UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Combining DASH with MPTCP for video streaming

Dissertation presented by
Lena Victoria PESCHKE

for obtaining the Master's degree in
Computer Science and Engineering
Option: Networking and security

Supervisor
Prof. Olivier BONAVENTURE

Readers
Benjamin HESMANS, Guillaume MAUDOUX

Academic year 2016–2017

Abstract

HTTP adaptive streaming (HAS) is widely deployed on today's Internet. It tailors a video streaming session to the end-user's needs by dynamically adapting the quality to the network conditions. Dynamic Adaptive Streaming over HTTP (DASH) is a recent standard attempting to unify similar approaches to HAS. Multipath TCP (MPTCP) is an extension of TCP that is able to bundle several connections and present it as if it were one to the application layer. It could improve video streaming when multiple network interfaces are available, such as with mobile devices. How transparently replacing TCP with MPTCP in DASH video sessions changes the streaming performance has not yet been clearly assessed. This thesis analyses how combining DASH with MPTCP affects the quality of experience (QoE) perceived by the end-user under equivalent network conditions as TCP. While not providing statistically relevant results, it gives insights into factors improving or diminishing performance. Several network setups with two paths between the client and the server are compared against each other and TCP. In ideal network conditions, without any loss or congestion, MPTCP mostly performs similar to TCP. This can be explained through application and transport layer analysis. When two competing clients are present on the network, MPTCP tends to offers less stability and a lower network utilisation than TCP.

Acknowledgements

I would like to thank my supervisor Prof. Olivier Bonaventure for his high availability, his invaluable input, and his guidance.

I would like to express my gratitude to Benjamin Hesmans and Quentin De Coninck for their help and the discussions on all aspects of this work, theoretical and technical.

I also wish to thank Prof. Niklas Carlsson for his class on networking and the talks about this thesis.

A special thanks goes to the *3 Génisses*, for everything else.

Finally, I would like to express my deepest gratitude to my friends and family for their encouragements and their constant support.

Contents

Introduction

Today's Internet is bigger than ever and still growing fast. Cisco forecasts that global IP Internet traffic will have increased by a factor of 95 from 2005 to 2020. Video streaming has become a major part of that traffic. According to Cisco, "IP video traffic will be 82% of all consumer Internet traffic by 2020, up from 70% in 2015" ([13]). The increase in online video content is accompanied by an increase in video quality, which in turn increases the network throughput demands [57]. In this context, HTTP video streaming (HAS) has received much attention lately. It tries to improve the streaming experience for the end-user by adjusting the quality of the feed to the available resources. Popular streaming services, such as YouTube and Netflix already use it [38]. Dynamic Adaptive Streaming over HTTP (DASH) is a recent standard seeking to unify similar approaches.

At the same time, the percentage of mobile devices connected to the Internet has increased in the past years. While smartphone traffic accounted for 8% of all global IP traffic in 2015, it will grow to 30% by 2020 [13]. Tablets and smartphones are often multihomed, having one WiFi and one cellular network interface, but can only use one at a time. This is where Multipath TCP (MPTCP) comes into play. By extending TCP, it can transparently aggregate several network paths into one transport layer connection.

Both MPTCP and DASH are no new protocols but extensions. MPTCP tries to perform at least as good as TCP while allowing multiple paths per connection. DASH works over-the-top and uses HTTP to deliver media content. The advantage of reusing TCP and HTTP is that the Internet infrastructure is designed and built around them. Both are adaptive to the network. They take their environment and the changing conditions into account and react to it.

The goal of this thesis is to assess how MPTCP and DASH interact when used together. Does the combination improve or degrade the video streaming experience? I will compare the quality of a video session when DASH is used with MPTCP and regular TCP in different network setups, paying special attention to quality of experience (QoE). In a further step, the application level performance is linked to events at the transport layer.

Chapter 1 serves as an introduction to HAS and the recent DASH standard. It outlines some key issues and characteristics of adaptive streaming to remember throughout this thesis. The importance of user experience as a performance framework is subsequently explained. Chapter 2 reviews how MPTCP was designed and how it works. A few use cases related to video streaming are reviewed as a starting point for the combination. In chapter 3, I present the simulation infrastructure used for testing. This leads to chapter 4, where several TCP baseline cases are defined and analysed to pave the way for chapter 5, where DASH and MPTCP are combined. Different setups using a single video player are compared against each other to uncover the parameters most influential for a good or bad streaming performance. Finally, chapter 6 presents a first assessment of DASH used with MPTCP in a congested network, followed by a chapter drawing conclusions.

1 | DASH

Video streaming over the Internet can be achieved in many different ways. In this chapter I assess how dynamic adaptive streaming over HTTP compares to other strategies and present the MPEG-DASH standard. I then review different quantitative and qualitative evaluation parameters and discuss a series of client-side adaptation schemes, which are at the core of DASH's behaviour. This review of the state of the art serves a double purpose: to understand how the technology works and to identify the most important parameters that define its performance when used with TCP.

1.1 Types of multimedia streaming

Video streaming combines video download with concurrent playback [48]. It can be found in three general forms: *on-demand*, *live*, and *real-time interactive*. On-demand streaming applies to all content recorded entirely, sent at some point after that, and then played. If the player supports it, it offers the ability of pausing, rewinding, and fast-forwarding. Live streaming delivers content in real-time. Sport events and internet radio are prominent examples. Real-time interactive multimedia generally involves several end-users exchanging video and audio content. In addition to download and playback, it uses video upload. IP telephony, video conferencing, and online gaming are the most well-known applications.

Multimedia streaming depends on a network and is affected by its characteristics. Streaming is a real-time application with soft timing constraints. It is delay-sensitive, especially to delay jitter, and generally loss-tolerant. Traffic burstiness also affects its performance. Buffering at the application level is used to mitigate the effects of all three.

In the following, I will only consider on-demand video streaming, also known as video-on-demand (VOD). It allows to use existing videos and to stream them as often as needed. The files are static for the time of the session. The streaming server does not have additional video processing overhead, such as encoding and frame re-ordering.

Different network protocols can be used to stream a video. The methods can be divided into the group that uses dedicated ones, and the one that uses HTTP.

1.1.1 Dedicated protocols

Real-Time Transport Protocol (RTP) is a stateful application-layer protocol. It usually uses UDP and allows low overhead transmission of audio and video packets [49]. The server is responsible for the transmission and has to preserve state for every session. Nowadays, content delivery networks (CDN) do not support RTP well. Moreover, firewalls tend to drop RTP packets [49].

Real-Time Streaming Protocol (RTSP) is a stateful application-layer protocol that uses RTP in conjunction with UDP or TCP. It has out-of-band control, so that

the user can send control messages to the server. When a client connects to the server, a session is established and maintained at the server. The server then sends a stream of data to the client.

1.1.2 HTTP-based

Since the advent of the Internet, video compression has become more efficient, available bandwidth has increased, connectivity has become more dense, and HTTP can deliver larger segments [2, 49]. The Internet has evolved around HTTP. HTTP-based streaming works "over-the-top" and uses unmodified HTTP [11, 37]. It takes advantage of the existing Internet infrastructure: it reuses caching solutions and CDNs, does not add new middlebox traversal issues, and simplifies its own deployment thanks to the popularity of HTTP and TCP/IP [11, 35, 37, 51]. The server can be stateless because the responsibility for the streaming session can be delegated to the client, which was not the case with the previous protocols. [11, 49, 51].

Progressive download

With progressive download, a server stores one or several versions of a media file and advertises their location through a manifest file (or index file). To play a video, users first request the manifest. Based on what is available, they choose the characteristics of the video (quality, bitrate, frame rate, resolution, etc.) and keep them for the entire session. They then send byte-range requests to progressively download the video from the server, relying on HTTP to take care of the entire delivery. This is sometimes referred to as pseudo-streaming. The technique is unsuitable for mobile environments, where bandwidth varies considerably more than in static environments [29]. Progressive download does not support live streaming [51], unless the video file is sliced into segments and the manifest supports it.

Adaptive streaming

HTTP adaptive streaming (HAS), or adaptive bitrate streaming (ABS), is an extension of progressive download. The server stores several representations of the video, encoded at different bitrates or resolutions, and possibly sliced into smaller segments of constant duration. During a session, dynamic adaptation to a different version is possible. The goal is to provide a better playback experience by taking the available resources into account.

Most methods target smooth streaming by adapting to the network conditions and are designed to cope with bandwidth variations during a session [12, 35], but also CPU capacity, the state of the buffer, the server load, etc. can be taken into account. If the adaptation responsibility lies at the client, the player fetches the manifest listing the available segments and their characteristics. Based on an adaptation strategy, the player dynamically chooses the characteristics of the individual video segments, based on availability, network conditions, playback conditions, and user input [38, 48]. It can decide to switch to a different representation at fixed time instants. This offers several advantages: video quality adaptation can be done on-the-fly while all the logic remains at the client side. The client knows its own bandwidth, available codecs, constraints, and preferences [26, 35, 51]. Compared to progressive download, HAS achieves a better bandwidth utilisation and decreases playback interruptions [48].

The complexity of HAS lies in the adaptation strategy, which needs to be carefully designed to achieve good results. The client essentially has to decide when to adapt (which events to

take into account), and which representation to request (how to react) [48]. In conjunction with TCP's congestion control, this adaptation based on HTTP observations creates a so called "nested double feedback loop" ([2]), the outcome of which is rather complex to model. The possible problems are explained in [48]. In [11], the authors further investigate what they call the "downward spiral effect". It stems from negative interaction between HTTP and TCP's congestion control when competing TCP traffic is present on the network and results in a bandwidth underuse. Furthermore, HAS results cannot only be measured in terms of quality of service (QoS) metrics of the network, because users expect a good quality of experience (QoE) too; QoS and QoE do always go hand in hand, as discussed below in section 1.3. Server-side adaption strategies exist, but they require control over the server (e.g. YouTube) [11]. Hybrid solutions using control messages [16] are another option. They will not be discussed here.

The difference between progressive download and HAS is illustrated in figure 1.1.

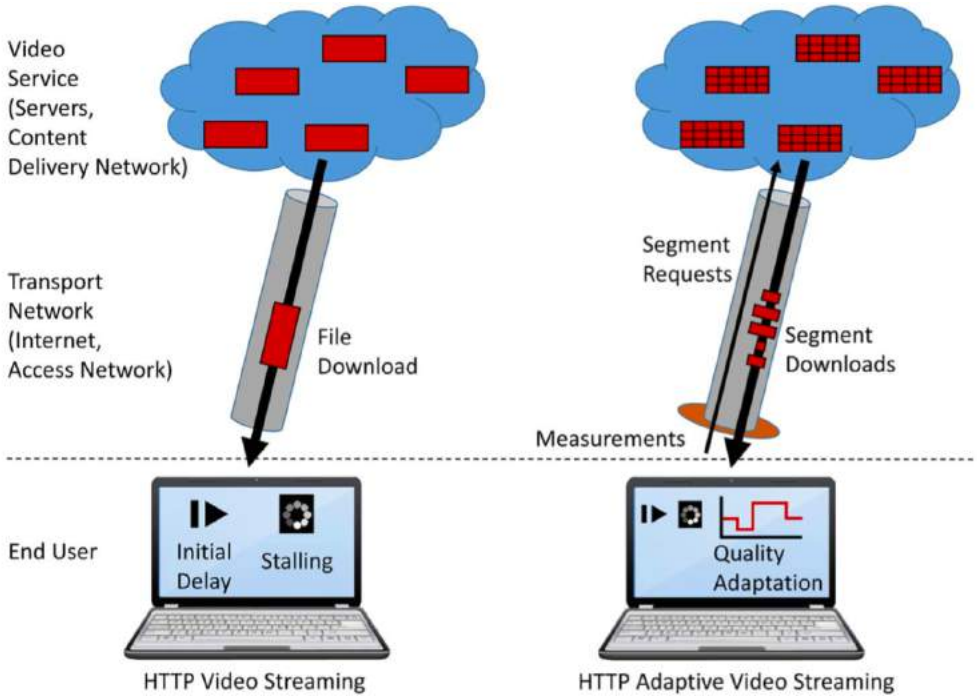


Figure 1.1: Comparison of HTTP video streaming and HTTP adaptive video streaming. End user is not aware of service or network influences, but only perceives initial delays, stalling, and quality adaptations. [48, p. 4]

1.2 MPEG-DASH

Dynamic adaptive streaming over HTTP (DASH) is a codec agnostic adaptive bitrate streaming technique for multimedia content based on HAS. It supports live streaming and time-shift viewing, but, as mentioned above, I will limit the scope to VOD.

The multimedia content stored on the server is twofold: the media presentation description (MPD) and the segments, stored in one or more files [49]. The MPD is a manifest containing the locations of the various segments and the different available versions of it, most importantly the different bitrates and resolutions [35]. To initiate a session, the client first requests the MPD from the server. Based on the information received from it, user preferences, and its capabilities, it chooses the most fitting representation of the content [49, 51]. The client then sequentially requests the segments via HTTP

GET requests, decodes them, and puts them together to play the medium. During the streaming session, the client can switch to a different representation on a per-segment basis.

1.2.1 Scope

DASH essentially specifies a media presentation on the client side. The client is responsible for the smooth progress of the session [49, 51]. It can choose how to interpret the MPD and which segments to request next, based on any type of input. Figure 1.2 shows the scope of the MPEG-DASH standard.

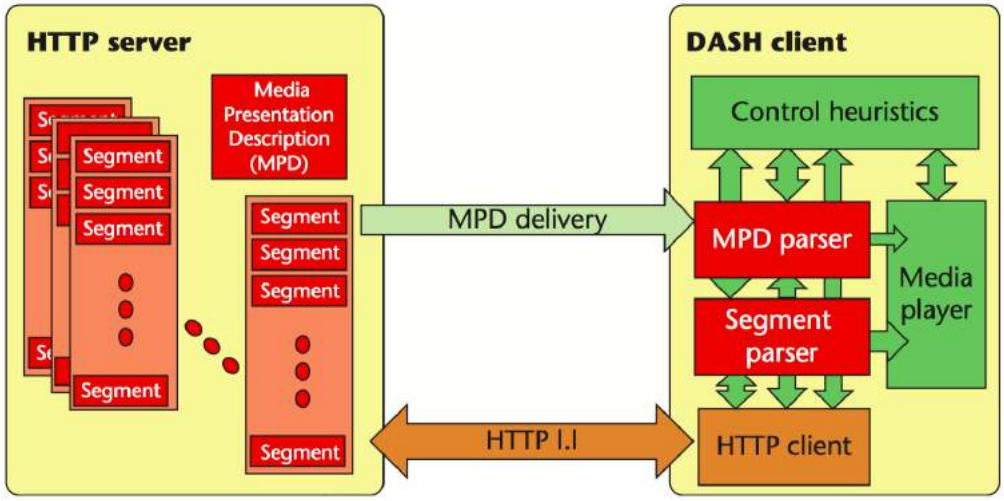


Figure 1.2: Scope of the MPEG-DASH standard. The formats and functionalities of the red blocks are defined by the specification. The clients control heuristics and media players, which aren't within the standard's scope. [49, p. 64]

DASH is an attempt to unify different stream-switching HAS solutions [5, 49], most notably Apple's HTTP Live Streaming [43], Microsoft's Smooth Streaming [59] and Adobe's HTTP Dynamic Streaming [1]. It aims at providing interoperability between servers and devices of different vendors [5, 49]. [48] gives a comparison of different HAS solutions. MPEG-DASH was first published as a standard in ISO/IEC 23009:2012 in 2012 and has since then been revised by ISO/IEC 23009:2014 in 2014 [23]. The first part defines the "Media presentation description and segment formats". The subsequent parts address the issue of conformity, provide implementation guidelines, and tackle encryption and authentication of segments.

1.2.2 Media Presentation Description

The media presentation description (MPD) is the manifest used by DASH. It is the starting point file for any DASH medium. An MPD is an XML document describing a medium. It contains information about the medium's segments and their location, metadata, and the timing [32]. Several different components can be involved, most often video, audio, and text [49]. This information must at least be sufficient for the client to access the segments using the HTTP protocol [23].

It is organised in hierarchical levels, as represented in figure 1.3.

Periods contain non-overlapping parts of the content with a start time and a duration. Multiple periods can be used to split a video into chapters, or to separate advertisement from the actual content.

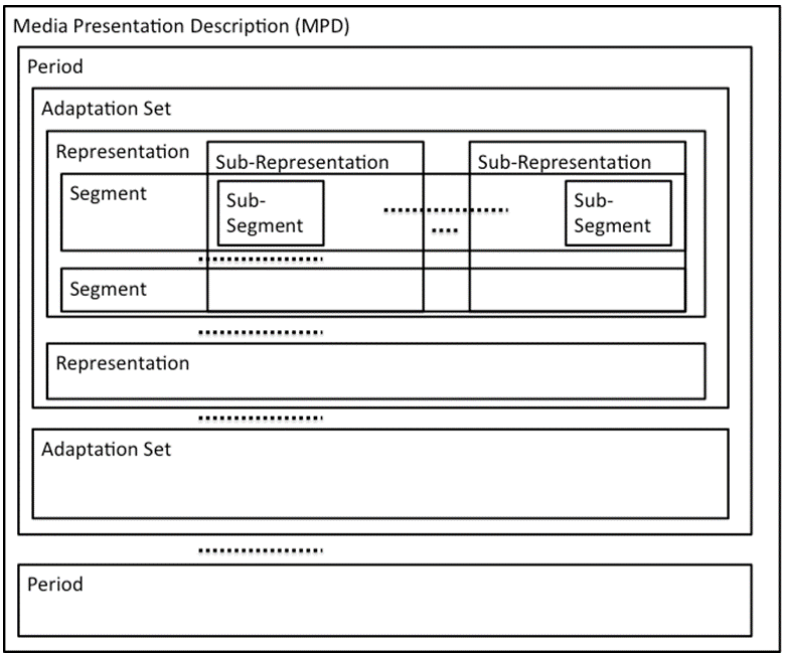


Figure 1.3: DASH high-level MPD model. [23, p. 9].

Adaptation sets contain one or more representations that logically belong together, such as the different encodings of a video [36, 49]. The set is chosen based on user preferences, manually or automatically. A single adaptation set can contain a certain type of media stream (e.g.: video, audio, subtitles) or multiplexed content.

Representations are different versions of the same media content. This can include one or several multiplexed media streams. The difference can lie in the resolution, the bitrate, the codecs [49]. Having multiple representations makes it possible for the client to automatically select the best fitting representation based on hardware and software support, and to dynamically adapt to network conditions. The only required information are the encoding bitrate and an ID.

Sub-Representations describe properties of a single stream inside a representation. For example, a representation can have an audio and a video stream. The sub-representation could give the codec, the sampling rate, or any other information the player might need to extract the content of the specific stream.

Segments are defined as "integral data unit[s] of a media presentation that can be uniquely referenced by a HTTP-URL (possibly restricted by a byte range)" ([51]). They are the actual media stream chunks assembled and played by the client. They have a unique URL, a start time, and a duration. Only one segment can be retrieved with a single HTTP request.

Sub-Segments are a subdivision of segments by byte-range.

1.3 Evaluation parameters

As with any application, there is a need to assess the performance of DASH. This becomes even more important as many different solutions are being published. To be able to compare them, a common set of parameters needs to be defined. I will review quality of service as well as quality of experience. While there has been a chronological evolution from the first to the second, they are nowadays complementary.

1.3.1 Quality of Service (QoS)

QoS metrics are quantitative measures of the quality of a network in terms of bandwidth, throughput, delay, jitter, packet loss and corruption. They are useful to describe the resources available to applications. While these metrics objectively describe the network and impact on the applications running on top of it, they are not sufficient to characterise video streaming applications because they do not exclusively determine the human experience of video watching [38]. To take more human factors into account than the technical ones QoS uses, QoE has emerged as a multidisciplinary measure.

1.3.2 Quality of Experience (QoE)

The International Telecommunication Union Telecommunication Standardization Sector (ITU-T) defines QoE as "the overall acceptability of an application or service, as perceived subjectively by the end-user. Quality of experience includes the complete end-to-end system effects (client, terminal, network, services infrastructure, etc.). Overall acceptability may be influenced by user expectations and context." ([24]). The authors of [48] claim QoE to be "the most important performance metric [for] video services".

A set of QoE factors for DASH-based services can be identified:

Start-up delay or initial delay: the time between the "service request" and the playback. It is composed of the time necessary to get and parse the MPD, to initialise the adaptation algorithm, and to fill the buffer up to the initial threshold.

Stalls or freezes, rebuffering, buffer underruns, playback interruptions. They have a frequency, a duration, and a position relative to the video playback.

