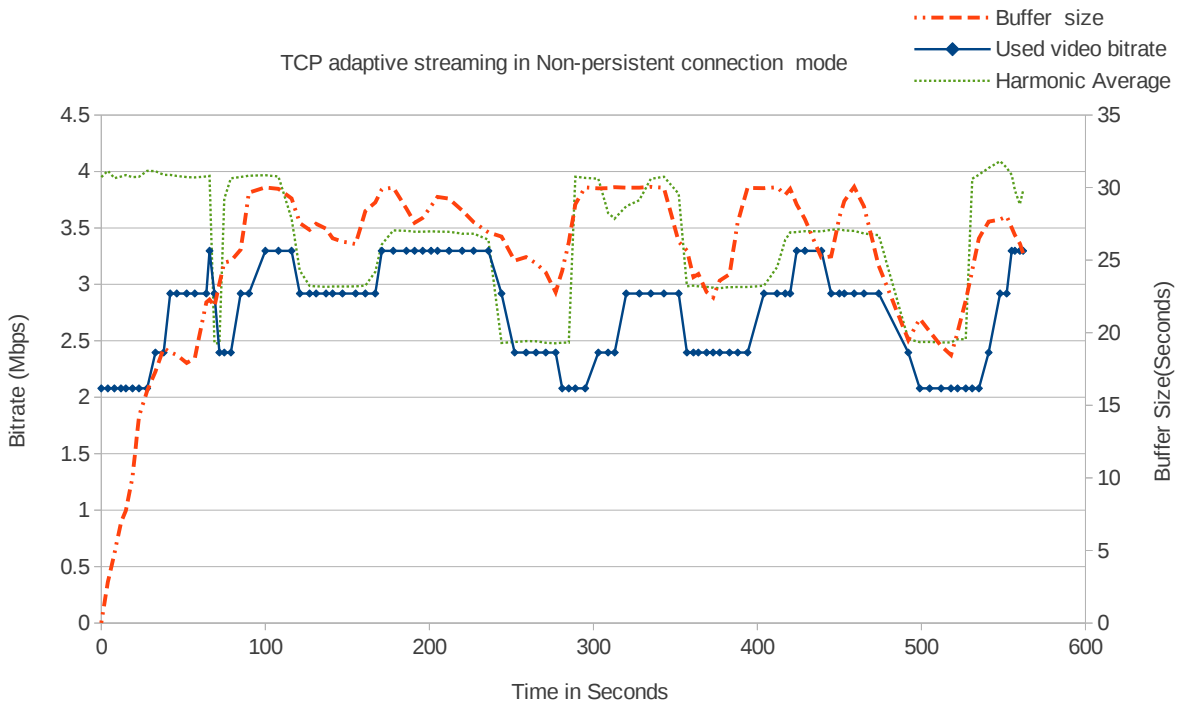


Figure 4.25: Streaming over TCP under non-persistent connections.