Quality switches or bitrate changes: refer to the moments where the adaptation algorithm requests a different representation for the player. They can be measured in terms of frequency and amplitude.

Media throughput or average video bitrate: the "media bits per second". Per encoding scheme, a higher throughput gives a higher video quality.

In [46], Rainer and Timmerer use crowdsourcing to analyse the QoE performance of web-based DASH clients on the Internet. They come to the conclusion that the start-up delay has no significant influence on the QoE. They identify the number of stalls and the average media throughput as the most important factors.

In [56], Timmerer et al. recommend low start-up delay, although is has more impact on live and short videos than on long VOD. Stalls should be avoided at all cost, even if that increases start-up delay, and the frequency and amplitude of quality switches should be minimised. The media throughput gives a good overall measure, but is not enough to assess the performance of a system.

In [48], Seufert et al. corroborate the result that start-up delay does not have a severe impact on perceived quality, especially when compared to the other parameters. However, this does not apply to "video browsing", where users do not initially intend to watch the requested video, and instead look at the first few seconds to decide whether to continue or not. Stalling is the worst influencing factor. When it cannot be avoided entirely, it should be subject to the following tradeoff: frequent stalls are worse than one long, and periodic stalling is preferred over irregular intervals. Quality switching is divided into upgrading and downgrading: while the first has either little negative or even a positive effect on QoE, the second is always bad. It should therefore be contained, especially its

frequency, when the difference is perceivable. In terms of amplitude, a stepwise decrease seems to be less annoying than a sudden one.

In [38], Nam, Kim, and Schulzrinne analyse the QoE factors that lead users to abandon YouTube videos. They conclude that stalls are by far the most decisive factor and should be avoided; frequent rebuffering is worse than a single long one. Quality switches are the second. Both up and down switching is perceived negatively.

1.3.3 End-to-end measurements

Performance is most often analysed at the client. This allows to assess the quality of HAS and detect problems, but is not always sufficient to determine the root causes of theses problems. In [19], Ghasemi et al. show that partial information about the streaming process can lead the adaptation logic to come to the wrong conclusions about the network and to consequently make wrong decisions. Their research uses real environments with complex end-to-end delivery paths, including CDNs. The authors collect data at the client and the server to assess the performance of HAS. They analyse the data path from end-to-end to uncover performance bottlenecks and component interactions. They are able to trace the root causes to transient and persistent problems occurring during a streaming session. However, this approach requires both, more resources than a simple client-side or server-side adaptation and access to all entities participating in the streaming session. It is useful when first designing and testing an algorithm, but impractical in a production environment.

1.4 Client-side adaptation schemes

With client-side adaptation, the adaptation algorithm is free to be implemented by the client. Different heuristics can be used in order to achieve the best viewing experience for the user. In general, the client needs to correctly estimate and predict the network conditions in order to achieve a high quality, but also to control the buffer level to avoid stalls [5, 38, 48, 50].

A number of implemented adaptation algorithms exist for DASH with single-layer content.[1]

Liu: In [31], bandwidth estimation based on application layer measurements and scheduling according to the buffer state are introduced and combined into an adaptation logic. Step-wise up switching and aggressive down switching avoid stalls and smoothen the playback.

Miller: In [33], the authors pursue the following goals, in decreasing priority: 1) avoid playback stalls, 2) maximise the minimum and average representation level, 3) minimise the amount of switches, 4) minimise the start-up delay. The start-up delay is tackled by always selecting the lowest quality for the first segment. To achieve the second goal nonetheless, the quality is then aggressively increased. The algorithm measures the throughput of the previous segments and the buffer level. It calculates the next representation level and the delay before download should start, expressed as a buffer level threshold. The algorithm is calibrated through buffer level thresholds; it tries to keep the level close to a certain value B_{opt} inside a target interval $\mathcal{B}_{tar}$. While inside $\mathcal{B}_{tar}$, the quality stays constant. When rising above or falling under the limit, quality may be adapted if a long-term variation can be observed.

[1] Single-layer codecs encode a video with one representation per file. Multi-layer codecs encode several quality levels of a video into a single file, making bitstream adaptation possible without the need to fetch a different file.

There is a last minimum threshold B_{min} under which quality is always decreased to the minimum available level to avoid buffer underruns.

Tian-Liu: The algorithm is designed to balance responsiveness to network fluctuations and playback smoothness. It uses a control-loop feedback mechanism to drive the buffer level to some set point. This computes the representation level to choose for the next segment. To avoid frequent quality switching, various other metrics are taken into account to smoothen the increase. A threshold is set to allow for sharp decrease, which avoids freezes. In order not to overstress TCP and not to overflow the buffer, the algorithm also uses a "video rate margin" and a "buffer cap". These simultaneously reduces buffer level oscillations and video quality fluctuations. Furthermore, the authors introduce the possibility of using more than a single server during one video session. They create two scenarios. In the first, the client chooses the server with the highest TCP throughput estimation. In the second, the client establishes several TCP connections to different servers and requests different chunks from each. The throughput estimation is computed over the aggregation of connections. The scheduling used to determine which chunk will be requested from which server is self-adaptive. The client creates a common chunk download request queue. Every time a download is finished, the connection takes on the next request from the queue. To quickly react to failure, a timer is associated with each GET request. Upon expiration, if the download is not finished, the request is sent again over a different connection. [55]

FESTIVE: The design goals are TCP fairness, efficiency to maximise user experience, and playback stability. The algorithm is composed of three components. First, a *harmonic bandwidth estimator* computing a throughput estimation over 20 observations for robustness. Second, a *stateful and delayed bitrate update module* which throttles the increase of the reference bitrate but allows a quick decrease, smoothing out quality switches while still avoiding stalls. Third, a *randomised scheduler* to introduce some jitter in the download period, which in turn makes the bandwidth estimation less periodic. This last component avoids the potentially negative feedback loop interaction. [26]

PANDA: The algorithm addresses the issues of competing HAS sessions: unfairness and bitrate oscillation. Its basic design goals include: avoiding stalls, a smooth quality, a high average bitrate. It uses the "probe-and-adapt" principle. Similar to TCP congestion control, it uses AIMD to estimate the available bandwidth. Its scheduling component adapts the segment inter-request to achieve better probing results and to keep the buffer above a certain threshold. [30]

ELASTIC: The algorithm has four design goals: minimise the stalls, maximise the average bitrate, minimise the quality switches, and provide TCP fairness. It uses feedback control theory with a single controller to drive the buffer level to a constant value by adapting the bitrates of the video segments. [16]

DASH-JS: The player holds a bandwidth estimator "to measure and estimate the current effective throughput". At each downloaded segment i, the measured bandwidth b_i is recorded and the throughput t_i is estimated: $t_i = \frac{0.7t_{i-1}+1.3b_{i-1}}{0.7+1.3}$, where t_{i-1} is the previously computed throughput at segment $i-1$, b_{i-1} is the measured bandwidth at segment $i-1$. The estimation is used to directly determine the next quality to request. This rate-based algorithm quickly reacts to bandwidth variations. [47]

BBA-0: In [22], the authors design a buffer based algorithm which picks the video quality level solely based on the current buffer fill state. The algorithm has two goals:

avoid stalls and maximise the average video bitrate. The function controlling the algorithm is defined by pieces on the buffer occupancy/video rate plane. It should always keep the buffer occupancy above a certain threshold: between an empty buffer and B_1, the chosen rate is constant, the minimum available R_{min}. Between this lower B_1 and an upper threshold B_m, the function increases linearly from R_{min} to the maximum representation R_{max}. Past B_m and until the upper limit of the buffer B_{max}, the chosen rate is again a constant, but the maximum available this time. This simple algorithm can be improved by using rate-based adaptation during the start-up phase.

ABR logic (implemented in dash.js v1.5.0): The algorithm works with four rules with increasing priority, two primary bandwidth rules and two secondary buffer based rules. The *throughput rule* computes a moving average over the last 2 or 3 requested and downloaded video segments to predict the future throughput. When possible, the player switches to a higher representation. The *abandon request rule* tracks the instantaneous throughput during each segment download. If the ratio between the estimated time to download the segment and its duration crosses a threshold of 1.5, the player abandons the current segment. It switches down to a lower appropriate representation and downloads the same segment again. This rule makes the player responsive to bandwidth drops. The *buffer occupancy rule* overrides the throughput rule and prevents a down switch when there is enough buffered data. This prevents quality oscillations. The *insufficient buffer rule* switches to the highest representation for the next segment whenever the buffer gets empty to prevent and quickly recover from stalls. [15]

BOLA (implemented in dash.js v2.0.0): The algorithm is based on a utility maximisation problem where the two parameters to maximise are the average video bitrate, the *playback utility*, and the duration not spent on stalls, the *playback smoothness*. The solution presented is a buffer-based algorithm that chooses the quality of the next segment by looking at current bitrate and the buffer fill. The complete algorithm is described in [50] and replaces the ABR logic in the dash.js reference player [15].

Some of the aforementioned algorithms are compared against each other in [16], [30], [56], and [50]. In [11], the authors assess the performance of throughput based adaptation algorithms with competing traffic. They recommend trying to use a high percentage of the available bandwidth to mitigate underuse, the use of medians and quantiles instead of throughput averages to lessen the impact of outliers, and big video segments to get a stable playback quality and buffer fill. Their observations ultimately lead them to conclude that buffer-based adaptation algorithms offer the correct separation of responsibilities between the TCP and the HTTP layers. In [58], the authors define a formal model of QoE combining the average video quality, the average quality variation, and the amount of stalls. They compare different algorithms and show that buffer-based adaptation algorithms outperform rate-based ones, especially with regard to bandwidth variability. In [22], the authors argue that buffer-based adaptation gives the better results while being less complex than bandwidth prediction, but that the rate-based approach is useful for the start-up phase.

2 | Multipath TCP

MPTCP adds multipath functionality at the transport layer. Its use is interesting in cases where bandwidth can be increased through path aggregation. This chapter describes its design (section 2.1) and functioning (section 2.2) in relation with TCP assuming the reader has basic knowledge of TCP. Section 2.3 collects use cases and performance evaluations relevant to video streaming.

2.1 Motivation and goals

The transmission control protocol (TCP) is the most widely deployed transport layer protocol on the Internet. It offers reliable and ordered data transmission. Connections are identified by the four-tuple <source IP, source port, destination IP, destination port>, tying them to one network address at each endpoint. If there is a change in addresses or a failure, the connection fails.

While TCP thus remains a single-path protocol, the networks it operates on have become more multipath. Nowadays mobile devices often have more than one interface: WiFi alongside cellular. Datacenters use redundant paths in order to increase load-balancing, augment failure resilience, and lower the costs with cheaper commodity hardware. [8, 44]

This creates a mismatch between hardware and network architecture advances and TCP's capabilities [45]. Multipath TCP (MPTCP) is a modification of TCP enabling multiple paths between two hosts to be joined into one transport layer connection. MPTCP decouples the transport and network layers and provides for multihoming. Its main design goals are [8, 41, 44]:

1. Transparent compatibility with unmodified applications using TCP.
2. Compatibility with today's networks: it must work where TCP works.
3. Efficient utilisation of the network to perform at least as well as TCP.
4. Fairness in resource sharing, especially towards TCP.
5. Implementation without excessive memory or processing overhead.

2.2 How it works

For MPTCP to work, only the network stack at the endpoints need to be modified. In the network stack (figure 2.1), it acts as a shim layer between one or several TCP connections in the transport layer – the subflows – and the socket API of the application layer [41, 45]. This makes it easy to deploy, since it supports unmodified applications using TCP's socket API and is (mostly) compatible with the Internet of today. More insight into middlebox compatibility can be found in [41] and [44].

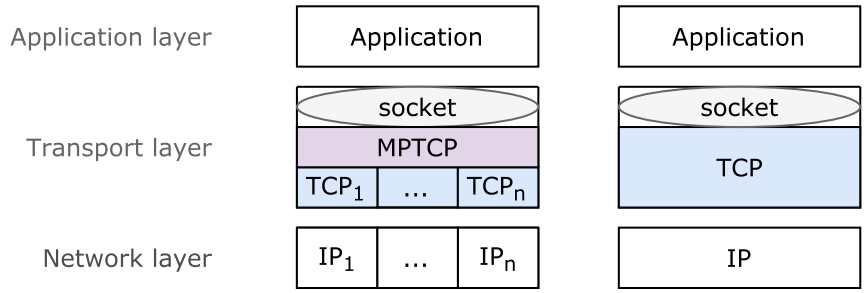
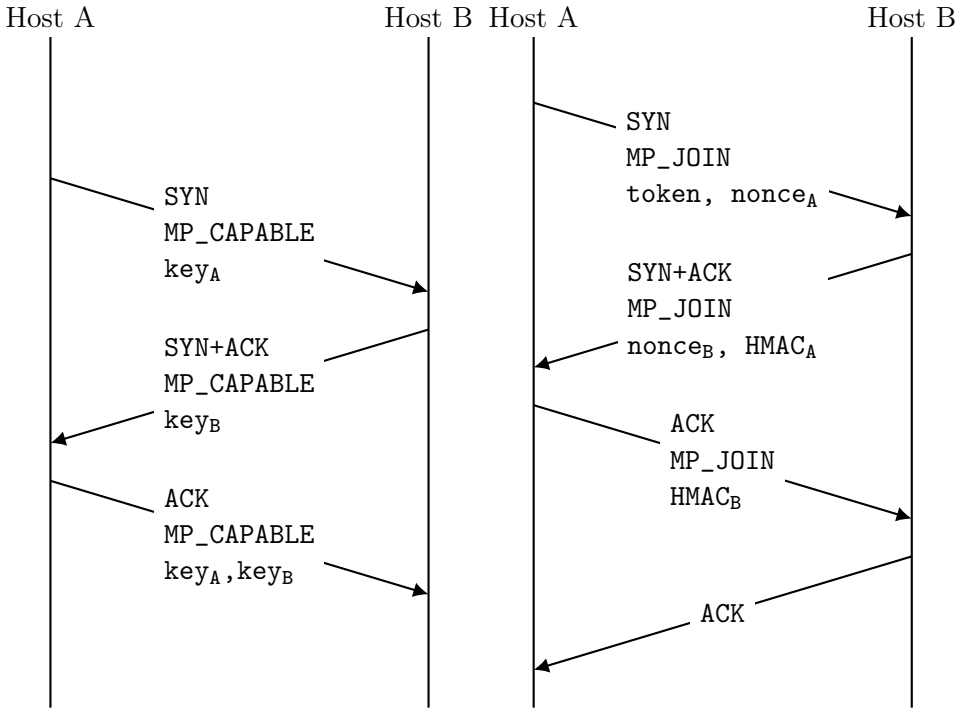


Figure 2.1: MPTCP (left) and TCP (right) in the networking stack.

2.2.1 Connection establishment

Like TCP, MPTCP uses a three-way handshake to establish a new connection. It adds new TCP options in the SYN segments to determine whether MPTCP can be used and to identify the connection [44]. The exchange is illustrated in figure 2.2a. The initiating host A determines whether the receiver B is MPTCP capable or not with the `MP_CAPABLE` field in the TCP options. The first `SYN` contains the `MP_CAPABLE` option and a random key. B replies with a `SYN+ACK` also containing the `MP_CAPABLE` option and another random key. Host A returns an `ACK`, including `MP_CAPABLE` and both keys, establishing the connection.



(a) Connection setup. *(b) Subflow addition.*

Figure 2.2: MPTCP handshakes.

One MPTCP connection can hold a variable amount of subflows. They are added and removed dynamically [8]. Figure 2.2b shows the handshake. When a subflow is added to an existing connection, a new TCP connection is established and tied to an existing MPTCP session. To unambiguously identify that session, a token is needed. The token is a truncated hash of the keys exchanged during the initial handshake. During the subflow handshake, the `MP_JOIN` option contains the token to make the association. The

hosts need to verify that the new subflow leads to the claimed host. Therefore, the hosts exchange nonces. Together with the previously exchanged keys, they compute a HMAC and authenticate each other. This makes subflow establishment a four-way handshake [39, 41, 44].

2.2.2 Data transmission

The subflows of a connection share a single send and receive buffer at the MPTCP level [44]. At the sender, the scheduler decides how to distribute the data on the subflows. At the receiver, reordering will occur more frequently due to segments arriving on several interfaces. To cope with these changes, the concepts of sequence numbers, acknowledgements, and the receive window have to be revisited.

Reliable and ordered transfer

MPTCP uses two sequence number spaces: one per-subflow sequence number and one data sequence number (DSN). The first ensures TCP compatibility in the network and its middleboxes and makes packet loss apparent. The second allows reordering before pushing the data to the application. The global receive window advertises the maximum DSN that can be received on any subflow [8, 41, 45]. Since regular acknowledgements refer only to the subflow sequence number, MPTCP adds explicit data acknowledgements [45].

Because of middleboxes modifying the TCP option field, the subflow sequence numbers need to be mapped to the DSN. This is the data sequence mapping (DSM). MPTCP uses a data sequence signal (`DSS`) field for the DSN and the DSM. For more information about its functioning and the additional optional checksum, refer to [44].

The receive buffer

Upon data reception, MPTCP performs reordering in two steps. Data is first reordered at the subflow level with the subflow sequence numbers. It is then reordered at the connection level using the DSN, before being pushed to the application [4].

For TCP, the size of the receive buffer is dynamically set to twice the bandwidth delay product (BDP). This accommodates to reordering and fast retransmission [4]. For MPTCP, where data comes over different paths, the strategy is different. The goal is to keep all paths fully utilised. Considering delay differences on the paths, connection-level reordering, and losses, the authors in [4] and [44] design the receive buffer to be $2 \times \sum_i^n \text{throughput}_i \times RTT_{\max}$ where n is the number of subflows and $RTT_{\max}$ the maximum round-trip time across all subflows.

2.2.3 Path management and scheduling

The path manager at each host decides how to add subflows to the connection. Four are available [40]:

- *default* does not establish additional subflows but accepts attempts from the other host.
- *fullmesh* (used by default) at the initiating host creates a full-mesh between all addresses from the host and all those advertised by the other host. Several subflows between pairs of addresses are possible, the limit depends on the configuration.
- *ndiffports* at the initiating host creates multiple subflows between one pair of addresses, using different ports. The amount of subflows is configurable.

- *binder*[1]

The scheduler at the sending host decides how to distribute the data over the different subflows. Poor scheduling is at risk of introducing head-of-line (HOL) blocking and receive-window limitation. With HOL blocking, the delivery of the data from the receive buffer to the application is delayed because of a missing segment. Upon arrival, the data can be pushed; this introduces burstiness in the transmission stream. The receive buffer delimits the amount of data that can be received out-of-order.

Additional mechanisms to improve scheduling are presented in [42].

2.2.4 Teardown

During a session, a subflow can be released using a regular `FIN` handshake, or a `RST`. For connection teardown, a host sends the TCP option `DATA FIN` inside the `DSS`. This is a connection-level signal informing the receiver that there is no more data. The sender can wait for a `DATA ACK` before sending a `FIN` on all subflows, or send a `DATA FIN` followed by a `FIN` on all subflows. [44]

2.2.5 Fallback to TCP

MPTCP must not prevent connectivity. Therefore, it can remove unusable subflows or transparently fall back to TCP whenever it is not able to carry on. This happens when the `MP_CAPABLE` flag is not exchanged properly during the connection setup, when it is removed from a data segment, when `DSS` options are missing, or when the `DSS` checksum does not go through [44].

2.2.6 Congestion control

MPTCP uses coupled congestion control schemes. It measures the congestion of each subflow and tries to realise a trade-off in load-balancing, stability, and efficiency, all the while ensuring fairness. By default, it uses all subflows and keeps traffic on them. During the increase phase, congestion balancing is implemented by moving more traffic to the less congested paths. To provide fairness, the aggregate congestion window size is limited to that of a regular TCP connection and cannot grow faster [41, 45].

The currently available congestion control schemes are [9, 40]:

- Linked Increase Algorithm (*LIA*)
- Opportunistic Linked Increase Algorithm (*OLIA*)
  OLIA is an adaptation of TCP New Reno for multiple paths [9]. It offers a better tradeoff between resource pooling and responsiveness than LIA and outperforms LIA when confronted with asymmetric paths [27].
- Delay-based Congestion Control for MPTCP (*wVegas*)
- Balanced Linked Adaptation Congestion Control Algorithm (*BALIA*)

[1] Not used, more details in: Luca Boccassi, Marwan M Fayed, and Mahesh K Marina. "Binder: a system to aggregate multiple internet gateways in community networks". In: *Proceedings of the 2013 ACM MobiCom workshop on Lowest cost denominator networking for universal access*. ACM. 2013, pp. 3–8.

2.3 Use cases

The official website for MPTCP[2] lists where it is currently used. In [10], the authors discuss recent use cases and operational experience. One notable deployment is the use of MPTCP for Apple's Siri, a personal assistant programme [10, 42].

To the best of my knowledge, there is no practical usage of MPTCP with adaptive streaming. As of the end of 2016, [25] contains a first analysis of the combination of DASH and MPTCP based on QoE. The aim is "to answer the questions whether MPTCP would always benefit mobile video streaming in terms of user experience, and ultimately, are there drawbacks and negative impact in some cases". The authors create several scenarios simulating a mobile device streaming a video over one WiFi and one cellular LTE connection. In their setup, they measure how well MPTCP uses additional bandwidth. Their conclusions are reported below in section 2.3.4. Besides this article, there are performance analyses for general HTTP download and media delivery that contain hints as to how the interaction could perform.

2.3.1 HTTP download

The authors of [44] assessed the performance of MPTCP for HTTP downloads without persistent connections. They connected a client to a server with two gigabit links and compared the amount of requests served per second. MPTCP outperforms TCP for files larger than 30 KB. Below that threshold, the connection overhead is too high. The transmission time is too small to allow the flows to exit the slow-start phase.

2.3.2 Benefits for media delivery

In [45], the authors tackle a media streaming optimisation with mobile WiFi and 3G. The WiFi connection highly fluctuates as the device moves, offering a high, but bursty, average throughput. The 3G connection is more stable but offers a lower average throughput. An MPTCP-aware player could schedule more downloads when the WiFi connection is good, filling the buffer as much as possible, and only resort to 3G when the buffer level gets too low. This strategy relies on the assumption that the bitrate is lower than the average full capacity.[3]

As stated in [42], MPTCP's advantages over TCP – increasing goodput and allowing vertical handover – are greater with long-lasting flows. Media streaming applications with rate-limited traffic benefit if a single connection is not sufficient to provide bandwidth or reliability. The authors point out that "not only the throughput but also the end-to-end delays, as well as buffer space requirements, become more relevant as these kind of applications are sensitive to delay-jitter."

2.3.3 Streaming issues

In [57], the authors outline three characteristics of MPTCP which are not desirable for live video streaming. They focus on heterogeneous wireless networks.

1. MPTCP, like TCP, retransmits data to achieve reliability when a packet is lost or an ACK delayed. However, since there is a timing constraint on video streaming, late data might not be useful.

[2] https://multipath-tcp.org/pmwiki.php/Main/HomePage
[3] In [45], the discussed scenario does not explicitly address bitrate adaptation.

2. MPTCP's congestion control uses additive increase/multiplicative decrease (AIMD), which results in throughput fluctuations and increased end-to-end delay.

3. "[Path asymmetry] may degrade the overall streaming quality", because two paths with very different delays may worsen the performance of MPTCP as compared to TCP.

They propose quAlity-Driven MultIpath TCP (ADMIT), a "quality-driven optimisation framework [for] the decision process of multipath data transfer". It is a modified version of MPTCP designed for high-quality video streaming.

In [14], the authors identify two reasons why MPTCP has not been broadly deployed by multimedia content providers. The first is the degradation in performance with heterogeneous paths. The second is the increased HOL blocking, compared to TCP, resulting in "short but abrupt throughput drops". With HAS, these two phenomena make the network even less predictable and add complexity to the selection algorithm. The authors introduce a cross-layer scheduler for the streaming server. Their objective is to maximise the video data received in time at the player by making MPTCP video-aware. The scheduler has access to application and transport layer information, as well as to feedback from the client. It knows the dependencies between the video units, the state of the network (smoothed RTT, global moving average bandwidth, loss probability per path), and the status of the video player. It estimates the deadline per video unit and decides which ones to send based on their importance for playback and the probability that they will be received in time. The solution is targeted towards progressive download, that is, without dynamic quality adaptation. However, it can still be used with HAS if used per requested video segment. The implementation does not alter MPTCP. Instead, it adds the new scheduler at the application layer. It also needs to be used with a client able to send the feedback.

2.3.4 Future work

The currently available results raise the question whether MPTCP can be beneficial for DASH. On the one hand, MPTCP can increase throughput with HTTP if the files are large enough and the connections persistent. With multiple symmetric paths, or when using multiple paths only for backup, the overall video quality and stability can be improved. On the other hand, many issues are still to be solved when the paths are heterogenous. Some (partial) solutions have been proposed and open different approaches. Except for the radical one, modifying MPTCP for video streaming, which undermines the goal of DASH to use existing and widely deployed protocols, these can be grouped as follows:

- Using adaptation algorithms compatible with MPTCP. For heuristics relying on a bandwidth estimation, this means lifting the hypothesis that all video segments traverse the same path.

- Using adaptation algorithms that do not try to infer the state of the network, such as the buffer based ones presented in section 1.4.

- Making applications (players or servers) MPTCP-aware.

The two first approaches still require optimisations inside MPTCP to stabilise the throughput and minimise the delay-jitter. The last one pushes the responsibility for network utilisation to the application through a new API, such as presented in [21].

The authors of [25], examining precisely whether MPTCP benefits DASH over asymmetric paths, find that depending on the network, using MPTCP can improve and degrade the streaming experience. More specifically, they draw four conclusions:

1. The available bandwidth on the secondary link is the determining factor, together with the available video bitrates. The aggregate bandwidth should be sufficient to stream at a higher bitrate than the primary link can sustain on its own, wasting the added resources of the secondary link otherwise.
2. MPTCP improves streaming when the bandwidth on the primary link is highly variable, because it stabilises the average video bitrate with the secondary link and reduces switching. This effect is strengthened if the secondary link is stable.
3. MPTCP deteriorates streaming when the primary link is stable and the secondary one variable because it adds more switching.
4. When both links have large and stable available bandwidths, MPTCP achieves good resource pooling.

3 Testing infrastructure

This chapter describes the infrastructure used as a controlled environment for the tests. After a brief overview, the different machines, settings, and tools are presented. The simulated network is then outlined, before the test parameters are reviewed. Finally, the main scripts interacting with the infrastructure are introduced. Throughout the chapter, several design choices are explained. Appendix A lists the git repositories where all code used to build the infrastructure and interact with it can be found.

3.1 Overview

The testing environment simulates a server and a client machine connected through two independent and configurable links. The setup is composed of one physical and two virtual machines (VM) inside it, created and configured with VirtualBox. [1] Their configuration is given in appendix D. The host is running Fedora 23.[2] It has 4 CPUs and 4 GB of RAM. The VMs are running Ubuntu Trusty (14.04)[3] with the modified MPTCP kernel 3.18.20-90-mptcp [40], installed via the repository. They each have 2 CPUs and 1 GB of RAM. The VMs have 3 network interfaces each: 2 for the links and 1 used to control them via the host. A high-level representation of the setup is given in figure 3.1.

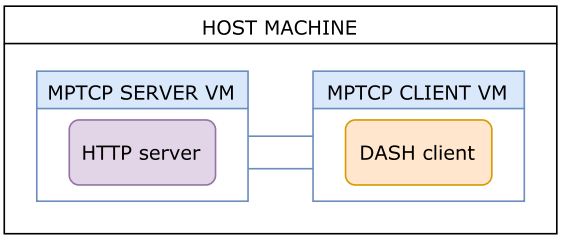


Figure 3.1: High-level view on the infrastructure: the server and client VMs are on the same host machine and connected by two disjoint virtual paths.

3.2 Streaming server

The server VM stores the video files and makes them available for streaming. It runs Apache HTTP Server Version 2.4.[4] The `KeepAliveTimeout` has been changed from the default 5 to 300 s, so that the server is less likely to close persistent HTTP connections while waiting for new requests.

[1] VirtualBox: https://www.virtualbox.org
[2] Fedora OS: https://getfedora.org
[3] Ubuntu OS: http://www.ubuntu.com
[4] Apache HTTP Server Project: https://httpd.apache.org/

Video dataset

The video used for the measurements is *Big Buck Bunny* [6], a computer animated short film which has a total duration of 9 minutes and 56 seconds. The encoded version belongs to the dataset from [29]. H.264 is used as video codec and MP4 as media encapsulation format. For each video, two MPD files are available. The "simple" version references separate segments, whereas the "on-demand" version refers to an unsegmented file for byte-range requests. Only the first has been used during the tests. The dataset provides:

- 6 different segment lengths: 1, 2, 4, 6, 10, 15 seconds;
- 20 representations with a bitrate between 45 kbps and 4.7 Mbps;
- 5 screen sizes: 320x240, 480x360, 854x480, 1280x720, 1920x1080 pixels.

It should be noted that the bitrates are defined by the screen sizes and the segment length. The 6 highest bitrates have the largest screen size; the smaller the segments, the higher the bitrate.

The segment length and the range of bitrates are the two key parameters for an adaptive streaming dataset [29]. They determine how often an adaptation can be performed and how fine-grained it can be [48]. However, both short segments and multiple bitrates come at a price.

Shorter segments allow more adaptations throughout the video. But the shorter the segment, the larger the relative overhead to get it, since it requires an entire HTTP round-trip. Müller et al. evaluate the incurred overhead for different bitrates in [37]. The lower bound for the overhead caused by TCP, IPv4, and Ethernet is $20 + 20 + 14 = 54$ B. If we consider the smallest segment in the dataset, i.e. segment 462, encoded at 47 kbps, with 1 s segmentation, and a size of 800 B, this represents 6.75 %. This does not yet include the HTTP overhead, nor connection establishment and reliable transfer mechanisms. For an entire playback session, the overhead decreases exponentially with the video bitrate [37]. 2 s segments at 100 kbps give an overhead of 12 % with HTTP/1.0 and 11.5 % with HTTP/1.1; 4.5 Mbps require around 5 % for both version of the protocol. Shorter segments also suffer from lower coding efficiency [48], making it necessary to send more data in total.

With more available representations, an adaptation logic can hope to find a bitrate close to the computed optimum instead of settling for a closest match that might be far from it. In terms of resources, the video needs to be encoded more times and the server needs to store more files. Many available representations can result in more quality switches during the playback if the adaptation logic does not smooth the measured bandwidth enough, especially if the segments are short. This is bad from a QoE point of view. Thus, more choice offers better adaptation possibilites but adds complexity to the decision algorithm.

Segments of 2 to 3 s have given the best results for regular TCP with persistent connections in [29]. For non persistent connections, the best range is 5 to 8 s. In their tests, the authors used the same dataset and compared the average media bitrate obtained with varying segment sizes and fluctuating bandwidth. Their test network was configured with a constant added delay of 150 ms.

3.3 DASH player

The client VM is responsible for playing the videos streamed from the server. The player used for the experiment is the open-source VLC media player[5] currently under development at version 3.0.0.

3.3.1 Evolution of the DASH plugin

VLC includes a DASH plugin. The earliest version of it is described in [36]. Since VLC version 2.0, the plugin is fully integrated into VLC. At the time of the publication in 2011, the authors stated that "the plugin opens a HTTP connection for each individual segment. This will be improved in a future version of the plugin that will use persistent HTTP connections to reduce the HTTP overhead." ([36]) Although the change to persistent connections appears to have been made (the researchers' blog mentions it [34]), it was subsequently reverted in February 2016 on the git master branch of the VLC project.[6] According to one of the VLC developers, the new backend does not support pipelining well and that is probably the reason why persistent connections were deactivated.

The use of HTTP/1.1 with persistent connections makes more sense to analyse the behaviour of MPTCP. The additional MPTCP subflow establishment overhead needs to be outweighed by the benefit of several paths [44]. Moreover with regular TCP, persistent connections add less overhead, as mentioned in section 3.2. In [29] and [16], the authors confirm that persistent connections with DASH result in better bandwidth usage. All test results mentioned in the remainder of this thesis have been obtained trying to use persistent connections by reverting the commit, unless mentioned otherwise. Unfortunately, the connections are not as persistent as expected.

Since its first implementation, the plugin has been decoupled into a `dash` and an `adaptive` part, to allow the reuse of the latter for other adaptive streaming standards. Apple Live Streaming and Microsoft Smooth Streaming are currently supported.

3.3.2 Adaptation algorithms

As of October 2016, two adaptation algorithms are implemented in VLC: a bandwidth adaptive and a buffer predictive logic. The latter one is more recent and has been chosen as default. Their respective algorithms are explained in the following.

Initialisation When the URL of a DASH streaming video is added to the VLC playlist, the `PlaylistManager` creates a new `AbstractAdaptationLogic` for it. Child classes have to implement the method `updateDownloadRate`, where the bandwidth estimation is computed. In the case of a rate based adaptation, the concrete object will be a `RateBasedAdaptationLogic`. It gets the preferred screen size (height and width) as input. For the predictive logic, a `PredictiveAdaptationLogic` is instantiated.

Vertical horizontal filter To understand how both algorithms work, let us first introduce the vertical horizontal filter (VHF) technique for bandwidth estimation, presented in [20]. Originally used in the financial world, VHF is a modified exponentially weighted moving average (EWMA). The goal is to obtain a stable estimation with low noise impact.

[5] VideoLAN's VLC: https://www.videolan.org/vlc/

[6] https://github.com/videolan/vlc, commit 874a409 closes the connection whenever there are no more bytes to be read in the response message.

An EWMA holds n observations O_k in its window to produce an estimation E_i with the formula

$$E_i = \alpha E_{i-1} + (1 - \alpha) O_i \tag{3.1}$$

A strong smoothing factor α makes the estimation stable, while a smaller α offers faster reactivity. The decision between the two properties can be dynamic: α becomes α_i, a function of the current circumstances, so that "sharp and non-persistent changes can at first be treated as noise using lower weights α_i. However, if the change persists, the filter should quickly converge to the new value" ([20]). Equation (3.1) becomes

$$E_i = \underbrace{\alpha_i E_{i-1}}_{\text{part 1}} + \underbrace{(1 - \alpha_i) O_i}_{\text{part 2}} \tag{3.2}$$

Equation (3.3) computes α_i in three steps with two factors $\Delta_{\max}$ and SD generated from the observation window. For VHF as used inside VLC, $\beta = 0.33$ and the window size $n = 10$.

$$\Delta_{\max_i} = O_{\max_i} - O_{\min_i} \tag{3.3a}$$

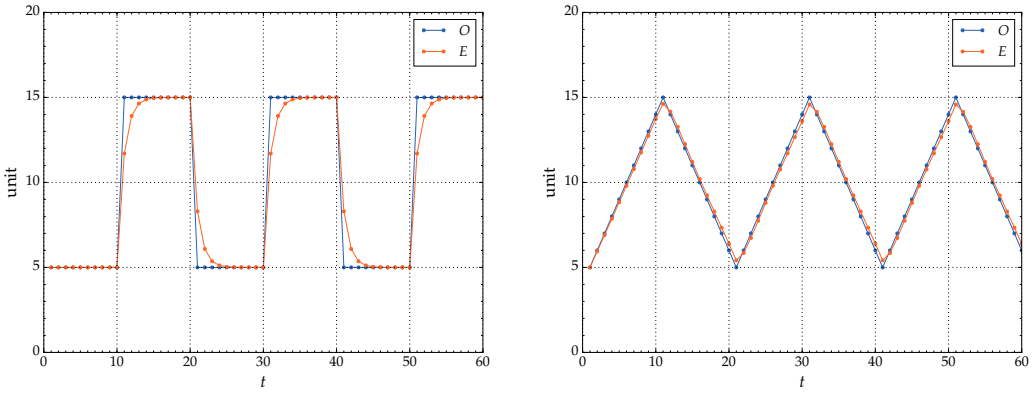
$$SD_i = \sum_{t=i-n}^{i} |O_t - O_{t-1}| \tag{3.3b}$$

$$\alpha_i = \begin{cases} \beta \frac{\Delta_{\max_i}}{SD_i} & \text{if } SD_i > 0 \\ 0.5 & \text{if } SD_i = 0 \end{cases} \tag{3.3c}$$

To better understand how the VHF is computed with different variations in the observations, let us define several profiles. They all get 60 observations with values ranging between 5 and 15.

Stable observations: When O is constant, linearly increasing, or linearly decreasing, E closely follows O. The α parameter converges to 0.5, 0.3, or 0.3 respectively. Due to part 2 of equation (3.2), this means that the current observations are still decisive for the current estimations, but since the variations are small, it does not qualify as noise.
When O is subject to large persistent variations, the VHF quickly converges to the new values. This is visible in figure 3.2. With a sharp change, as in figure 3.2a, α decreases and gives the new observation more weight than previously. This makes the VHF react quickly, but it is not sure yet that the observation is to be a persistent change. Figure 3.2b depicts more progressive changes. α oscillates between 0.15 and 0.3, peaking when the slope has stayed constant for 10 observations; α is minimal when the observation window holds the spikes, giving part 2 more weight once again. The VHF slightly smoothes the edges.

Noise: Quick convergence is desired when the changes are persistent. This should not prevent noise from being smoothed out. Figure 3.3 shows different types of non persistent variations in the observations. At $t = 21$, the value suddenly increases from 10 to 15. $E(t = 21)$ follows with a value of 13.35; α decreases from 0.5 to 0.33. At $t = 28$, O drops to 5 after a constant section at 10. This time, α has further decreased to 0.22. This results in $E(t = 28)$ following with a value of 6.1 because part 2 has much more weight than part 1. With more noise, the VHF trusts the previous estimation less and gives the new noisy observations more weight. This is also apparent between $t = 40$ and $t = 60$, with narrower variations. When O first slightly rises from 10 to 11, E follows with a value



(a) O varies between constant levels. *(b) O linearly increases and decreases.*

Figure 3.2: VHF with persistent variations.

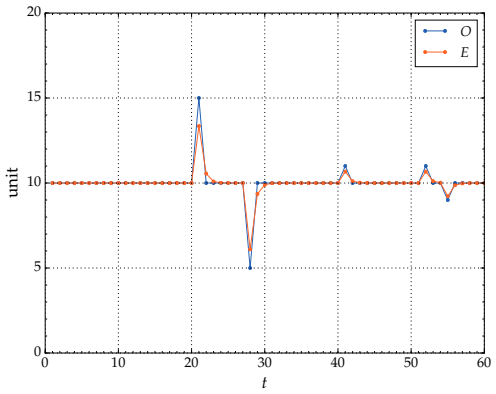


Figure 3.3: VHF with non persistent variations.

of 10.67. At $t = 55$, O drops to 9. This time E reacts with a drop to 9.22, because of the previous noise.

The performance of this VHF for network bandwidth estimation (and prediction) remains subject to experimental analysis. It depends on the profile of the network and the accuracy of the measurements. The measurements, in turn, partly depend on the length and schedule of observation. Nonetheless this theoretical assessment suggests that when faced with noisy observations and large variations, the estimation is very reactive to new values.

Bandwidth adaptive logic

The algorithm decides which bitrate to request for the next segment based on two variables: the preferred screen size and the currently estimated available bandwidth.

Bandwidth estimation Each time an HTTP message is read in and buffered, the `RateBasedAdaptationLogic` updates its rate estimation. Listing 1 shows the associated pseudo-code. It uses the size of the received content and the time it took to download, read, and buffer it. The size and time are added to the size and time of the current observation and used to compute a bandwidth from it.[7] On line 13, this observation is pushed to the end of the list `average`, popping the one at the front. This triggers

[7] If the time is less than 0.25 s, nothing is computed until it is reached.

a VHF computation and the result is returned to `bpsAvg`. To add further robustness, this estimation is multiplied by 3/4. The obtained `currentBps` is used for the bitrate selection.

Listing 1: Bandwidth adaptive logic (pseudo-code).

```
void RateBasedAdaptationLogic::updateDownloadRate(size_t size, mtime_t time)
{
    /* Accumulate up to the observation window (min. 0.25 seconds) */
    dllength += time;    // download time (microseconds)
    dlsize += size;      // download size (bytes)
    if (dllength < CLOCK_FREQ / 4)
        return;

    /* Current observation (bits per second) */
    size_t bps = CLOCK_FREQ * dlsize * 8 / dllength;

    /* Compute new bandwidth estimation (bits per second) */
    bpsAvg = average.push(bps)
    currentBps = bpsAvg * 3/4;

    /* Reset for the next observation */
    dlsize = dllength = 0;
}
```

Representation selection When a new video segment is requested from the server, a representation needs to be chosen for it. The logic takes the last computed `currentBps` as a threshold. It filters the adaptation set to find the representations respecting the preferred height and width. If any are found, it looks for the representation with the highest bitrate below `currentBps` inside that subset. If no representation with a matching resolution is found, it again looks for the representation with the highest bitrate below `currentBps`, but inside the whole adaptation set. At startup `currentBps` is considered to be 0 and the logic chooses the lowest bitrate matching the resolution preferences.