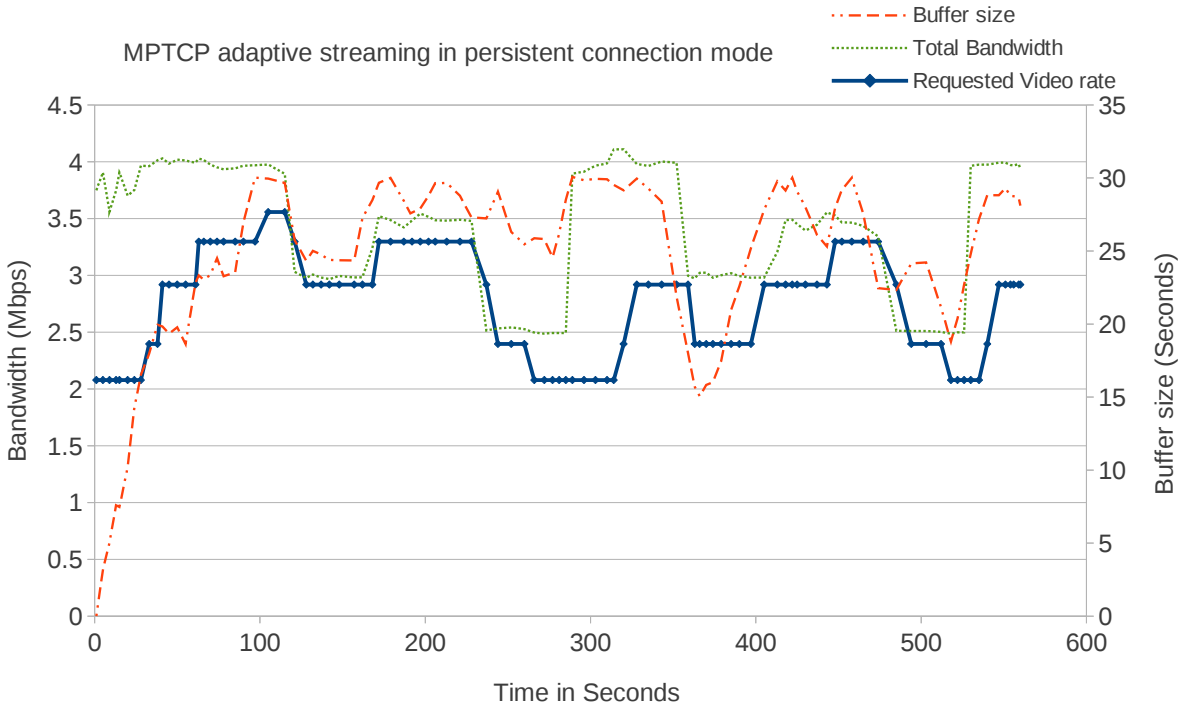


Figure 4.26: Streaming over MPTCP under persistent connections.

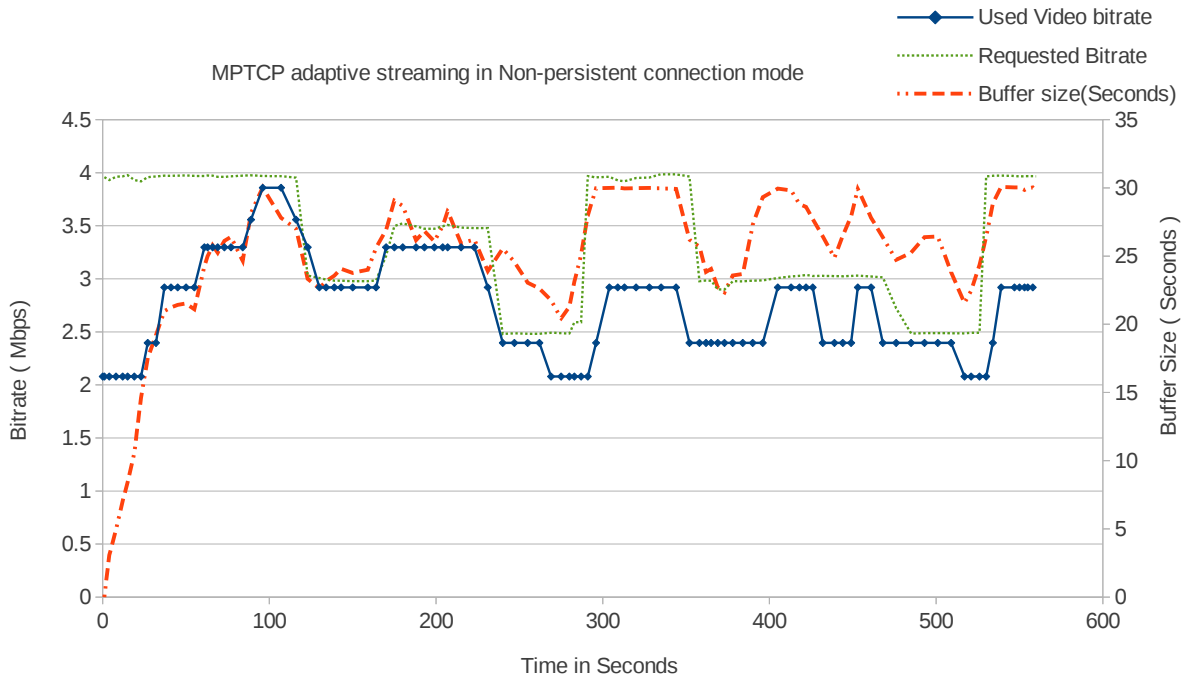


Figure 4.27: Streaming over MPTCP under non-persistent connections.