Predictive logic

The buffer predictive algorithm decides which bitrate to request next based on the buffer level, the last bitrate, and a prediction of the available bandwidth. It is based on the work in [60]. The authors designed an algorithm that takes advantage of a bandwidth prediction within a horizon of several seconds. They showed that prediction combined with consideration for buffer occupancy outperforms FESTIVE [26] and BBA [22], presented in section 1.4. The access to accurate prediction has not yet been solved, although proposals are outlined for cellular networks, where bandwidth variability is particularly challenging.

The algorithm implemented inside VLC is a hybrid form of it, for two reasons. Firstly, it accounts for media with multiple streams (video, audio, etc). Multiple buffers with potentially different occupancies are used. Secondly, the bandwidth prediction is replaced by a VHF estimation over the last 10 bandwidth observations, as previously. It is updated on each download.

Representation selection Each stream for which segments can be picked inside an adaptation set has its own associated statistics. They are made of the segment count, the buffering level B_{current}, the maximum buffering level B_{max}, the VHF estimation of the bandwidth, and the last segment duration (for variable segment sizes). The relative current buffering level is $B = B_{\text{current}}/B_{\text{max}}$. The buffers are divided into three zones based on occupancy: risky, transient, and safe. Two thresholds, B_{risky} and B_{safe}, delimit them [60]. Their values are 50 % and 80 % of B_{max}, respectively.

When choosing the next bitrate R_{next_S} for any stream S, its statistics and those of all other streams are retrieved to prevent allocating used resources. The remaining bitrate R_{rest_S} is computed as the difference between the highest bandwidth estimation of all streams and the bandwidth used by other streams. The previous bitrate of S is R_{prev_S}. The minimum relative current buffer B_{min} among all streams is used to decide on the zone for the computation to follow.

B_{min} **in safe zone:** $R_{\text{next}_S} = \max(R_{\text{rest}_S}, R_{\text{prev}_S})$. The quality either switches up or stays the same.

B_{min} **in transient zone:** $R_{\text{next}_S} = R_{\text{prev}_S}$. The quality stays the same.

B_{min} **in risky zone:** If there are two segments in the buffer, R_{next_S} is lowered by one representation with respect to R_{prev_S}. Else, the bitrate is decreased proportionally to the buffer level: $R_{\text{next}_S} = R_{\text{rest}_S} \times B_{\text{min}}$. In both cases, the quality switches down (if possible).

3.4 Simulated network

In order to test different network topologies, network emulation is necessary. The entire setup (excluding the routing tables) is done on the host machine.

3.4.1 Two simulated links

The host has four virtual link layer devices (TAP): `tapc1`, `taps1`, `tapc2`, and `taps2`. They are bridged together pairwise with the bridges `vbr1` and `vrb2` to simulate two links. Each VM is connected to two of those interfaces:[8] the client to `tapc1` and `tapc2`, and the server to `taps1` and `taps2`. The addresses are configured statically. The whole setup simulates a client and a server connected via two LANs and is shown in figure 3.4.

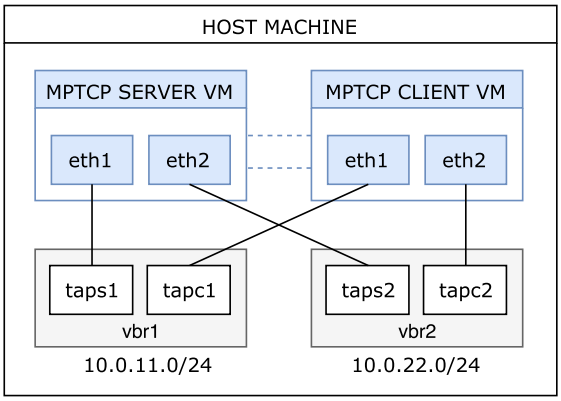


Figure 3.4: Network simulation setup.

[8] VirtualBox offers different types of networking. Here, bridged networking is used: https://www.virtualbox.org/manual/ch06.html#network_bridged.

To have two separate paths, no traffic between `vbr1` and `vbr2` is permitted. This is enforced with iptables in the forwarding chain of the filter table.

3.4.2 Linux Traffic Control

To simulate different network conditions, Linux Traffic Control (tc) (packaged with iproute2) [52] and the Network Emulator (NetEm) [53] are used. They allow direct control over the kernel packet scheduler. In order to control the bandwidth and to introduce delay, queuing disciplines (qdiscs) are added to all four TAP interfaces on the host. For all tests, bandwidth shaping is done symmetrically on a link: the same rate is configured for up- and download. Added delay is introduced in the same way, on both ends.

tc was not designed to be used as a network emulation tool. Although it is widely used as such, correctly setting up a simulated network with chained qdiscs is not trivial. To keep the design as simple as possible, a single NetEm qdisc is used on each interface. It allows to introduce delay and a rate limit. In order not to introduce buffer bloat, the length of the queue needs to be limited. It is set to twice the bandwidth-delay product (BDP) with a burst of one packet, dynamically computed from the desired rate and the RTT delay introduced on the link. With NetEm, the "limit" option determines the amount of packets emulation is applied on. The formulas used to compute the burst b, the queue q, and the limit l on each interface are given in equations 3.4, where r is the bandwidth on the link and d the RTT delay added on the link. On the host machine, the maximum transmission unit (MTU) on the considered interfaces is 1500 B and the kernel timer has a configured frequency (f) of 1000 Hz.

$$b = \max\left(MTU, \left\lceil \frac{r}{f} \right\rceil\right) \tag{3.4a}$$

$$q = 2rd \tag{3.4b}$$

$$l = \left\lceil \frac{b+q}{MTU} \right\rceil \tag{3.4c}$$

3.4.3 Network performance

I used ping and iperf to check the behaviour and performance of the virtual network. The script used to perform the tests is `testnetwork.py`, with all combinations of RTT delays of 25, 50, 100 and 150 ms and bandwidths of 1, 10, 20, 50 and 100 Mbps. The detailed results are reported in appendix E.

ping The delay setup has been tested with ping. The standard deviation ranges between 0.031 to 0.46 ms. The error margin for all recorded average values is always less than 8 %, on average it is 1.5 %.

iperf The bandwidth limit has been tested with iperf. UDP is first used with a target rate equal to the desired bandwidth to check whether it is reached. Second, the target rate is raised to twice the set bandwidth to check that the limit is respected. Finally, TCP is used to assess whether the rate limit and the buffer make it possible for the scheduler and the congestion control (New Reno) to reach the limit. The results show that the bandwidth cap is enforced with both protocols and that both are able to transfer data. While UDP uses more than 94 % of the available bandwidth, TCP falls behind with around 71.2 % in the worst case.

3.5 Parameters

To compare the performance of DASH with TCP against MPTCP, a restricted set of parameters is used. Their chosen values are explained below.

MPTCP: When MPTCP is enabled, the default configuration for version 0.90 is kept. The congestion control is OLIA on the client and on the server. The path manager is *fullmesh*. Because of the iptables settings on the host, only two disjoint paths are created. The scheduler is left to *default*.

Link characteristics: The added delay on each link is set to a RTT ranging from 25 ms to 150 ms. MPTCP always gets two links with different delays to make the links asymmetric and more realistic. The bandwidth is varied as well. For TCP, section 4.1 explains how a sufficient value is chosen. Section 5.1 discusses how the delay and bandwidth on the links are set to create comparable test cases for TCP and MPTCP.

Adaptation logic: Both logics implemented by VLC, the bandwidth adaptive and the buffer predictive, are tested.

Videos: 2 s and 10 s segments are used because they were the most common for HAS services in 2015 [48, 56]. Appendix C gives the two MPD files that are used. Appendix B technically describes the dataset in terms of resolutions, bitrates, and segment sizes. When referring to *Big Buck Bunny* with segments of length of x seconds in the simple version, the shorthand BBBxs will be used throughout the remainder of the report.

3.6 Main scripts

To automate the playback of the video and the capture and analysis of the measurements, I wrote several Python and Bash scripts. To explain what they do, let us walk through a simple test during which the video will be played once in a specific test setup. The starting point is `do_test.py`. As input, it gets the video to play, which adaptation logic to use, the delay and rate to configure on each link, and whether to use MPTCP or not, which congestion control to set on the VMs.

Before playback The host starts by configuring the rates and delays on the links. It then enables or disables MPTCP and sets the congestion control algorithm on the client and the server. After that, it flushes the TCP metrics cache on both machines. Next, the host stores several parameters about its state. It instructs the server to do the same. Then the host loads the TCP Probe[9] on port 80 of the server, redirecting the output to file. This logs the congestion window (cwnd), the slow start threshold (ssthresh), the smoothed RTT (SRTT), and other TCP connection state variables for each packet the server sends on port 80. The server also launches tcpdump[10] to listen to all traffic exchanged with the client. Once the server is ready, the host tells the client to play the video. Before the client carries out that instruction, it stores its own state and launches tcpdump on both its interfaces connected with the server.

Playback The client runs an X virtual frame buffer (Xvfb)[11] and launches VLC with the desired video and adaptation logic on it. Xvfb is needed to run VLC on a headless

[9] TCP Probe: https://wiki.linuxfoundation.org/networking/tcpprobe
[10] tcpdump: http://www.tcpdump.org/
[11] Xvfb: https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml

server, because it needs a video and audio output. Simply suppressing them inside VLC is possible with the `dummy` option for `--vout` (video output) and `--aout` (audio output). However this does not correctly simulate playback, because the player does not have to produce any output, knowing it is suppressed, and can optimise it away. Instead, Xvfb requires the programme to perform all operations as if the output were used.

After playback Once VLC completes, the client stops tcpdump and records its state again. The host is notified and makes the server stop all processes and store its state. It then does the same and, finally, fetches all produced log files from the VMs.

Application-level analysis The first part of the analysis looks at what happened inside VLC. All recorded metrics, such as the behaviour of the adaptation logics, the buffer, the chosen representation bitrates, are first stored in a single file. They are dispatched into separate CSV files to ease further processing. For each single test, the evolution of all QoE measures can then be directly plotted. To make the performance comparable, a one-line QoE summary can be produced for each test. It includes the average, standard deviation, minimum, maximum, 25^{th}, 50^{th}, 75^{th} percentiles of:

- the video bitrate,
- the measured bandwidth,
- the switching amplitude,
- the buffer fill state,
- the segment performance (time of playback contained in the segment over the time it takes to get it from HTTP request to the end of the read by VLC).

It also holds:

- the estimated startup delay,
- the amount of up/down switches,
- the amount of *drops*/*raises*: defined as local minima/maxima on the curve of chosen representations, so that a progressive decrease/increase in quality has the same weight as a sudden one,
- the *success rate* of the maximum video bitrate: defined as the percentage of segments at maximum bitrate over all segments used during the session.

Network analysis The second part of the analysis inspects TCP and MPTCP events on both VMs. The evolution of the congestion window and the SRTT on each connection can be directly extracted from the file produced by TCP Probe on the server. The tcpdump traces are processed with dedicated MPTCP analysis scripts [17]. Per video session, they can produce a single file containing information on each connection, the amount of bytes exchanged, the amount of retransmissions, reinjections, and other such metrics.

The case of the single test is not very useful, except for testing that VLC and all other tools behave as expected. `do_test.py` is embedded in more complete test suites for the calibration discussed in chapter 4 and the comparison detailed in chapter 5.

4 | Bandwidth calibration with TCP

Before comparing TCP with MPTCP, the bandwidth and delay of the links need to be decided. For a certain delay, it is possible to find a sufficient bandwidth at which TCP allows streaming at a certain video bitrate. This bitrate will be the highest one available in the dataset. The RTT delays are 25, 50, 100, and 150 ms. For each one, I identify how much bandwidth VLC needs with TCP to play the video at its maximum available bitrate. The found bandwidths, together with the delays, form the baseline set used for comparison in chapter 5.

After explaining the methodology and presenting the results for an initial setup, I will discuss additional findings and the results with other setups.

4.1 Methodology

Two main QoE indicators are used to define the bandwidth as sufficient. They are used one after another, the second only being tested if the first is met.

1. The success rate of the maximum video bitrate.[1]
   I chose the success rate instead of the average media bitrate, discussed in section 1.3.2, because the goal is to stream at the highest available quality. The success rate gives a directly comparable hit or miss information, whereas the average bitrate reflects the stability as well, without indicating whether the maximum was ever reached.

2. The amount of quality drops.[1]
   I chose to minimize the "drops" instead of the stalls, identified as most important parameter in section 1.3.2. At such high bitrate levels, one could expect no stalls at all, and therefore minimising them is not necessary. Quality switches have an amplitude and a frequency, but the frequency was more often seen as decisive. Since the goal is to remain at maximum bitrate, only down switching is considered. The drops are defined as potentially progressive to make the calibration approach useful for any length of video segment: with shorter segments, a progressive decrease is more likely.

Inside a search space of candidate bandwidths, a binary search finds the calibration bandwidth as follows: the chosen bandwidth is the lowest one achieving a success rate of at least 95 % and which minimises the amount of drops. The threshold accounts for adaptation logics not necessarily starting at the highest bitrate. The associated pseudo-code is given in listing 2.

[1] Defined in section 3.6.

Listing 2: TCP bandwidth calibration

```
def calibrate_bandwidth(delay, min_bw, max_bw, n):
    """
    :param  delay:      RTT delay
    :param  min_bw:     lowest value of the bandwidth search space
    :param  max_bw:     highest value of the bandwidth search space
    :param  n:          size of the bandwidth search space
    :return chosen_bw, chosen_drops:
                        (chosen calibration bandwidth, amount of drops)
                    or (None, infinity)
    """
    # Bandwidth search space of n equally spaced values.
    bws = linspace(start=min_bw, stop=max_bw, num=n)

    # Binary search initialisation and return values.
    first = 0
    last = n - 1
    chosen_bw = None
    chosen_drops = float("infinity")

    # Binary search.
    while first <= last:
        middle = (first + last) // 2  # floor division
        current_bw = bws[middle]
        success_rate, current_drops = play_video(mptcp=0, delay=delay, bw=current_bw)
        if success_rate >= 0.95:
            if (chosen_bw is None) or \
               (current_bw < chosen_bw and current_drops <= chosen_drops):
                # new search space: [min;current[
                first = first
                last = middle - 1
            else:
                # new search space: ]current;max]
                first = current + 1
                last = last

    return chosen_bw, chosen_drops
```

I performed the calibration in several rounds. The first tested bandwidths started from the maximum video bitrate up to five times its value. When no suitable bandwidth was found for a given delay in the first round, a new round was launched with higher bandwidths and a wider range. For added robustness, each test has been performed twice. The resulting success rates and drop amounts discussed throughout this section are averages.

Starting point The initial setup is given in table 4.1.

Table 4.1: Initial calibration setup.

video	BBB2s (max. bitrate = 4.2 Mbps)
VLC logic	bandwidth adaptive
TCP congestion control	New Reno on both VMs

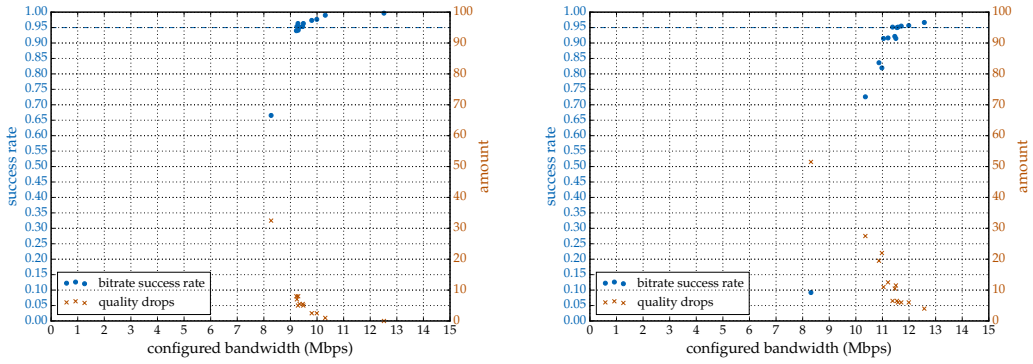
The characteristics of the successive search spaces are given in table 4.2.

Table 4.2: Bandwidth calibration search spaces for BBB2s.

Round	min. Mbps	max. Mbps	values
1st	4.2	21	100
2nd	21	50.4	100
3rd	50.4	105	100
4th	105	210	100

4.2 Short RTT delays with the initial setup

The calibration results for 25 ms and 50 ms RTT are depicted in figure 4.1. I fine-tuned the approximate sufficient bandwidth with an additional round between 9 Mbps and 13 Mbps.



(a) RTT = 25 ms *(b) RTT = 50 ms*

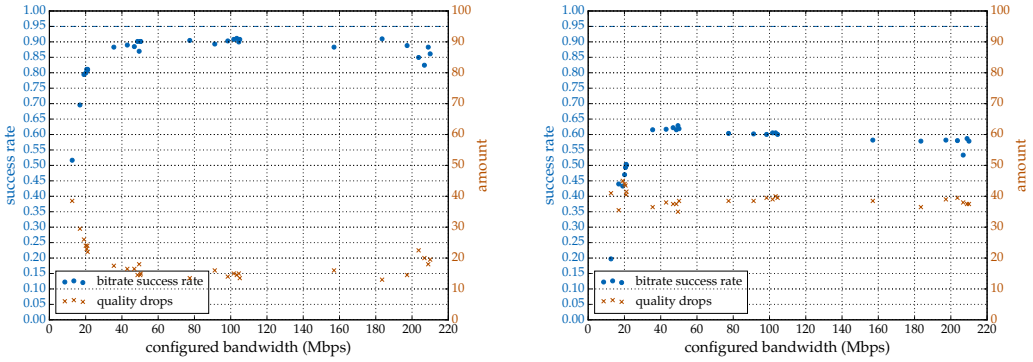
Figure 4.1: Bandwidth calibration results for short RTT delays with the initial setup. The blue dots show the achieved success rate (left y-axis) and the orange crosses the amount of drops (right y-axis) over the configured bandwidth (x-axis).

In both cases, the success rate increases and the amount of drops decreases as the configured bandwidth increases. This indicates that VLC correctly detects when there is more bandwidth and that the bandwidth adaptive algorithm effectively chooses the bitrates accordingly.

For 25 ms the success rate threshold is reached starting at 9.34 Mbps, but there are 5.5 drops. At 10 Mbps, the success rate reaches 98 % with only 2.5 drops. For 50 ms, 11.7 Mbps are needed to reach the threshold, with 6 drops. By comparing both, it appears that more bandwidth is needed as the delay increases to achieve the same result. This makes sense, since VLC approximates the available bandwidth as the measured data throughput from HTTP request to response, at the application level.

4.3 Long RTT delays with the initial setup

For RTTs of 100 ms and 150 ms none of the tested bandwidths between 4.2 Mbps and 210 Mbps yields the desired QoE result. Figure 4.2 shows the obtained results. The success rate reaches a peak around 50 Mbps and then stagnates at higher bandwidths.



(a) $RTT = 100\,\text{ms}$ *(b)* $RTT = 150\,\text{ms}$

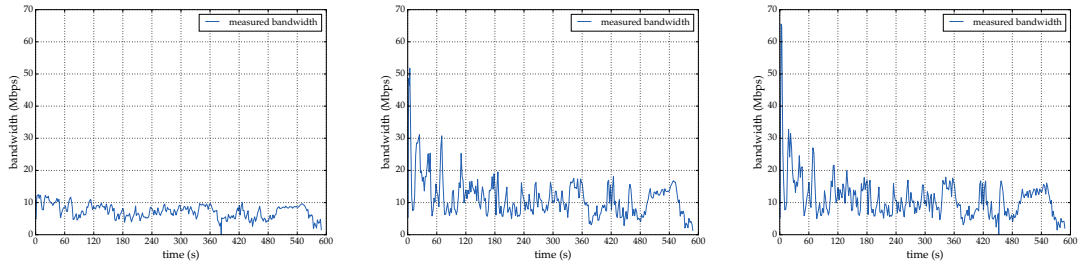
Figure 4.2: Bandwidth calibration results for long RTT delays with the initial setup. The blue dots show the achieved success rate (left y-axis) and the orange crosses the amount of drops (right y-axis) over the configured bandwidth (x-axis).

Above 50 Mbps, VLC is not able to exploit the available bandwidth. To better understand why, let us take three test instances to analyse: 20.08, 104.45, and 203.64 Mbps with 100 ms RTT delay. They achieved success rates of 76.25 %, 89.97 %, and 89.63 % and switched representations 69, 36, and 33 times, respectively. The first follows the calibration curve before its stagnation and can therefore be considered as behaving normally. The two others however belong to the unexpectedly not increasing second part of the curve; indeed they achieve similar results while their respective configured bandwidths are in a ratio of one to two. The bitrate selection depends on the bandwidth estimation and VLC seems not to estimate it correctly. I will take this as a starting point to find the root cause for the bad performance.