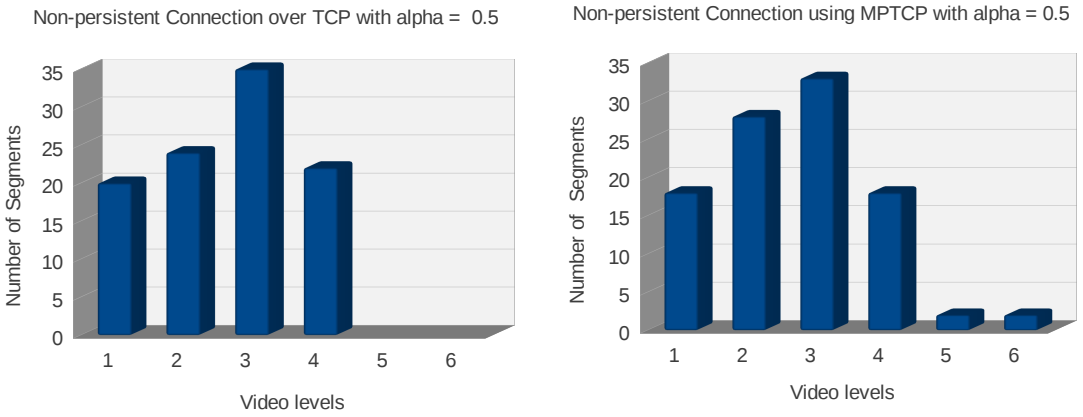


Figure 4.28: Number of segments requested at different levels in TCP and MPTCP using non-persistent connections.

4.6 Summary

In this chapter, we have validated each of the implemented streaming components. The experimental results show that the `bandwidth estimator` gives accurate results for both TCP and MPTCP streaming. The harmonic averaging used by the `bandwidth estimator` gives accurate estimation by removing the positive and negative spikes in the measured bandwidth. Further, we have evaluated the proposed `adaptation logic` for both TCP and MPTCP. We have found that the adaptation logic efficiently translates the achieved throughput into video quality (as represented by video levels) and allows to

correctly select the desired trade-off between efficiency and stability. Our results also emphasize the importance of the path-stability parameter in MPTCP streaming, which allows to overcome the burstiness and achieve a good performance.

This chapter also provides a performance comparison of MPTCP and TCP streaming in "equivalent" conditions. Our findings can be summarized as follows. We have found that TCP streaming performs better compared to MPTCP streaming under low delay and low packet loss. Under high packet loss, MPTCP streaming achieves a better performance compared to TCP in terms of the achieved throughput. As a result, it achieves a higher video quality on average, as evidenced by employing higher video levels throughout a streaming session. Finally, our comparison of streaming using non-persistent connections has shown that MPTCP performs better than TCP.

Chapter 5

Conclusion

To conclude our work in this document, a summary of the work, main results and possible directions for future work are given below.

The adaptation algorithms presented in this work were based on the use of new standards for adaptive streaming (DASH) and multi-path transport (MPTCP). In our work, a DASH streaming prototype has been successfully integrated with MPTCP. Every component of our system was designed for a specific functionality. We show how these components can be used effectively to constitute an adaptive streaming system. The bandwidth estimator uses pcap library, gives accurate measured bandwidth values and performs a short-term harmonic averaging over a dynamic window. It is effective for both TCP and MPTCP streaming, removes the outliers and reacts to abrupt changes in the bandwidth. Our method can be used to replace the erroneous bandwidth estimation in the original vlc-DASH plugin. The adaptation logic for both TCP and MPTCP streaming considers not only the available bandwidth but also the buffer size and user preferences for adaptation. It supports different functional modes and allows to optimize the trade-off between streaming efficiency and stability. Further, our experiments have demonstrated that in case of MPTCP streaming, frequent throughput variations may occur in scenarios with high delays, high packet losses and competing traffic, thus causing potentially disturbing variations of the video level. This behavior of MPTCP has necessitated the use of an additional path-stability parameter for the adaptation over MPTCP. We have found that the use of this parameter in MPTCP adaptation avoids frequent switching events and thus enhances the user experience.