4.3.1 What happens inside VLC?

Inside VLC, the mean measured bandwidths and standard deviations for the three cases are: (7.31 ± 2.15) Mbps, (11.81 ± 6.97) Mbps, (11.34 ± 6.93) Mbps. Figure 4.3 shows their evolution over the video session. In the first case, the variations are spread over the entire session. In the latter two, the measured bandwidth is high at first, reaching a maximum of 51.80 Mbps and 65.51 Mbps in the first 5 seconds, before it plummets. VLC's bandwidth estimation computes the size of the payload over the download time. The distributions of payload size in the later two cases are very similar. This leaves the download time as a factor that could explain that the throughput stays the same, even though the configured bandwidth is doubled.

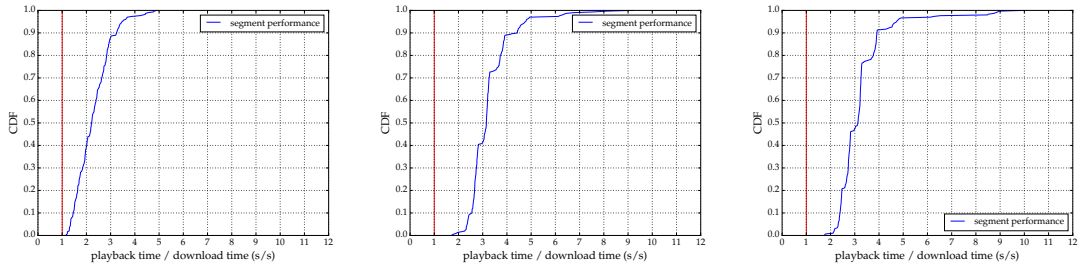
Let us turn towards the segment performance, shown in figure 4.4, which measures the time of video playback contained in a segment (here, 2 s) over the time it takes to get it from HTTP request to the end of the read by VLC. It should stay above 1 so that the segments do not take more time to download than to play, keeping the buffer non empty. For DASH, it should all the same not be too high, otherwise the player could and should switch up the quality (if possible). In the first case, this approximate rule is held: the mean segment performance is 2.29 (s/s), 3.3 (s/s) is the 90^{th} percentile. In the later



(a) 20.08 Mbps *(b)* 104.45 Mbps *(c)* 203.64 Mbps

Figure 4.3: Measured bandwidth over time for 100 ms *RTT delay with the initial setup.*

two cases, the mean segment performance is above 3 (s/s) and the 90$^{\text{th}}$ percentile around 4 (s/s). This contradicts the expectation that the measured bandwidth stems from a large end-to-end delay at the application.



(a) 20.08 Mbps *(b)* 104.45 Mbps *(c)* 203.64 Mbps

Figure 4.4: Segment performance CDF for 100 ms *RTT delay with the initial setup.*

4.3.2 What happens with TCP?

The behaviour of the application does not seem to be the problem. Let us look at the network in search for an explanation.

Many TCP connections with BBB2s

The first thing to notice is that instead of one long lasting TCP flow for the entire session, VLC establishes more than 200. The first only fetches the MPD. This connection stays alive but unused for 300 s or 600 s. The second lasts for around 3 min and transports around 95 video segments. After this long-lived connection is teared down, each HTTP request is transported over its own TCP connection. BBB2s is composed of 299 video segments. This sums up to a total of around 200 TCP connections between the client and the server. This amount increases with every quality switch after the first long-lived connection, because each switch requires an initialisation segment containing meta-data to be requested first.

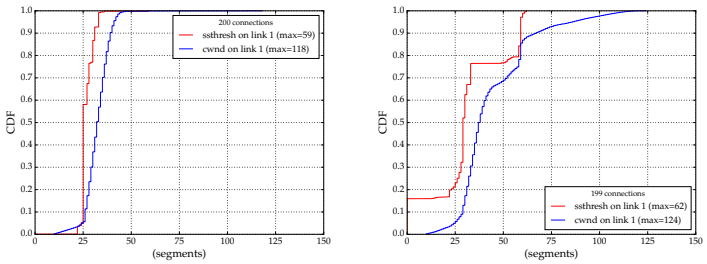
Analysing why VLC is not able to establish a persistent HTTP connection for BBB2s is beyond the scope of this thesis.

TCP congestion window growth

Linux' implementation of TCP contains two settings inside `/proc/sys/net/ipv4/` affecting the growth of the cwnd: `tcp_no_metrics_save` and `tcp_slow_start_after_idle`.

The first has an impact on several consecutive connections, whereas the second changes the behaviour of connections long enough to experience idle periods.

`tcp_no_metrics_save`: When the option is set to `0` (default) "(...) TCP saves various connection metrics in the route cache when the connection closes, so that connections established in the near future can use these to set initial conditions. Usually, this increases overall performance, but may sometimes cause performance degradation." ([54]) In this case, disabling the caching gives higher congestion windows for the short lived connections, but leeds to more variation in the bandwidth estimation and an overall degradation of the video streaming experience. Figure 4.5 shows comparative results for a video session using the same settings but where `tcp_no_metrics_save` is first disabled (figure 4.5a) and then enabled (figure 4.5b). In the first case, the success rate is 97.99 % and the mean video bitrate 4.21 Mbps; in the second 94.98 % and 4.16 Mbps. The total traffic from server to client (measured at the client) consists in 314.22 MB in the first case, and 313.38 MB in the second. The performance decrease can be explained by looking at the amount of TCP retransmissions: 79 in the first and 10 758 in the second.

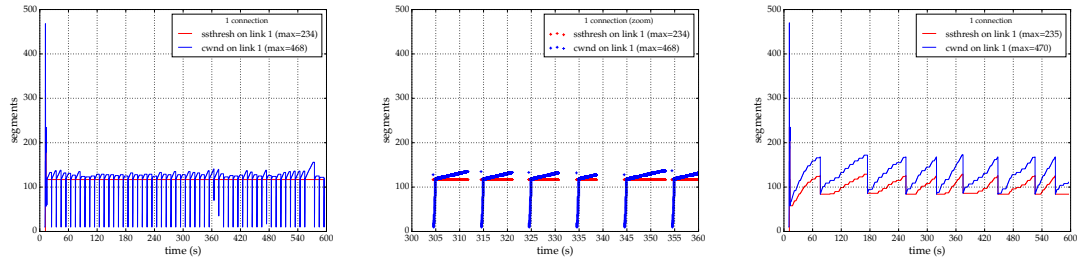


(a) `tcp_no_metrics_save=0` *(b)* `tcp_no_metrics_save=1`

Figure 4.5: cwnd and ssthresh CDF on the server with `tcp_no_metrics_save` *(25 ms RTT delay, 10 Mbps, BBB2s, New Reno).*

`tcp_slow_start_after_idle`: When the option is set to `1` (default), Linux will "time out the congestion window after an idle period. An idle period is defined at the current RTO." ([54]) Disabling the setting prevents the congestion window from being reset during idle periods. This makes the bandwidth estimation inside VLC higher, but comes with a negative side-effect: the measured bandwidth is more variable. Figure 4.6 compares the congestion window evolution for a video session with a persistent HTTP connection (with BBB10s) using the same settings but where `tcp_slow_start_after_idle` is first enabled (figure 4.6a) and then disabled (figure 4.6c). In this present example, the measured bandwidth (and standard deviation) are (7.08 ± 0.38) Mbps and (6.79 ± 0.30) Mbps in the first and second case, respectively. The first enters TCP slow start around every 10 s, followed by a short transition to congestion avoidance. This corresponds to the period at which VLC requests a new video segment, because the segment size is 10 s.[2] This "on-off" traffic pattern is characteristic for streaming applications [28] with a bounded buffer: during the "on" part, the client downloads video chunks, and during the "off" part it waits, rendering its connection to the server idle. Segments of 10 s are big enough to enter congestion avoidance (figure figure 4.6b), but with smaller segments, TCP stays in slow start; the bandwidth estimation is far below the actual network capacity.

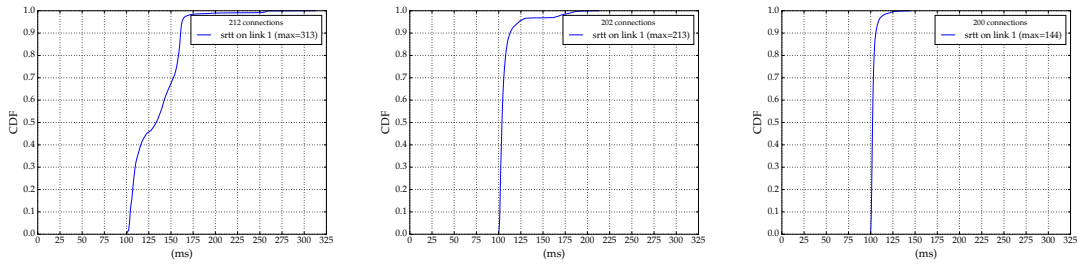
[2] In addition, VLC's buffer has a soft upper bound of 9 s, meaning that a new segment can be put in it whenever the buffer fill state is below that level.



(a) `tcp_slow_start_after_idle=1` *(b)* `tcp_slow_start_after_idle=1` *zoom* *(c)* `tcp_slow_start_after_idle=0`

Figure 4.6: cwnd and ssthresh evolution on the server with `tcp_slow_start_after_idle` *(*25 ms *RTT delay,* 10 Mbps*, BBB10s, New Reno).*

To definitely exclude bufferbloat, let us examine the SRTT measured at the server with TCP Probe, given at figure 4.7. In all three cases, the SRTT has a lower bound on the configured delay. It becomes more stable as the bandwidth increases. The packets do not sit in the buffer for an excessive amount of time. The infrastructure has an acceptable queue size limit.

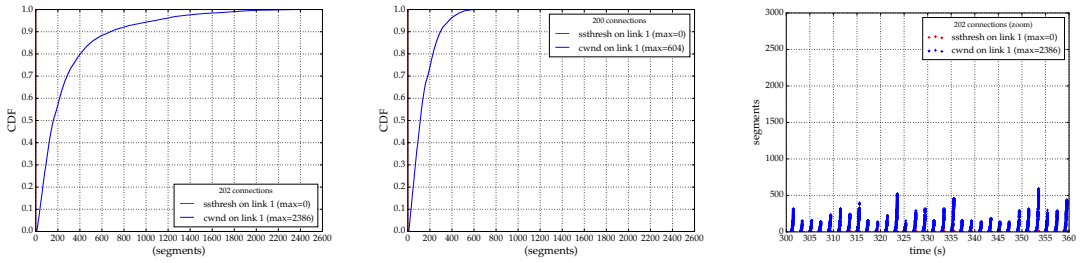


(a) 20.08 Mbps *(b)* 104.45 Mbps *(c)* 203.64 Mbps

Figure 4.7: SRTT CDF on the server for 100 ms *RTT delay with the initial setup.*

If neither the application nor the latency limit the throughput, let us have a look at the server's congestion window and its growth. With non persistent HTTP and New Reno as congestion control, it could be that it stays too small.

Figure 4.8a shows a CDF on the cwnd and ssthresh for the second instance. Figure 4.8b shows the same CDF, but without the two special connections mentioned earlier, to show their impact. Figure 4.8c zooms in on the evolution of the cwnd when the player uses multiple short connections. What happens during a session is the following: VLC opens a connection to request the MPD. The server sends it in two segments. TCP stays in slow start. The next connection is meant to transport the video segments and stays alive for approximately 3 min. Every 2 s, VLC requests a new video segment, which the server sends. Between segments, the connection is idle. Because of `tcp_slow_start_after_idle=1`, the congestion window is reset to 10. After this persistent connection, the remaining segments and initialisation files with sizes between 883 B and 2.2 MB are requested over individual connections. The data is not big enough to make the cwnd grow so that TCP never fully utilises the available bandwidth. As the request schedule is set by the application, another remedy is to use bigger segment sizes.



(a) cwnd and ssthresh CDF

(b) cwnd and ssthresh CDF without the MPD request and the 3 min *connection*

(c) cwnd and ssthresh zoom between 300 s *and* 360 s

Figure 4.8: cwnd and ssthresh on the server for 100 ms *RTT delay and* 104.45 Mbps *with the initial setup.*

4.4 Conclusions on the initial setup

The TCP calibration with the initial setup succeeds for short delays. It results in bandwidths more than twice as large as the maximum video bitrate, because VLC uses non persistent HTTP adding much overhead to the short segments. With long delays, the "on-off" traffic pattern and 2 s segments make it impossible to reach a sufficient throughput because the server's congestion window does not grow fast enough.

4.5 Further results with different parameters

To get a better idea of the influence of the different parameters, I repeated the calibration with different setups. Three parameters are successively changed:

1. 10 s *video segments*, as this is the other most widely used size;
2. *VLC's rate predictive adaptation algorithm*, as it is VLC's new default;
3. *CUBIC congestion control*, as this is Linux' default setting.

4.5.1 Segment sizes of 10 s

The first notable difference when using BBB10s instead of BBB2s is that VLC uses persistent HTTP, and thus establishes a single TCP connection, for the video transfer.[3]

The second difference is that the calibration results for short delays are much better. They are not directly comparable because BBB2s has a maximum available bitrate of 4.2 Mbps and BBB10s of 3.8 Mbps.

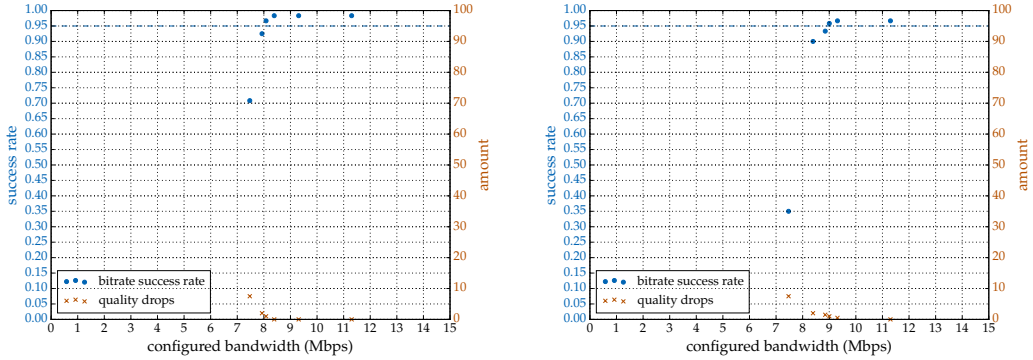
- At 25 ms, 7 Mbps are enough to reach the success rate threshold. This is a bandwidth/bitrate ratio of 184.58 %. With BBB2s, 9.3 Mbps were needed and the ratio was 220.38 %.
- At 50 ms, 7.5 Mbps are sufficient. This is a bandwidth/bitrate ratio of 197.76 %. With BBB2s, 11.7 Mbps were needed and the ratio was 277.26 %.

The better performance with BBB10s confirms the findings in [29], that "[the results] showed a significant performance drop by streaming from a Web server which does not allow persistent connection, especially for short segments lengths like 2 seconds or less", although "[s]egment sizes greater than 6 seconds are not influenced so much by the

[3] In addition to the MPD fetching connection.

connection settings of the Web server (...) due to the lower amount of segments and, thus, the smaller overhead produced by reconnects (...)."

The third difference is that the setup also yields sufficient bandwidths for long RTT delays. They are illustrated in figure 4.9. The authors of [29] also stated that "for TCP-based



(a) $RTT = 100\,\text{ms}$ *(b)* $RTT = 150\,\text{ms}$

Figure 4.9: Bandwidth calibration results for long RTT delays with the initial setup but video BBB10s. The blue dots show the achieved success rate (left y-axis) and the orange crosses the amount of drops (right y-axis) over the configured bandwidth (x-axis).

streaming [using persistent connections] one needs about the double bandwidth of the media bitrate to achieve a sufficient performance" (their setup had a constant added delay of 150 ms). The authors in [48] support this statement with a nuance: "It is found that TCP generally provides good streaming performance when the achievable TCP throughput is roughly twice the video bit rate (...)." The results with BBB10s confirm this, although it should be taken into account that the bandwidth adaptive algorithm introduces a constant ceiling at 3/4 of the perceived bandwidth and that the achievable TCP throughput is strictly below the configured limit.

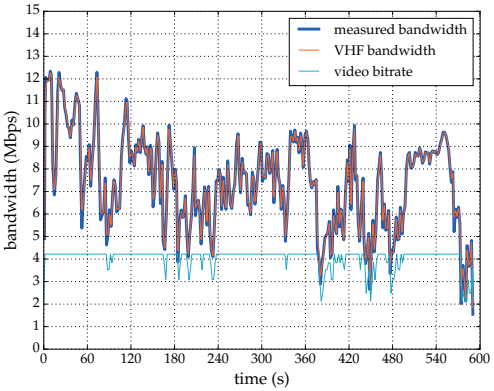
4.5.2 VLC's predictive logic

With VLC's other adaptation algorithm, the predictive logic, the success rate threshold is never reached. With the same video BBB2s, the same bandwidth search space, and same RTT delays as for the initial setup, the success rate is always close to 0 %. The first three segments get the highest available bitrate. Then the remainder of the video is played at the lowest bitrate, at a smaller resolution than the given input. When using BBB10s and persistent HTTP, the bitrate starts at the highest value and decreases, a little slower, towards the lowest one.

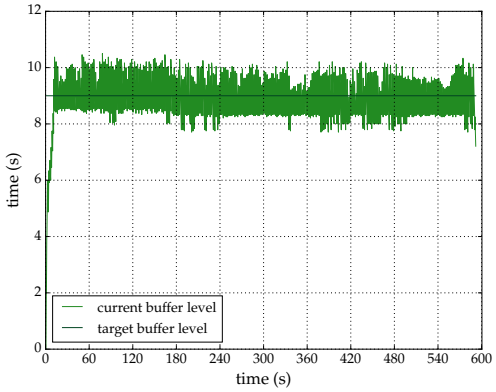
The algorithm does not work as expected: its bandwidth estimation is always close to 0, which prevents it from requesting any representation but the lowest one. In the remainder of the thesis, only the bandwidth adaptive logic is used. The calibration tests with the initial setup, and especially the failed calibration for long delays, offered a first opportunity to assess the behaviour of VLC's rate based adaptation logic with regular TCP along the variation of one network parameter: the configured bandwidth.

VLC's bandwidth adaptive algorithm favours reactivity over stability: The VHF estimation of the available bandwidth (section 3.3.2) does not smooth the observations much, the estimation and the measurements are almost overlapping. Figure 4.10a

illustrates this behaviour with a case with much bandwidth estimation variation (the configured bandwidth and delay are constant). This confirms the expectation that too frequent noise does not allow the estimation to become stable and favours its reactivity. In addition, VLC's bandwidth adaptive logic is quick to both increase and lower the quality when faced with a perceived bandwidth change, even when the buffer is not nearly empty, as shown in figures 4.10a and 4.10b. For more stability, the algorithm could benefit from a less reactive bandwidth estimation or a more conservative reaction to bandwidth estimation fluctuations. Ideally, it would also take the buffer fill state into account for switching decisions.



(a) Estimated bandwidth and chosen video bitrate over time.



(b) Buffer fill and target over time.

*Figure 4.10: Example session with VLC's bandwidth adaptive logic (*100 ms *RTT delay,* 20.08 Mbps*, initial setup).*

The end of BBB2s is more prone to quality drops: this is probably due to the fact that the last 14 video segments contain less information (they contain the end credits, which mainly show a black screen). The overhead to get those segments is comparatively higher, but not taken into account by the algorithm, which estimates the bandwidth on the size of the HTTP payload.

All in all, the algorithm chooses the bitrate that best matches the estimated instantaneous throughput, without trying to reduce switching and without taking the buffer into account.

4.5.3 CUBIC as TCP congestion control

The calibration results with CUBIC for short RTT delays are similar to those with New Reno. At 25 ms, CUBIC reaches the success rate threshold of 95 % at 9 Mbps, whereas New Reno needs 9.3 Mbps. At 50 ms, CUBIC needs 12.5 Mbps and New Reno 11.7 Mbps.

For long RTT delays, the tests do not reach the success rate threshold either. As with New Reno, they reach a peak around 50 Mbps and then stagnate at higher bandwidths.

Since CUBIC does not give significantly better results than New Reno, even at high latencies and high bandwidths, for which it has been designed, the choice to use New Reno as congestion control for the comparison does not give TCP an unfair disadvantage.