Our experimental evaluation has included a validation of the implemented components, their integration in an adaptive streaming system and a performance comparison of TCP and MPTCP streaming. We summarize the main experimental findings as follows. We have validated that our multipath-streaming system supports the MPTCP by using

multiple available paths and continuing streaming on active paths when a path goes down. Next, the experiments show that video segments of size 6s achieve the best performance in terms of variation of the throughput and buffer size. Further, we have compared MPTCP and TCP streaming in terms of the number of used segments at each video level, under "equivalent conditions". Under low delay and low packet loss conditions, the buffer size grows faster with TCP compared to MPTCP, as a result of which TCP streaming performs better compared to MPTCP streaming. However, under high packet loss conditions, the MPTCP streaming achieves a better performance. As a result, MPTCP is better suited for adaptive streaming in high packet-loss environments like wireless networks. Finally, MPTCP outperforms TCP when using non-persistent connections, which would make it attractive for use in multi-server environments.

There is space for future work. On the algorithmic side, further work is needed on both the bandwidth estimation and adaptation. To fully evaluate the accuracy of our passive bandwidth estimation, a comparison with active techniques is necessary [30]. As for adaptation, it is still an open question how to optimally set the parameters used in our algorithm so as to maximize user satisfaction. To this end, a better understanding of the relationship between objective video-quality metrics and user satisfaction is required [11]. On the implementation side, for highly dynamic environments, a dynamic selection of the optimal segment size can be implemented. Further, our adaptation logic still needs to be evaluated in scenarios where multiple streaming clients share a bottleneck link [20]. It would be interesting to check if our adaptation is able to ensure fairness and maximize the overall performance. In addition, an automatic selection of functional modes of the algorithm in response to time-varying network conditions or user preferences would be a useful property in many practical streaming scenarios. Finally, the fundamental performance limitations of MPTCP and TCP streaming can be mitigated with novel protocols specifically designed to better support streaming applications, such as PRRT [13].

List of Figures

List of Tables

Appendix A

MPD Example

MPD is a XML document which describes the content information such as manifest of the available content, its alternatives, their URL addresses and other information. Below is the MPD file of the dataset used in our experiments. It lists the available segments in list format, each of duration 6 seconds on media time scale. The 'basic-cm' is the used profile and is identified by the URN "urn:mpeg:dash:profile:isoff-main:2011" and makes use of the following features:

- Single period per media presentation
- ISO Base Media File Format segments

We will restrict our explanation to only important fields here. 'profiles' field defines the profile to be used, its the mandatory field that defines the mode of operation of the DASH plugin. Static 'type' field specifies that the segment duration remains constant for the whole duration. 'mediaPresentationDuration' provides the duration of the whole video, which implies the duration of the streaming session. 'minBufferTime' specifies the amount of data should be buffered before starting the playout of the video. 'BaseURL' gives the URL of the parent folder which contains all the segment files. 'Period start' defines the starting time of the period, below example MPD have only one period. 'AdaptationSet bitstreamSwitching' attribute specifies whether adaptation is enabled or not. If true, then adaptation is supported else only a single bit rate video is used for the whole streaming session. 'Representation' attribute defines the next level in the hierarchy of the MPD structure, it specifies Codecs to be used and multiplexing type , supported width, height and also bandwidth required over the link to stream this representation. Initialization source URL is associated with each representation and is used to access the initialization segment at the beginning of the session.

```
1
2  <MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3       xmlns="urn:mpeg:DASH:schema:MPD:2011"
4       xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011"
5       profiles="urn:mpeg:dash:profile:isoff-main:2011"
6       type="static"
7       mediaPresentationDuration="PT0H9M56.46S"
8       minBufferTime="PT6.0S">
9       <BaseURL>http://pc-stud9/BigBuckBunny/bunny_6s/</BaseURL>
10      <Period start="PT0S">
11           <AdaptationSet bitstreamSwitching="true">
12 Representation id="0" codecs="avc1" mimeType="video/mp4" width="1920"
13 height="1080" startWithSAP="1" bandwidth="2078678">
14 <SegmentBase>
15     <Initialization sourceURL="/bunny_2500kbit_dash.mp4"/>
16 </SegmentBase>
17 <SegmentList duration="6">
18     <SegmentURL media="/bunny_6s1.m4s"/>
19     <SegmentURL media="/bunny_6s2.m4s"/>
20     <SegmentURL media="/bunny_6s3.m4s"/>
21     </SegmentList>
22 </Representation>
23 <Representation id="1" codecs="avc1" mimeType="video/mp4" width="1920"
24 height="1080" startWithSAP="1" bandwidth="2395950">
25 <SegmentBase>
26     <Initialization sourceURL="/bunny_3000kbit_dash.mp4"/>
27 </SegmentBase>
28 <SegmentList duration="6">
29     <SegmentURL media="/bunny_6s1.m4s"/>
30     <SegmentURL media="/bunny_6s2.m4s"/>
31     <SegmentURL media="/bunny_6s3.m4s"/>
32      </SegmentList>
33 </Representation>
34 <Representation id="2" codecs="avc1" mimeType="video/mp4" width="1920"
35 height="1080" startWithSAP="1" bandwidth="2919314">
36 <SegmentBase>
37     <Initialization sourceURL="/bunny_4000kbit_dash.mp4"/>
38 </SegmentBase>
39 <SegmentList duration="6">
40     <SegmentURL media="/bunny_6s1.m4s"/>
41     <SegmentURL media="/bunny_6s2.m4s"/>
42     <SegmentURL media="/bunny_6s3.m4s"/>
43     </SegmentList>
44 </Representation>
45           </AdaptationSet>
46     </Period>
47 </MPD>
```

'Duration' corresponds to the duration of each segment on the media time scale. In the above example, we have shown only 3 segment files in each representation, but the actual MPD contains more, depending on the segment as well as program duration. For example, program with duration of 596 seconds and segment of 6 seconds number of segments are approximately 99 . With 2 seconds segments, approximately 298 segments and so on.

Bibliography

[1] M. Handley A. Ford, C. Raiciu and J. Iyengar S. Barre. Architectural guidelines for multipath tcp development. http://tools.ietf.org/rfc/rfc6182.txt, March 2011.

[2] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proceedings of the second annual ACM conference on Multimedia systems*, MMSys 2011, pages 157–168, New York, NY, USA, 2011.

[3] John Apostolopoulos. Reliable video communication over lossy packet networks using multiple state encoding and path diversity. In *Visual Communications and Image Processing (VCIP)*, pages 392–409, 2001.

[4] Sébastien Barré, Christoph Paasch, and Olivier Bonaventure. Multipath TCP: From theory to practice. In *IFIP Networking, Valencia*, May 2011.

[5] Ali Begen, Tankut Akgul, and Mark Baugher. Watching video over the web: Part 1: Streaming protocols. *IEEE Internet Computing*, 15(2):54–63, March 2011.

[6] M. Handly C. Raiciu and D. Wischik. Coupled congestion control for multipath transport protocols. http://tools.ietf.org/rfc/rfc6356.txt, October 2011.

[7] Stefan Lederer Christopher Müller and Christian Timmerer. An evaluation of dynamic adaptive streaming over http in vehicular environments. In *In Proceedings of the ACM Multimedia Systems Conference 2012 and the 4th ACM Workshop on Mobile Video*, Chapel Hill, North Carolina, February 2012.

[8] Microsoft corporation. Microsoft smooth streaming. http://www.iis.net/download/smoothstreaming, 2012.

[9] Mark Handley Costin Raiciu, Damon Wischik. Practical congestion control for multipath transport protocols. In *Universirty of College London Technical Report*, 2011.

[10] Philippe de Cuetos. Network and content adaptive streaming of layered–encoded video over the internet, 2003.

[11] Florin Dobrian, Asad Awan, Dilip Joseph, Aditya Ganjam, Jibin Zhan, Vyas Sekar, Ion Stoica, and Hui Zhang. Understanding the impact of video quality on user engagement. *Communications of the ACM*, 56(3):91–99, 2013.

[12] M. Bagnulo E. Nordmark. Shim6: Level 3 multihoming shim protocol for ipv6. http://tools.ietf.org/rfc/rfc5533.txt, June 2009.

[13] M. Gorius, Y. Shuai, and Th. Herfet. Dynamic media streaming under predictable reliability. In *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (ISBMSB), Seoul*, June 2012.

[14] Stephane Gouache, Guillaume Bichot, and Christopher Howson. Experimentation of multipath HTTP streaming over internet. In *NEM summit*, 2011.