4.6 Chosen baseline cases

For short delays, I will keep the initial setup. It seems likely that MPTCP will be less able to fully utilise the network compared to TCP, because of its added connection establishment overhead. Whether this has significant repercussions on the QoE remains to be evaluated.

For long RTT delays, I will use BBB10s. This makes a comparison with high delays possible and gives an opportunity to asses the behaviour of persistent HTTP in addition to non persistent. `tcp_slow_start_after_idle` is left enabled to prevent queue overruns and because it makes the bandwidth estimation by VLC somewhat more stable.

I chose the setups described in table 4.3 as baseline cases for the comparison between TCP and MPTCP.

Table 4.3: TCP baseline cases for the comparison.
The congestion control is New Reno and the VLC logic bandwidth adaptive.

	RTT delay (ms)	**Bandwidth (Mbps)**	**Video**	**Max. bitrate (Mbps)**	**HTTP**
Case 1	25	10	BBB2s	4.2	persistent
Case 2	50	11.7	BBB2s	4.2	persistent
Case 3	100	8	BBB10s	3.8	non-persistent
Case 4	150	9	BBB10s	3.8	non-persistent

5 | Measurements with a single client

After the calibration, it is time to compare the baseline cases defined in the previous chapter with setups using MPTCP instead of TCP. This chapter presents and links application and network measurements obtained with the infrastructure described in chapter 3. A single client streams videos in a non congested environment. The goal is to understand how DASH works with TCP and MPTCP. The first step is to uncover differences in QoE performance between the two. The next step is to identify parameters favouring or hindering DASH in comparable setups.

To start, the methodology deployed to create comparable cases and to analyse them is detailed, as TCP and MPTCP are not equivalent protocols. The comparison begins with an overview of all bandwidth estimations. The QoE results are then introduced and different application and network measurements discussed.

5.1 Methodology

Comparable cases For each TCP baseline case chosen in section 4.6 of the previous chapter, I defined 10 MPTCP comparison setups with two asymmetric paths. They have the following characteristics:

- the total bandwidth used for the baseline is split across the two links with a ratio in [0.8, 0.6, 0.5, 0.4, 0.2],[1] so that MPTCP does not get additional bandwidth;
- the delays added on the links are equal to the baseline delay with a variation of $\pm$ 15 ms on the first and $\mp$ 15 ms on the second.

For ease of reading, a setup where one link has a higher bandwidth and a lower delay and the other a lower bandwidth and a higher delay is described as having one "*strong*" and one "*weak*" link, respectively. When the bandwidth and the delay are both higher on one link and lower on the second, I will refer to them as "*heavy*" and "*light*", respectively. The wording for the latter two is chosen as analogy to means of transportation: one can think of a *heavy* link as a cargo ship, carrying more goods with increased latency; a *light* link corresponds to a plane, providing for faster delivery but for a smaller load only.

QoE For each setup, the following QoE parameters are taken into account:

- the success rate as defined in section 4.1 – higher is better,
- the average video bitrate ($\bar{R}$) – higher is better,
- the total amount of quality switches (both up and down) – lower is better,
- the estimated startup delay – lower is better.

[1] Complementary ratios $1 - x$ and $1 - (1 - x)$ are two different cases. In the first, the link with a bandwidth share of $1 - x$ establishes the connection and the one with with a share of $1 - (1 - x)$ can be used for subflows. In the second case, the responsibilities are reversed.

The two remaining parameters also discussed in section 1.3.2, stalls and the switching amplitude, are not used. There are no stalls to be expected in any of the setups, since the baseline cases have sufficient network characteristics. The switching amplitude is a tricky measure. It highly depends on the available bitrates, the spacing between them, and the behaviour of the adaptation logic. In the dataset in use, the spacing is not equal. A high switching amplitude may mean that the logic directly switched up from a very low to a very high one, without ever using an intermediate representation. This actually happens in some instances, since the bandwidth adaptive logic always starts at the lowest bitrate.

QoS To characterise the traffic between the client and the server, statistics about TCP and MPTCP are obtained from the the files generated by tcpdump and processed by the MPTCP analysis scripts.[2] For each setup, the scripts compute the amount of traffic from the server to the client, the traffic share on each link, the amounts of retransmissions and MPTCP reinjections[3], the number of RTO and unnecessary RTO events identified by Tstat[4]. The values obtained with TCP Probe are also taken into account.

Analysis All tests are repeated three times to ensure the reliability of the measurements. The resulting 44×3 instances will first be used for a global comparison of the bandwidth estimation. Next I will review the QoE results of the different scenarios in cases 1 and 2 (short delays and non-persistent HTTP). I will zoom in on three results: the TCP baseline with the median success rate and the *best* and *worst* scenarios. The latter two are defined as the test instances with the median success rate among the setups with the highest and lowest average success rate over the 3 repetitions, respectively. The following step is to assess their link utilisation to tie results at the application level to events on the network. The same procedure is repeated for cases 3 and 4 (long delays and persistent HTTP).

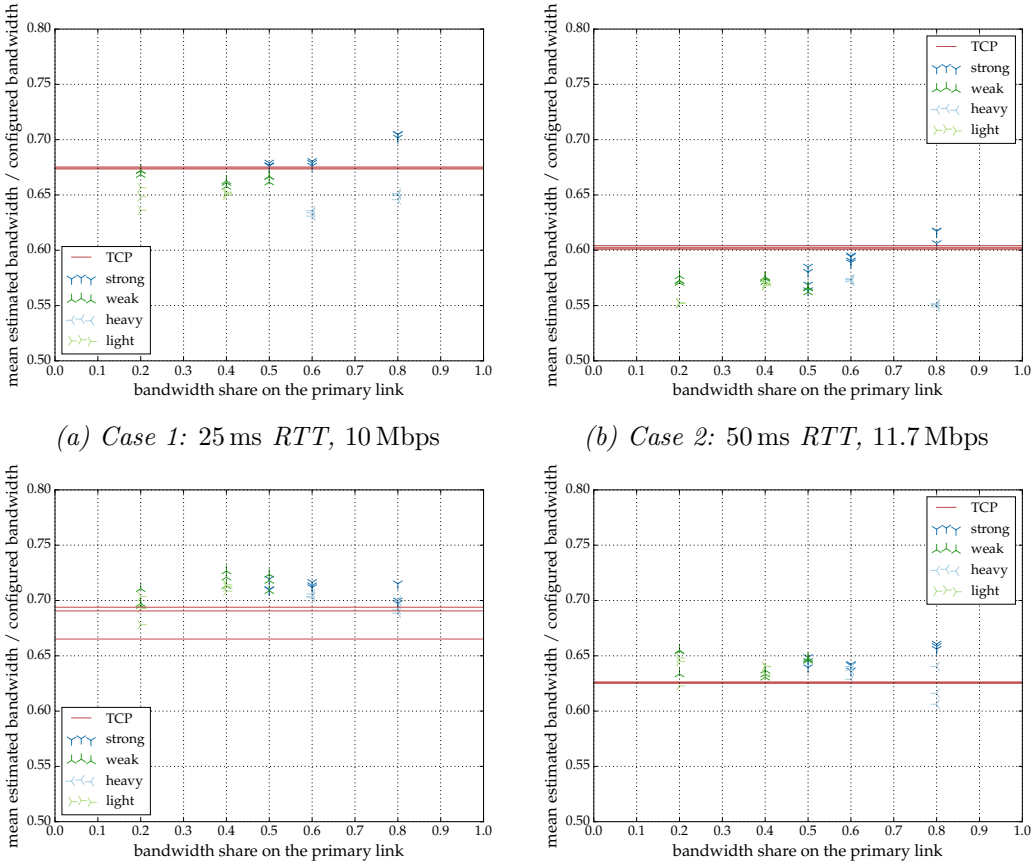
The results of this section are not fit for generalisation. They are not repeated often enough and do not sample a large enough parameter space. Instead, I present specific scenarios in an attempt to detect common and diverging characteristics between TCP and MPTCP used with DASH. The goal is to discover tendencies to explore and, if present, describe network dynamics that favor or hinder good QoE with MPTCP.

5.2 Bandwidth estimation

The bandwidth estimation performed by VLC is the most important aspect of this comparison because the bandwidth adaptive logic entirely builds on it. Let us compare the estimated throughput in all generated instances and see what changes. Figure 5.1 shows the relative average bandwidth estimation for each tested setup. The results are grouped by comparison case. Each dot on the graph represents the average bandwidth estimation for the entire session over the configured bandwidth. Its x-position is defined by the bandwidth share on the primary link. For each case, I separated the results into the four categories *strong*, *heavy*, *weak*, and *light* presented in section 5.1. The results with TCP are also drawn as lines. All results are narrow and per case the variation is never more than 10 %. In all cases MPTCP sometimes gives a higher and sometimes gives a lower estimate than TCP.

For cases 1 and 2 (figures 5.1a and 5.1b) a *strong* primary link with a large bandwidth share gives MPTCP an advantage. A *heavy* or *light* primary link always results in a

[2] Available on GitHub: https://github.com/multipath-tcp/mptcp-analysis-scripts/
[3] Reinjections are counted on the link that sent the packet first.
[4] Tstat: http://tstat.polito.it



(a) Case 1: 25 ms *RTT,* 10 Mbps

(b) Case 2: 50 ms *RTT,* 11.7 Mbps

(c) Case 3: 100 ms *RTT,* 8 Mbps

(d) Case 4: 150 ms *RTT,* 9 Mbps

Figure 5.1: Relative mean bandwidth estimation for all comparison cases. All test instances are represented. The y-axis spans 0.5 *to* 0.8*, no values fall outside that range.*

poorer performance. Giving the primary link a lower bandwidth share is also harmful.

For cases 3 and 4 (figures 5.1c and 5.1d) the distinction between primary and secondary link is less marked; symmetric setups with a *strong* primary link have similar results to those with a *weak* one. The same holds for *heavy* and *light* scenarios. One *strong* link, whether primary or secondary, is beneficial. When there is a larger asymmetry in the bandwidth shares, *heavy*/*light* links can fall behind TCP.

MPTCP follows TCP in that a higher delay yields a lower relative bandwidth estimation.[5] For non-persistent HTTP, the choice of which link is primary is more important because each new TCP connection starts over it. For persistent HTTP, the choice is smoothed out during the session. At short delays, a *strong* link with a smaller RTT than TCP's single link can improve the performance.

In the future more setups can be explored to refine the results. The constant delay variation (30 ms) between the two links can be increased or decreased, as well as the delay variation with regard to the TCP baseline. Different configured bandwidths can be tested in addition to the one calibrated with TCP. It should always be remembered that the results rely on a particular algorithm and that discrete sampling is inevitable, not least because of the available bitrates.

For now we can anticipate close QoE results between TCP and MPTCP from this

[5] In case 1 compared to 2 and in case 3 compared to 4. Another comparison is not possible because of the use (or not) of HTTP persistent connections.

preliminary comparison. In absolute values, all instances achieve a mean estimated bandwidth sufficient for the highest video bitrate. However QoE includes other parameters that need investigation.

5.3 Short RTT delays and non-persistant HTTP

For cases 1 and 2 we can expect an equal QoE experience with MPTCP. The performance depends on the resource pooling perceived by VLC. As long as the estimated bandwidth stays above 5.6 Mbps,[6] there is no down switching.

5.3.1 QoE comparison

Table 5.1 shows the *best* and *worst* scenarios for the 1st case. With 25 ms baseline RTT delay the overall QoE performance of MPTCP is at its highest when there is a clear difference between one *strong* primary link and one *weak* secondary link. It is lowest when the primary link is *heavy* and the secondary *light*.

The estimated startup delay ranges from 3.8 s to 5.1 s, without noticeable differences between TCP and MPTCP.

Table 5.1: Selected QoE comparison results for case 1. The TCP baseline is highlighted.

Setup				**QoE results**			
Link 1		**Link 2**		**Success rate (%)**	**$\bar{R}$ (Mbps)**	**Quality switches**	**Startup delay (s)**
(Mbps)	**(ms)**	**(Mbps)**	**(ms)**				
8	10	2	40	98.66	4.21	8	3.77
10	25			97.32	4.20	8	3.99
6	40	4	10	88.29	4.14	40	4.23

Table 5.2 shows the *best* and the two *worst* scenarios for the 2nd case. With 50 ms baseline RTT delay the overall QoE performance of MPTCP again reaches a peak with a *strong* primary and a *weak* secondary link. The two worst setups are one *heavy* and one *light* link with a maximum difference in bandwidth, whichever is used as primary.

The estimated startup delay varies from 3.7 s to 5.9 s, but this time it is always longer with MPTCP.

Table 5.2: Selected QoE comparison results for case 2. The TCP baseline is highlighted.

Setup				**QoE results**			
Link 1		**Link 2**		**Success rate (%)**	**$\bar{R}$ (Mbps)**	**Quality switches**	**Startup delay (s)**
(Mbps)	**(ms)**	**(Mbps)**	**(ms)**				
9.36	35	2.34	65	96.32	4.19	12	4.01
11.7	50			95.32	4.18	17	3.80
9.36	65	2.34	35	81.27	4.08	56	3.88
2.34	35	9.36	65	81.94	4.08	57	4.19

[6] $4.2 \times (3/4)^{-1} = 5.6$, where 4.2 is the max. available video bitrate of BBB2s and 3/4 the factor by which VLC multiplies its bandwidth estimation to set the threshold for the highest bitrate it can request.

For short delays, a setup with one *strong* primary and one *weak* secondary link gives a slightly better success rate, average bitrate, and less switching. This is in line with the bandwidth comparison results in section 5.2, and good news for two reasons. Firstly, mobile devices with a WiFi (*strong*) and a cellular (*weak*) interface have exactly that profile. Moreover, users generally favour WiFi over cellular, making it the primary by default. Secondly, no tradeoff is needed between the QoE indicators. When streaming over a *heavy* and a *strong* link, the performance decreases, more at higher delays. To understand why, let us assess the link utilisation in all presented cases.

5.3.2 Link utilisation

Let us start by looking at the mean estimated bandwidth ($\bar{R}$) and its standard deviation (std). The values are reproduced in table 5.3. They confirm that a high and stable bandwidth improves the QoE and implies that MPTCP does not necessarily add variability.

Table 5.3: Selected mean bandwidth estimation and variation for cases 1 and 2. Same setups (and same order) as for tables 5.1 and 5.2.

	$\bar{R} \pm$ std (Mbps)	
	Case 1	**Case 2**
configured	10	11.7
best	7.04 ± 0.47	7.23 ± 0.79
TCP	6.74 ± 0.54	7.03 ± 0.81
worst	6.34 ± 0.67	6.46 ± 1.01
		6.45 ± 1.01

What happens on the two links that causes an increase or decrease in this estimation? Tables 5.4 and 5.5 show network measures of the traffic from server to client taken at the server for the previously featured instances.

Table 5.4: Selected link utilisation results for case 1. The TCP baseline is highlighted.

Setup				**Network measurements**				
	ratio	(Mbps)	(ms)	traffic (% of total)	retransmissions (pkts)	RTO (pkts)	unnecessary RTO (pkts)	reinjections (pkts)
strong link 1	0.8	8	10	82.65	1084	2	0	0
weak link 2	0.2	2	40	17.91	135	0	0	0
single link	1	10	25	100	121	6	13	–
heavy link 1	0.6	6	40	57.40	61	30	0	0
light link 2	0.4	4	10	43.42	1717	7	0	0

Table 5.5: Selected link utilisation results for case 2. The TCP baseline is highlighted.

Setup				**Network measurements**				
	ratio	(Mbps)	(ms)	traffic (% of total)	retransmissions (pkts)	RTO (pkts)	unnecessary RTO (pkts)	reinjections (pkts)
strong link 1	0.8	9.36	35	82.31	65	5	1	0
weak link 2	0.2	2.34	65	17.74	28	7	0	1
single link	1	11.7	50	100	86	48	13	–
heavy link 1	0.8	9.36	65	79.21	56	0	0	0
light link 2	0.2	2.34	35	20.72	82	1	0	0
light link 1	0.2	2.34	35	23.43	20	0	0	0
heavy link 2	0.8	9.36	65	76.62	78	19	0	1

In the presented scenarios, the link with the lowest delay always carries a little more traffic than its configured bandwidth share. This influences the bandwidth estimation. When the primary link has a lower delay, there are as many connections with a single subflow as there are quality switches. This is due to the initialisation segment needed to play the new bitrate, which is small enough that it is sent fast enough not to leave time for subflow addition. When the primary link has a higher delay, the second subflow is established and wastes resources because it is not used.

In case 1 there are many more retransmissions with MPTCP than with TCP, always on the link with the lower delay. This means that TCP spends more time in congestion avoidance on that link. It explains the increased traffic share, which is good for a *strong* link, less so for a *light* one. This does not happen in case 2; the increase in delay is enough to keep TCP in slow start. Reinjections only happen once in case 2; a bigger congestion window than the receive windows rarely occurs.

5.3.3 Findings

At short delays, with non-persistent HTTP, and with short video segments, a *strong* primary link with much more bandwidth than the *weak* secondary one yields the best performance with MPTCP because the protocol favours the link with a lower delay. More traffic is sent with a decreased end-to-end delay, leading to a higher bandwidth estimation and a better network utilisation. Setups with *heavy* and *light* links should be avoided because the added overhead of using MPTCP lessens the bandwidth estimation and adds variability, which impedes the overall QoE, leading to a lower average bitrate and more frequent switching.

5.4 Long RTT delays and persistent HTTP

For cases 3 and 4 we can expect better resource pooling with MPTCP thanks to the persistent connections and longer segments, avoiding recurring connection establishment

and subflow additions and permitting more stable bandwidth estimations. However, due to the higher delays, retransmissions become more likely, and there might be more reinjections as well.

5.4.1 QoE comparison

Table 5.6 gives the QoE performance of the *best* and *worst* scenarios for the 3rd case, at 100 s baseline RTT delay. The average bitrates are equal and the amount of switches really close. All 11 tested scenarios in this case have a similar performance. This means that MPTCP efficiently exploits the links whether they have similar or diverging characteristics.

Table 5.6: Selected QoE comparison results for case 3. The TCP baseline is highlighted.

Setup				**QoE results**			
Link 1		**Link 2**		**Success rate (%)**	$\bar{R}$ **(Mbps)**	**Quality switches**	**Startup delay (s)**
(Mbps)	**(ms)**	**(Mbps)**	**(ms)**				
3.2	115	4.8	85	98.33	3.76	1	9.09
8	100			95	3.76	4	9.36
1.6	85	6.4	115	95	3.76	3	8.83

Table 5.7 shows the best and worst comparison setups for the 4th case. All results are again very close. One setup is visibly worse: a *heavy* primary link with a bandwidth share of 0.8.

Table 5.7: Selected QoE comparison results for case 4. The TCP baseline is highlighted.

Setup				**QoE results**			
Link 1		**Link 2**		**Success rate (%)**	$\bar{R}$ **(Mbps)**	**Quality switches**	**Startup delay (s)**
(Mbps)	**(ms)**	**(Mbps)**	**(ms)**				
7.2	135	1.8	165	96.67	3.75	2	6.60
9	150			95	3.74	3	7.96
7.2	165	1.8	135	85	3.71	8	8.21

In cases 3 and 4, there are only small variations in the QoE performance. The startup delays are higher but vary less than in the first two cases. In general, MPTCP performs slightly better than TCP. Let us have a look at the link utilisation in the selected cases.

5.4.2 Link utilisation

The mean estimated bandwidth and its standard deviation for the instances selected in the previous QoE analysis are reproduced in table 5.8. As for the QoE results, the values are close, in line with the observations in section 5.3.2: a higher and more stable estimation gives a better QoE.

Tables 5.9 and 5.10 contain the network measurements for the comparison of cases 3 and 4. The previous observations about the link load shares only hold for one instance: the *worst* setup in case 4 sends more traffic than its configured share and has a lower

Table 5.8: Selected mean bandwidth estimation and variation for cases 3 and 4. Same setups (and same order) as for tables 5.6 and 5.7.

	$\bar{R} \pm$ std (Mbps)	
	Case 3	**Case 4**
configured	8	9
best	5.77 ± 0.26	5.93 ± 0.84
TCP	5.55 ± 0.30	5.63 ± 0.81
worst	5.63 ± 0.32	5.54 ± 0.86

Table 5.9: Selected link utilisation results for case 3. The TCP baseline is highlighted.

Setup				**Network measurements**				
	ratio	(Mbps)	(ms)	traffic (% of total)	retransmissions (pkts)	RTO (pkts)	unnecessary RTO (pkts)	reinjections (pkts)
weak link 1	0.4	3.2	115	40.95	103	10	0	0
strong link 2	0.6	4.8	85	59.14	71	0	0	0
single link	1	8	100	100	0	0	0	–
light link 1	0.2	1.6	85	19.81	60	7	0	0
heavy link 2	0.8	6.4	115	80.37	288	67	0	0

Table 5.10: Selected link utilisation results for case 4. The TCP baseline is highlighted.

Setup				**Network measurements**				
	ratio	(Mbps)	(ms)	traffic (% of total)	retransmissions (pkts)	RTO (pkts)	unnecessary RTO (pkts)	reinjections (pkts)
strong link 1	0.8	7.2	135	79.66	478	113	4	0
weak link 2	0.2	1.8	165	20.59	20	6	0	0
single link 1	1	8	100	100	367	37	0	–
heavy link 1	0.8	7.2	165	76.76	360	72	2	0
light link 2	0.2	1.8	135	23.48	56	4	0	24

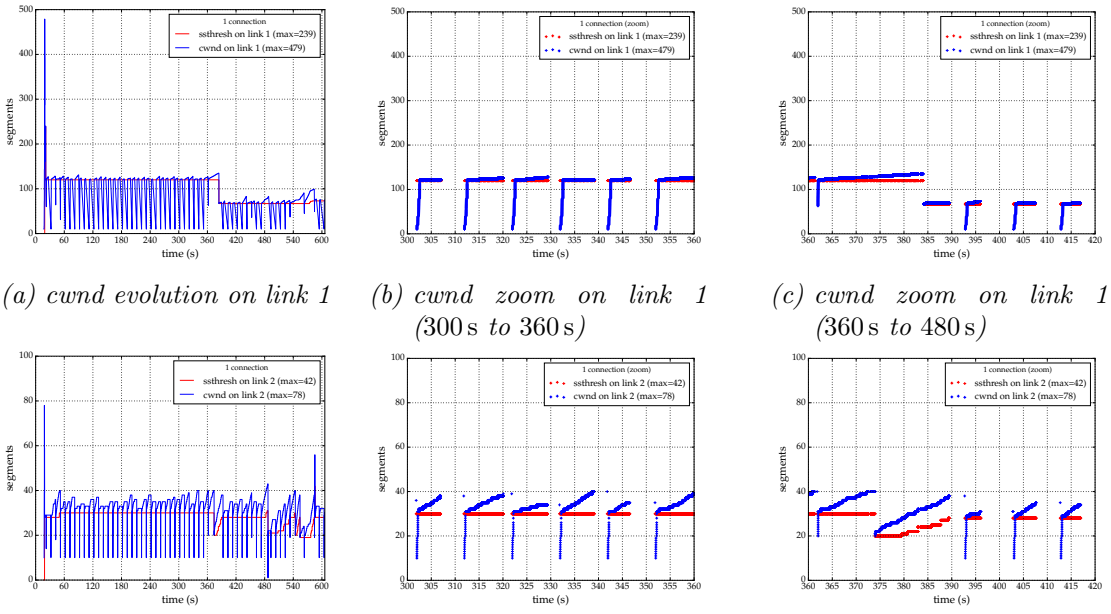
delay. Otherwise the links' traffic share mirrors their configured asymmetry in terms of bandwidth.

Let us take a closer look at the congestion window evolution of the *worst* setup in case 4. To understand what happens, let me first introduce a way of looking at HAS transfer with persistent connections. When using HAS and sufficiently large segments to transition from slow start to congestion avoidance, there are three phases of data transfer [18, 48]:

1. during the *initial burst phase* the congestion window is filled (TCP slow start),
2. during the *ACK clocking phase* the server sends data upon reception of an ACK (TCP congestion avoidance),
3. during the *trailing ACK phase* the server just waits for the last ACKs but has no more data to send.

At the end of such a cycle, the connection becomes idle. A packet loss in the 3$^{\text{rd}}$ phase has the most negative impact. The server does not have any more data to send and can not use fast retransmit. The RTT delay increases, which leads to a retransmission timeout (RTO). The congestion window drops and the next transfer cycle has to start at that new low value.

Figure 5.2 shows the size of the congestion window (cwnd) over time for the *worst* setup in case 4. Around 360 s, there is a change in the "on-off" traffic pattern. The server sends 2 video segments fast enough to avoid the idle phase. At the end of the transfer, in the trailing ACK phase, a loss occurs. The cwnd and the ssthresh drop considerably. The next time data is transfered, the second link with less bandwidth but a smaller delay is (relatively) favoured. This slows down the transmission and prevents the first link from recovering, leading to a lower bandwidth estimation and 4 down-switches.



(a) cwnd evolution on link 1 *(b) cwnd zoom on link 1 (300 s to 360 s)* *(c) cwnd zoom on link 1 (360 s to 480 s)*

(d) cwnd evolution on link 2 *(e) cwnd zoom on link 2 (300 s to 360 s)* *(f) cwnd zoom on link 2 (360 s to 480 s)*

Figure 5.2: Zoom on the worst instance in case 4: MPTCP, 7.2 Mbps *and* 165 ms *RTT on the primary link,* 1.8 Mbps *and* 135 ms *on the second link.*

5.4.3 Findings

At high delays, with segments of 10 s, and with persistent HTTP, there are only small variations in the QoE performance. MPTCP is able to aggregate the available resources and to offer a behaviour that mirrors TCP. Both protocols are nonetheless subject to the same complexity of HAS and congestion control interactions.

5.5 Conclusion

The comparison results do not produce clear-cut results. Big asymmetries in the links' bandwidths and delays can be both good and bad for QoE performance, as can be links with similar characteristics. However, the best results in all tested cases have in common that they use one *strong* and one *weak* link. The worst results always use the a setup with a *heavy* and a *light* link. Other parameters need to be assessed before a generalisation is possible and applicable to real networks. In particular, bandwidth variability, lossy networks, and higher differences in the two links for MPTCP could prove to have a stronger impact.

In the end, the performance greatly depends on the adaptation logic and how it actually exploits the network. It is responsible for scheduling the requests, estimating the bandwidth, responding to perceived fluctuations, dealing with the buffer.

6 | Adding congestion

Both TCP and MPTCP use congestion control. All previous tests made the network fully available for a single streaming session. In this chapter, congestion is added, introducing bandwidth variability all at once. After deriving a set of measurement goals from known issues, the updated infrastructure is presented. It is followed by the methodology and a discussion of the results.

6.1 Known issues

When several DASH players share a network, they influence each other's performance through TCP's congestion control, which in turn affects the adaptation logic and vice versa. Three factors have been identified as particularly challenging with TCP in the literature [3, 11, 48].

Stability: As previously identified, each player has an "on-off" period. During the "on" phase, it downloads video chunks; during the "off" phase, it stays idle. In the case of VLC, the bandwidth is estimated over one "on" part. Depending on how the periods of several players are synchronised, this estimation can greatly vary in time and lead to oscillations in the bitrate selection, causing frequent switching.

Fairness: If one client manages to get a large share of the bandwidth, the other clients will compete for the remains. The first client will be able to stream at high bitrate, while the other players will limit the quality they request, unaware of the unfairness. This behaviour shows how the doubly nested feedback loop of TCP and DASH can have a negative effect.

Network underutilisation: Even if stability and fairness are provided, multiple clients can end up underutilising the network because they underestimate the available bandwidth due to the presence of others on the network.

As fairness and efficient network utilisation are two of MPTCP's design goals, let us take all three parameters to assess its performance compared to TCP when multiple players use a network.

6.2 Infrastructure update

This section describes the new infrastructure as an adaptation of the one presented in chapter 3 and settles a few parameters.

6.2.1 Overview

The new testing environment comprises one server and two clients on the same host machine. The clients are connected to the server through two shared independant links. From the setup described in chapter 3, I took a few steps to get the new infrastructure.

First I cloned[1] the initial client VM to get a new identical VM, referred to as the *rival* VM. Next I added a TAP interface to each of the two bridges and connected them to the new VM. Each bridge now contains 3 TAPs, one for each VM. Finally, I adapted and extended the routing tables and host files to make the new machine reachable. A high-level overview on the new infrastructure is given in figure 6.1.

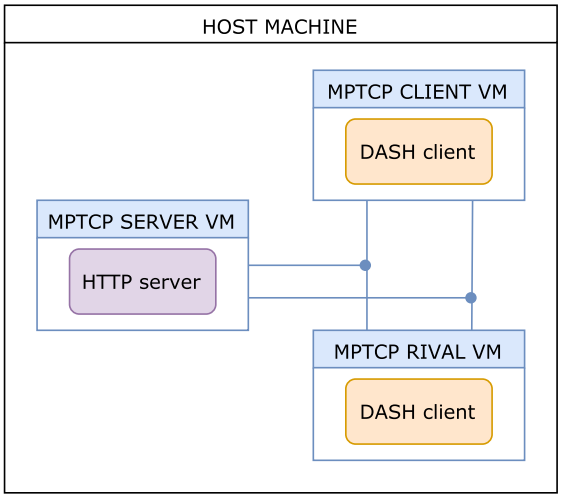


Figure 6.1: High-level view on the updated infrastructure: the new rival VM is on the same host machine and connected to the same two disjoint virtual links as the client and server VMs.

6.2.2 Parameters

All parameters initially fixed in section 3.5 remain unchanged. The TCP baseline cases 1 and 3 from the calibration (section 4.6) are also directly reused, together with the corresponding setups that gave the *best* and *worst* QoE performance (sections 5.3.1 and 5.4.1). Achieving streaming at maximum bitrate can no longer be expected, at least not for both players simultaneously. This introduces competition.

Type of congestion: All congestion tests are performed with two clients streaming a DASH video. No other network traffic is added.

Client sameness: Both clients behave in the same way during one session. They use either TCP or MPTCP, play the same video with the bandwidth adaptive logic, and use the same primary link.

Synchronisation: Both clients start playback approximately at the same time. The host machine takes the same steps as detailed in section 3.6. It launches two threads to start playback on both the client and the rival VM in parallel.

6.3 Methodology

Unlike for the comparison with a single client, the tests are not repeated for the congestion comparison. Instead, all setups were tested with twice the calibration bandwidth to make sure that, given sufficient bandwidth, two players are able to stream at highest bitrate.

[1] VirtualBox allows cloning with a single command.

Congestion comparison

The comparison is limited to the three issues explained in section 6.1. *Stability* is assessed with the video bitrate stability, the amount of switches, and the variability of the bandwidth estimation. *Fairness* is evaluated on the QoE performance of each client and the estimated bandwidth, as a share of the total bandwidth. *Network utilisation* is a joint measure. Both bandwidth estimations are linearly interpolated and added up. All three totals with congestion are compared against each other and against their counterpart with a single client.

6.4 Results

For both cases 1 and 3 presented and discussed hereunder, the bandwidth is no longer sufficient for streaming at maximum bitrate because it is shared. Without network underutilisation, both clients should however be able to choose bitrates above the minimum. The congestion creates actual bandwidth fluctuations in the available bandwidth. In view of the previous performance of VLC's bandwidth adaptive logic, this is likely to add even more variations in the bandwidth estimation.

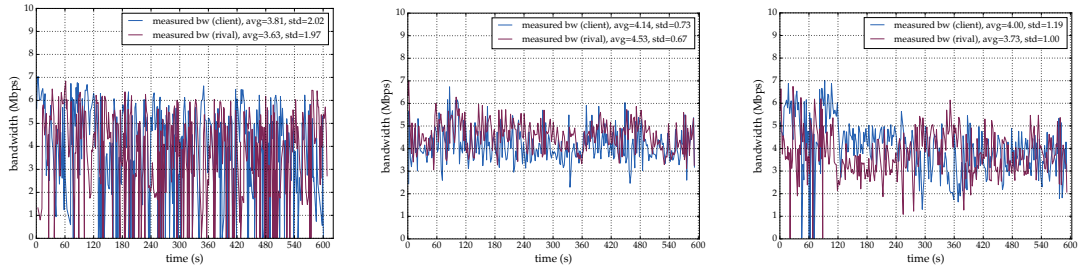
6.4.1 Case 1

The TCP baseline for case 1 and the setups identified as *best* and *worst* in section 5.3 have are configured as summarised in table 6.1.

Table 6.1: Best, TCP, and worst setups for case 1, taken from section 5.3.

	Link 1		**Link 2**	
	(Mbps)	(ms)	(Mbps)	(ms)
best	8	10	2	40
TCP	10	25		
worst	6	40	4	10

Stability All three setups are highly unstable. With only one client, there were between 8 and 40 switches, in the *best* and *worst* case using MPTCP, respectively. With two clients, both experience 177 to 220 switches, while there are only 299 video segments. This is due to the bandwidth estimation, which is more variable with MPTCP than with TCP. Figure 6.2 shows the evolution of the estimated bandwidth at the two clients.



(a) MPTCP best *(b) TCP* *(c) MPTCP worst*

Figure 6.2: Bandwidth estimation with two clients in case 1.

Fairness The TCP setup favours the second client at the expense of the first. Its average video bitrate is 3.16 Mbps, compared to 2.85 Mbps, with the same variability. The clients using MPTCP are treated more fairly, as there is less difference between the average video bitrates. The estimated bandwidths are more equal than with TCP.

Network utilisation The sum of the bandwidth estimations with two clients is higher with TCP than with MPTCP: 8.73 Mbps compared to 7.45 Mbps and 7.76 Mbps in the *best* and *worst* setup, respectively. In all tests with two clients, the network utilisation increases compared to a single client. The fact that two clients are able to stream the video without interruptions and with an average higher bitrate than the minimum confirms that a single client does not fully utilise the available bandwidth.

Conclusion

Stability is not achieved, neither with TCP, nor with MPTCP. Both clients experience high bitrate oscillations throughout the video session. MPTCP bandwidth estimation is more variable than TCP's, which leads to a higher bitrate variability. However, the switching frequency is already so high that this has not much weight from a QoE point of view. MPTCP enforces more fairness than TCP, but at the expense of a relative network underutilisation. This represents a tradeoff between its two design goals.

6.4.2 Case 3

Table 6.2 is a reminder of the TCP baseline for case 3 and the setups identified as *best* and *worst* in section 5.4.

Table 6.2: Best, TCP, and worst setups for case 3, taken from section 5.4.

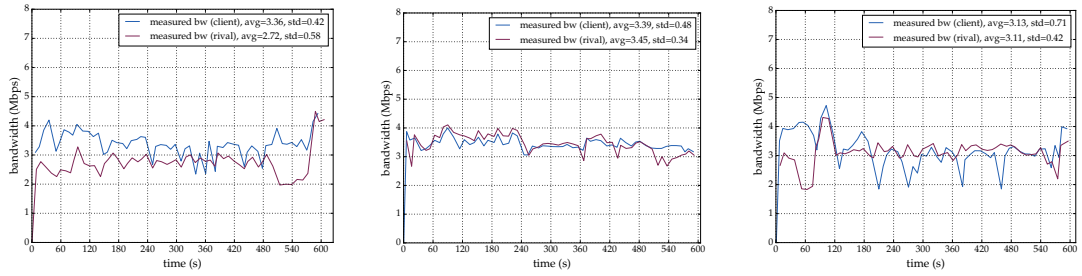
	Link 1		**Link 2**	
	(Mbps)	(ms)	(Mbps)	(ms)
best	3.2	115	4.8	85
TCP	8	100		
worst	1.6	85	6.4	115

Stability Two clients on the network again lead to more switching, but not necessarily for both. With a single client, the three setups experienced between 1 and 4 switches for 60 video segments. With two competing players, it varies between 4 and 22. In the present case, less switching, even if providing a stable bitrate, happens when the minimum bitrate is already used and no more down switching is possible. Stability is not always due to a stable bandwidth estimation.

Using MPTCP leads to equal or higher bandwidth estimation variations than with TCP. Thanks to persistent HTTP and to longer segments, they are lower than in case 1 with congestion (section 6.4.1).

Fairness When it comes to fairness in the average bitrate, TCP performs better than MPTCP. The two clients have exactly the same average bitrate (2.37 Mbps). With MPTCP, the bitrate averages are a little more apart. Considering the switching, however, no protocol has a significantly better performance.

Figure 6.3 shows the evolution of the bandwidth estimation in all three tests. Throughout the video session, both clients using TCP (figure 6.3b) estimate the bandwidth in the same way. With MPTCP (figures 6.3a and 6.3c), the first client more often monopolises the network and achieves a higher estimate than the second client.



(a) MPTCP best *(b) TCP* *(c) MPTCP worst*

Figure 6.3: Bandwidth estimation with two clients in case 3.

Network utilisation The sum of bandwidth estimations are higher with TCP than with MPTCP, as in case 1 with congestion (section 6.4.1). Both are nevertheless higher than when a single client uses the same network.

Conclusion

Neither MPTCP nor TCP achieve stability and fairness in terms of QoE for both clients, yet with TCP, the clients have more equal bandwidth estimation. The total estimated bandwidth is always higher than with only one client, so the network utilisation is improved in all cases. TCP does this better than MPTCP.

6.5 Conclusion and future work

The results presented and discussed in this chapter bring the analysis one step closer to real traffic where multiple users share the resources. Using only two clients and with no other applications, they are still a simplification and constitute very specific cases. Despite that, they are even less clear-cut than during the comparison with a single client. MPTCP seems to degrade stability. It improves fairness, but falls behind TCP for network utilisation. All in all, MPTCP is subject to the same problems as identified for TCP.

Future work should include a thorough characterisation of MPTCP and DASH in congested networks, especially with setups close to real use cases.

Conclusion

In this thesis I examine how well Dynamic Adaptive Streaming over HTTP and Multipath TCP perform when used together for video streaming. Different setups are compared using quality of experience as main performance measure. I uncovered scenarios where MPTCP adds to the performance, and setups where it is outperformed by TCP.

By reviewing the state-of-the-art for Dynamic Adaptive Streaming over HTTP as well as for multipath TCP, I created a starting point for a combined performance analysis. Several elements had to be taken into account. DASH's complexity lies in the design of its adaptation algorithm, which reacts and acts on the network it uses. MPTCP extends TCP and aims to perform at least as good while remaining fair and resource efficient.

After putting together the testing infrastructure, I selected four baseline TCP cases where the bandwidth was chosen according to the delay to achieve streaming at the maximum available bandwidth. During this calibration, the importance of underlying network events was made clear due to double feedback loop problems. Whether proposed solutions for TCP can help with MPTCP remains to be studied. The equal importance of the adaptation algorithm was highlighted by VLC's often unstable and quite underperforming bandwidth estimation. Even if using VLC proved suboptimal in terms of achievable results, sticking to it made sense, as the player is widely used.[2] Since DASH does not specify an adaptation logic, and since there are diverging opinions on an optimal implementation, the chances are high that many different algorithms will coexist in the near future, among which not all will perform well in all cases.

The comparison results, albeit particular rather than general and not pronounced, bring to light where MPTCP can be beneficial. MPTCP's performance depends on the link's delays and bandwidth, and how they are combined. It is highest when MPTCP uses a *strong* link (low delay, high bandwidth) and a *weak* one (high delay, low bandwidth). When no HTTP persistent connections are used, the choice of the primary link is crucial. With persistent HTTP, MPTCP's resource pooling is efficient in all cases, but does not offer a significant advantage compared to TCP. These results are tendencies, and further investigations will have to look, i.a., into bandwidth variability, lossy networks, and higher asymmetries in the two links.

After analysing a single client in an entirely available network, it was time to make the infrastructure more realistic. This was done by expanding the setup with an additional client, introducing a competing player, together with congestion, and variable bandwidth. In the tested setups, chosen as those where MPTCP previously performed relatively well and poorly, both MPTCP and TCP experience well-known issues: instability, unfairness, and network underutilisation. This time MPTCP exacerbates the instability and the underutilisation, while sometimes providing for more fairness than TCP. If nothing else,

[2] The VideoLAN organisation states that version 2.2.1 was downloaded 197 059 343 times (https://www.videolan.org/vlc/stats/downloads.html, accessed 03-January-2017).

this indicates that correctly using MPTCP for DASH video streaming is more complex than with TCP, where there are already many problems.

Throughout this thesis, it appears that for DASH and MPTCP to work well together, improving and adapting the network quality alone is not enough. The bandwidth estimation and bitrate selection of the adaptation logic are paramount. A holistic approach is needed to maximise the QoE. Therefore, it is necessary to understand the interaction between the application and the transport layers. MPTCP adds complexity to problems already identified and not yet completely solved with TCP, as well as new issues.

This analysis barely scratched the surface of the numerous possible cases to test. It is not statistically relevant, but gives insights to directions of further exploration. There are many parameters to refine: the studied network profiles, the adaptation algorithm, the specific protocol settings, different types of videos, multiple players or other applications using the network. As mentioned in the first chapter, the opinions on whether the transport protocol and the application should work more closely together or divide their respective responsibilities better go in different directions. In any case, it is helpful to further study and measure what happens when combining DASH with MPTCP, aiming to ensure that the best performance can always be achieved.