[15] R Frederick H Schulzrinne, S Casner and V Jacobson. RTP: A transport protocol for real-time applications. http://tools.ietf.org/rfc/rfc3550.txt, July 2003.

[16] C. Hopps. Analysis of an equal-cost multi-path algorithm. http://tools.ietf.org/rfc/rfc2992.txt, Novemeber 2000.

[17] Adobe Inc. Adobe http dynamic streaming. http://www.adobe.com/products/hds-dynamic-streaming.html?promoid=GYMXT, 2012. (last access: Dec. 2012).

[18] J. Mogul Irvine, J.Gettys, P.Leach H.Frystyk, L.Masinter, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. http://www.ietf.org/rfc/rfc2616.txt, June 1999.

[19] Janardhan R. Iyengar, Keyur C. Shah, Paul D. Amer, and Randall Stewart. Concurrent multipath transfer using SCTP multihoming. In *Technical Report, University of Delaware*, 2004.

[20] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. 2012.

[21] Stefan Lederer, Christopher Müller, and Christian Timmerer. Dynamic adaptive streaming over HTTP dataset. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 89–94, New York, NY, USA, 2012. ACM.

[22] E. Blanton M. Allman, V. Paxson. Tcp congestion control. http://tools.ietf.org/rfc/rfc5681.txt, September 2009.

[23] J. Padhye M. Handley, S. Floyd and J. Widmer. TCP friendly rate control (TFRC): Protocol specification. http://www.ietf.org/rfc/rfc3448.txt, Jan 2003.

[24] Shiwen Mao, Shunan Lin, Shivendra S. Panwar, Yao Wang, and Emre Celebi. Video transport over ad hoc networks: multistream coding with multipath transport. volume 21, pages 1721–1737, 2003.

[25] Christopher Müller and Christian Timmerer. A vlc media player plugin enabling dynamic adaptive streaming over http. In *In Proceedings of the ACM Multimedia 2011 , Scottsdale, Arizona*, November 2011.

[26] IP networking Lab. Multipath TCP - linux kernel implementation. http://mptcp.info.ucl.ac.be/.

[27] May Pantos, R. Http live streaming. http://tools.ietf.org/html/draft-pantos-http-live-streaming-04.

[28] M. Reha Civanlar Pascal Frossard, Juan Carlos de Martin. Media streaming with network diversity. In *Proceedings of IEEE ,Volume: 96 , Issue: 1 Page(s): 39 - 53*, Jan 2008.

[29] J. Postel. RFC 793: Transmission control protocol. http://rfc.sunsite.dk/rfc/rfc793.html, September 1981.

[30] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy, Ravi Prasad, and Constantinos Dovrolis Georgia. Bandwidth estimation: Metrics, measurement techniques, and tools. *IEEE Network*, 17:27–35, 2003.

[31] P. Nikander R. Moskowitz. Host identity protocol (HIP) architecture, may 2006.

[32] Costin Raiciu, Christoph Paasch, Sébastien Barré, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? designing and implementing a deployable multipath TCP. In *USENIX Symposium of Networked Systems Design and Implementation (NSDI'12), San Jose (CA)*, 2012.

[33] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. In *IEEE Multimedia*, pages 62 – 67, OctoberâĂŞDecember 2011.

[34] Thomas Stockhammer. Dynamic adaptive streaming over HTTP –: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, MMSys '11, pages 133–144, New York, NY, USA, 2011. ACM.

[35] Saarland University Telecommunication Lab. Future media internet (FMI) (Video-Audiotransport - A New Paradigm). 2011.

[36] Klagenfurt University. Dash plugin for vlc. http://www-itec.uni-klu.ac.at/dash/, June 2012.

[37] Bing Wang, Jim Kurose, Prashant Shenoy, and Don Towsley. Multimedia streaming via TCP: an analytic performance study. In *Proceedings of the 12th annual ACM international conference on Multimedia*, MULTIMEDIA '04, pages 908–915, New York, NY, USA, 2004. ACM.

[38] Damon Wischik, Mark Handley, and Marcelo Bagnulo Braun. The resource pooling principle. *SIGCOMM Comput. Commun. Rev.*, 38(5):47–52, September 2008.