Bibliography

[1] Adobe. *HTTP Dynamic Streaming*. [Online; accessed 14-June-2016]. URL: https://www.adobe.com/products/hds-dynamic-streaming.html.

[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*. ACM. 2011, pp. 157–168.

[3] Saamer Akhshabi et al. "What happens when HTTP adaptive streaming players compete for bandwidth?" In: *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*. ACM. 2012, pp. 9–14.

[4] Sébastien Barré, Christoph Paasch, and Olivier Bonaventure. "Multipath TCP: from theory to practice". In: *International Conference on Research in Networking*. Springer. 2011, pp. 444–457.

[5] Arkadiusz Biernacki. "Analysis and modelling of traffic produced by adaptive HTTP-based video". In: *Multimedia Tools and Applications* (2016), pp. 1–22. ISSN: 1573-7721. DOI: 10.1007/s11042-016-3623-8. URL: http://dx.doi.org/10.1007/s11042-016-3623-8.

[6] Blender Institute. *Big Buck Bunny*. [Online; accessed 3-June-2016]. Apr. 2008. URL: https://peach.blender.org/.

[8] Olivier Bonaventure, Mark Handley, and Costin Raiciu. "An overview of Multipath TCP". In: *USENIX login;* 37.5 (Oct. 2012), pp. 17–23. ISSN: 1044-6397.

[9] Olivier Bonaventure, Christoph Paasch, and Gregory Detal. *Use Cases and Operational Experience with Multipath TCP*. Internet-Draft draft-ietf-mptcp-experience-02. IETF Secretariat, July 2015. URL: %5Curl%7Bhttp://www.ietf.org/internet-drafts/draft-ietf-mptcp-experience-02.txt%7D.

[10] Olivier Bonaventure and SungHoon Seo. "Multipath TCP Deployments". In: *IETF Journal*. Vol. 12. 2. Berlin, Germany: IETF, Nov. 2016. URL: https://www.ietfjournal.org/multipath-tcp-deployments/.

[11] Te-Yuan Huang Nikhil Handigol Brandon and Heller Nick McKeown Ramesh Johari. "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard". In: *Proc. ACM SIGCOMM Conf. Internet Measurement*. 2012.

[12] Wael Cherif et al. "DASH Fast Start Using HTTP/2". In: *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV '15. Portland, Oregon: ACM, 2015, pp. 25–30. ISBN: 978-1-4503-3352-8. DOI: 10.1145/2736084.2736088. URL: http://doi.acm.org/10.1145/2736084.2736088.

[13] Cisco. "Cisco Visual Networking Index: Forecast and methodology, 2015–2020". In: *CISCO White paper* (2016).

[14] Xavier Corbillon et al. “Cross-Layer Scheduler for Video Streaming over MPTCP”. In: *Proc. of ACM MMSys* (2016).

[15] DASH Industry Forum. *dash.js – DASH IF Reference Client*. [Online; accessed 4-July-2016]. Jan. 2016. URL: https://github.com/Dash-Industry-Forum/dash.js/wiki.

[16] Luca De Cicco et al. “Elastic: a client-side controller for dynamic adaptive streaming over http (dash)”. In: *Packet Video Workshop (PV), 2013 20th International*. IEEE. 2013, pp. 1–8.

[17] Quentin De Coninck and Matthieu Baerts. *MPTCP Analysis Scripts*. http://github.com/multipath-tcp/mptcp-analysis-scripts. 2015.

[18] Jairo Esteban et al. “Interactions between HTTP adaptive streaming and TCP”. In: *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*. ACM. 2012, pp. 21–26.

[19] Mojgan Ghasemi et al. “Performance Characterization of a Commercial Video Streaming Service”. In: *arXiv preprint arXiv:1605.04966* (2016).

[20] Emanuele Goldoni, Giuseppe Rossi, and P Gamba. “Improving available bandwidth estimation using averaging filtering techniques”. In: *Rapport technique, Università degli Studi di Pavia, Laboratorio Reti* (2008).

[21] Benjamin Hesmans and Olivier Bonaventure. “An enhanced socket API for Multipath TCP”. In: *Proceedings of the 2016 Applied Networking Research Workshop*. ACM. 2016, pp. 1–6.

[22] Te-Yuan Huang et al. “A buffer-based approach to rate adaptation: Evidence from a large video streaming service”. In: *ACM SIGCOMM Computer Communication Review* 44.4 (2015), pp. 187–198.

[23] ISO/IEC. *ISO/IEC 23009-1:2014. Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*. [Publicly available at http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip]. 2014.

[24] ITU-T Recommendations. “P. 10: Vocabulary for performance and quality of service, Amendment 2: New definitions for inclusion in Recommendation ITU-T P. 10/G. 100”. In: *Int. Telecomm. Union, Geneva* (2008).

[25] Cyriac James et al. “Is Multipath TCP (MPTCP) Beneficial for Video Streaming over DASH?” In: *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*. IEEE. 2016, pp. 331–336.

[26] Junchen Jiang, Vyas Sekar, and Hui Zhang. “Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE”. In: *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM. 2012, pp. 97–108.

[27] Ramin Khalili et al. “MPTCP is not pareto-optimal: performance issues and a possible solution”. In: *IEEE/ACM Transactions on Networking* 21.5 (2013), pp. 1651–1665.

[28] Tomas Kupka, Pal Halvorsen, and Carsten Griwodz. “Performance of on-off traffic stemming from live adaptive segmented HTTP video streaming”. In: *Local Computer Networks (LCN), 2012 IEEE 37th Conference on*. IEEE. 2012, pp. 401–409.

[29] Stefan Lederer, Christopher Müller, and Christian Timmerer. “Dynamic adaptive streaming over HTTP dataset”. In: *Proceedings of the 3rd Multimedia Systems Conference*. ACM. 2012, pp. 89–94.

[30] Zhi Li et al. “Probe and adapt: Rate adaptation for http video streaming at scale”. In: *IEEE Journal on Selected Areas in Communications* 32.4 (2014), pp. 719–733.

[31] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. “Rate adaptation for adaptive HTTP streaming”. In: *Proceedings of the second annual ACM conference on Multimedia systems*. ACM. 2011, pp. 169–174.

[32] Brendan Long. *The Structure of an MPEG-DASH MPD*. [Online; accessed 3-April-2016]. Mar. 2015. URL: https://www.brendanlong.com/the-structure-of-an-mpeg-dash-mpd.html.

[33] Konstantin Miller et al. “Adaptation algorithm for adaptive streaming over HTTP”. In: *2012 19th International Packet Video Workshop (PV)*. IEEE. 2012, pp. 173–178.

[34] Christopher Müller. *VLC Plugin uses now Persistent Connections and Pipelining* . [Online; accessed 3-April-2016]. Apr. 2013. URL: http://www-itec.uni-klu.ac.at/dash/?p=809.

[35] Christopher Müller, Stefan Lederer, and Christian Timmerer. “An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments”. EN. In: *Proceedings of the Fourth Annual ACM SIGMM Workshop on Mobile Video (MoVid12)*. Ed. by Mohamed Hefeeda et al. Chapel Hill, North Carolina, USA: ACM, Feb. 2012, pp. 37–42.

[36] Christopher Müller and Christian Timmerer. “A VLC Media Player Plugin Enabling Dynamic Adaptive Streaming over HTTP”. In: *Proceedings of the 19th ACM International Conference on Multimedia*. MM ’11. Scottsdale, Arizona, USA: ACM, 2011, pp. 723–726. ISBN: 978-1-4503-0616-4. DOI: 10.1145/2072298.2072429. URL: http://doi.acm.org/10.1145/2072298.2072429.

[37] Christopher Müller et al. “Dynamic adaptive streaming over HTTP/2.0”. In: *Multimedia and Expo (ICME), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1–6.

[38] Hyunwoo Nam, Kyung-Hwa Kim, and Henning Schulzrinne. “QoE Matters More Than QoS: Why People Stop Watching Cat Videos”. In: *IEEE INFOCOM* (2016).

[39] Christoph Paasch. “Improving Multipath TCP”. PhD thesis. Louvain-la-Neuve, Belgium: Louvain School of Engineering, Oct. 2014.

[40] Christoph Paasch, Sébastien Barré, and et al. *Multipath TCP in the Linux Kernel*. available from http://www.multipath-tcp.org [Online; accessed 30-November-2016].

[41] Christoph Paasch and Olivier Bonaventure. “Multipath TCP”. In: *Queue* 12.2 (Feb. 2014), 40:40–40:51. ISSN: 1542-7730. DOI: 10.1145/2578508.2591369. URL: http://doi.acm.org/10.1145/2578508.2591369.

[42] Christoph Paasch et al. “Experimental Evaluation of Multipath TCP Schedulers”. In: *ACM SIGCOMM Capacity Sharing Workshop (CSWS)*. ACM, 2014.

[43] Roger Pantos. *HTTP Live Streaming*. Internet-Draft draft-pantos-http-live-streaming-19. RFC Editor, Oct. 2016.

[44] Costin Raiciu et al. “How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP”. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI’12. San Jose, CA: USENIX Association, 2012, pp. 29–29. URL: http://dl.acm.org/citation.cfm?id=2228298.2228338.

[45] Costin Raiciu, Janardhan Iyengar, Olivier Bonaventure, et al. “Recent advances in reliable transport protocols”. In: *SIGCOMM ebook on Recent Advances in Networking* (2013).

[46] Benjamin Rainer and Christian Timmerer. "Quality of Experience of Web-based Adaptive HTTP Streaming Clients in Real-World Environments Using Crowdsourcing". In: *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*. VideoNext '14. Sydney, Australia: ACM, 2014, pp. 19–24. ISBN: 978-1-4503-3281-1. DOI: 10.1145/2676652.2676656. URL: http://doi.acm.org/10.1145/2676652.2676656.

[47] Benjamin Rainer et al. "A Seamless Web Integration of Adaptive HTTP streaming". EN. In: *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. Ed. by Béatrice Pesquet-Popescu and Corneliu Burileanu. Bucharest, Romania: European Signal Processing (EURASIP) Society, Aug. 2012, pp. 1519–1523.

[48] Michael Seufert et al. "A survey on quality of experience of HTTP adaptive streaming". In: *Communications Surveys & Tutorials, IEEE* 17.1 (2015), pp. 469–492.

[49] Iraj Sodagar. "The MPEG-DASH Standard for Multimedia Streaming Over the Internet". In: *IEEE MultiMedia* 4 (2011), pp. 62–67.

[50] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. "BOLA: Near-Optimal Bitrate Adaptation for Online Videos". In: *ArXiv e-prints* (Jan. 2016). arXiv: 1601.06748 [cs.NI].

[51] Thomas Stockhammer. "Dynamic adaptive streaming over HTTP-design principles and standards". In: *Proceedings of the Second Annual ACM Conference on Multimedia Systems*. Vol. 2014. New York, USA: ACM. 2011, pp. 2–4.

[52] The Linux Foundation. *iproute2*. [Online; accessed 13-June-2016]. Nov. 2009. URL: http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2.

[53] The Linux Foundation. *netem*. [Online; accessed 13-June-2016]. Nov. 2009. URL: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[54] The Linux Kernel Organization. *ip-sysctl.txt*. [Online; accessed 2-December-2016]. Dec. 2016. URL: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt.

[55] Guibin Tian and Yong Liu. "Towards agile and smooth video adaptation in dynamic HTTP streaming". In: *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM. 2012, pp. 109–120.

[56] Christian Timmerer et al. "Quality of Experience of Adaptive HTTP Streaming in Real-World Environments". In: *IEEE Multimedia Communications Technical Committee E-Letter* (May 2015), pp. 6–9.

[57] Jiyan Wu et al. "Streaming High-Quality Mobile Video with Multipath TCP in Heterogeneous Wireless Networks". In: *IEEE Transactions on Mobile Computing* (2015).

[58] Xiaoqi Yin, Vyas Sekar, and Bruno Sinopoli. "Toward a Principled Framework to Design Dynamic Adaptive Streaming Algorithms over HTTP". In: *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. ACM. 2014, p. 9.

[59] Alex Zambelli. "IIS smooth streaming technical overview". In: *Microsoft Corporation* 3 (2009), p. 40.

[60] Xuan Kelvin Zou et al. "Can Accurate Predictions Improve Video Streaming in Cellular Networks?" In: *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM. 2015, pp. 57–62.

A | Code repositories

The code used for all parts of this thesis can be found in several git repositories.

`https://bitbucket.org/lvpeschke/dash-mptcp-thesis` contains scripts used to build and modify the infrastructure, launch tests instances, analyse and process results, and generate graphs.

`https://github.com/lvpeschke/vlc` is a fork of the VLC media player[1] available at `https://github.com/videolan/vlc`. It essentially includes additional code to capture the state of the player during playback.

`https://github.com/lvpeschke/mptcp-analysis-scripts` is a fork of the MPTCP Analysis Scripts [17] available at `https://github.com/multipath-tcp/mptcp-analysis-scripts`.

[1] VideoLAN's VLC: `https://www.videolan.org/vlc/`

B | DASH video dataset

Table B.1 gives the resolutions and bitrates, table B.2 gives the amount of segments and selected segment sizes of the Big Buck Bunny dataset [29] for 2 s and 10 s segments.

Table B.1: Big Buck Bunny dataset [29] resolutions and bitrates.

Segment length	**2 s**	**10 s**
Resolution=320x240	46.0 kbps	45.0 kbps
	89.0 kbps	88.0 kbps
	131.0 kbps	127.0 kbps
Resolution=480x360	178.0 kbps	177.0 kbps
	222.0 kbps	217.0 kbps
	263.0 kbps	253.0 kbps
	334.0 kbps	317.0 kbps
	396.0 kbps	369.0 kbps
Resolution=854x480	522.0 kbps	503.0 kbps
	595.0 kbps	569.0 kbps
Resolution=1280x720	791.0 kbps	771.0 kbps
	1.0 Mbps	987.0 kbps
	1.2 Mbps	1.2 Mbps
	1.5 Mbps	1.4 Mbps
Resolution=1920x1080	2.1 Mbps	2.1 Mbps
	2.5 Mbps	2.4 Mbps
	3.1 Mbps	2.9 Mbps
	3.5 Mbps	3.2 Mbps
	3.8 Mbps	3.5 Mbps
	4.2 Mbps	3.8 Mbps

Table B.2: Big Buck Bunny dataset [29] segment sizes.

Segment length	**2 s**	**10 s**
Total segments	299	60
Segment size Min. at lowest bitrate	3.9 kB	38.0 kB
Max. at lowest bitrate	16.0 kB	60.0 kB
Min. at highest bitrate	132.0 kB	1.3 MB
Max. at highest bitrate	2.2 MB	8.6 MB

C | Media Presentation Description

Listing 3: Big Buck Bunny, 2 seconds, simple

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- MPD file Generated with GPAC version 0.5.1-DEV-rev5379 on
  ↪ 2014-09-10T13:23:18Z-->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500000S" type="static"
  ↪ mediaPresentationDuration="PT0H9M56.46S"
  ↪ profiles="urn:mpeg:dash:profile:isoff-live:2011">
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
   <Title>dashed/BigBuckBunny_2s_simple_2014_05_09.mpd generated by GPAC</Title>
 </ProgramInformation>
 <Period duration="PT0H9M56.46S">
   <AdaptationSet segmentAlignment="true" group="1" maxWidth="480" maxHeight="360"
     ↪ maxFrameRate="24" par="4:3">
    <SegmentTemplate timescale="96"
      ↪ media="bunny_$Bandwidth$bps/BigBuckBunny_2s$Number$.m4s" startNumber="1"
      ↪ duration="192"
      ↪ initialization="bunny_$Bandwidth$bps/BigBuckBunny_2s_init.mp4" />
    <Representation id="320x240 46.0kbps" mimeType="video/mp4" codecs="avc1.42c00d"
      ↪ width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1"
      ↪ bandwidth="45652" />
    <Representation id="320x240 89.0kbps" mimeType="video/mp4" codecs="avc1.42c00d"
      ↪ width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1"
      ↪ bandwidth="89283" />
    <Representation id="320x240 131.0kbps" mimeType="video/mp4"
      ↪ codecs="avc1.42c00d" width="320" height="240" frameRate="24" sar="1:1"
      ↪ startWithSAP="1" bandwidth="131087" />
    <Representation id="480x360 178.0kbps" mimeType="video/mp4"
      ↪ codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
      ↪ startWithSAP="1" bandwidth="178351" />
    <Representation id="480x360 222.0kbps" mimeType="video/mp4"
      ↪ codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
      ↪ startWithSAP="1" bandwidth="221600" />
    <Representation id="480x360 263.0kbps" mimeType="video/mp4"
      ↪ codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
      ↪ startWithSAP="1" bandwidth="262537" />
    <Representation id="480x360 334.0kbps" mimeType="video/mp4"
      ↪ codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
      ↪ startWithSAP="1" bandwidth="334349" />
    <Representation id="480x360 396.0kbps" mimeType="video/mp4"
      ↪ codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
      ↪ startWithSAP="1" bandwidth="396126" />
```

```
18        <Representation id="854x480 522.0kbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c01e" width="854" height="480" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="522286" />
19        <Representation id="854x480 595.0kbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c01e" width="854" height="480" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="595491" />
20        <Representation id="1280x720 791.0kbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="791182" />
21        <Representation id="1280x720 1.0Mbps" mimeType="video/mp4" codecs="avc1.42c01f"
            ↪  width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1"
            ↪  bandwidth="1032682" />
22        <Representation id="1280x720 1.2Mbps" mimeType="video/mp4" codecs="avc1.42c01f"
            ↪  width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1"
            ↪  bandwidth="1244778" />
23        <Representation id="1280x720 1.5Mbps" mimeType="video/mp4" codecs="avc1.42c01f"
            ↪  width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1"
            ↪  bandwidth="1546902" />
24        <Representation id="1920x1080 2.1Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="2133691" />
25        <Representation id="1920x1080 2.5Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="2484135" />
26        <Representation id="1920x1080 3.1Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="3078587" />
27        <Representation id="1920x1080 3.5Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="3526922" />
28        <Representation id="1920x1080 3.8Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="3840360" />
29        <Representation id="1920x1080 4.2Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="4219897" />
30      </AdaptationSet>
31    </Period>
32  </MPD>
```

Listing 4: Big Buck Bunny, 10 seconds, simple

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- MPD file Generated with GPAC version 0.5.1-DEV-rev5379  on
  ↪  2014-09-10T13:40:30Z-->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500000S" type="static"
  ↪  mediaPresentationDuration="PT0H9M56.46S"
  ↪  profiles="urn:mpeg:dash:profile:isoff-live:2011">
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
   <Title>dashed/BigBuckBunny_10s_simple_2014_05_09.mpd generated by GPAC</Title>
 </ProgramInformation>
 <Period duration="PT0H9M56.46S">
   <AdaptationSet segmentAlignment="true" group="1" maxWidth="480" maxHeight="360"
     ↪  maxFrameRate="24" par="4:3">
     <SegmentTemplate timescale="96"
       ↪  media="bunny_$Bandwidth$bps/BigBuckBunny_10s$Number$.m4s" startNumber="1"
       ↪  duration="960"
       ↪  initialization="bunny_$Bandwidth$bps/BigBuckBunny_10s_init.mp4" />
     <Representation id="320x240 45.0kbps" mimeType="video/mp4" codecs="avc1.42c00d"
       ↪  width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1"
       ↪  bandwidth="45373" />
     <Representation id="320x240 88.0kbps" mimeType="video/mp4" codecs="avc1.42c00d"
       ↪  width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1"
       ↪  bandwidth="88482" />
     <Representation id="320x240 127.0kbps" mimeType="video/mp4"
       ↪  codecs="avc1.42c00d" width="320" height="240" frameRate="24" sar="1:1"
       ↪  startWithSAP="1" bandwidth="127412" />
     <Representation id="480x360 177.0kbps" mimeType="video/mp4"
       ↪  codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
       ↪  startWithSAP="1" bandwidth="176780" />
     <Representation id="480x360 217.0kbps" mimeType="video/mp4"
       ↪  codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
       ↪  startWithSAP="1" bandwidth="216536" />
     <Representation id="480x360 253.0kbps" mimeType="video/mp4"
       ↪  codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
       ↪  startWithSAP="1" bandwidth="252988" />
     <Representation id="480x360 317.0kbps" mimeType="video/mp4"
       ↪  codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
       ↪  startWithSAP="1" bandwidth="317328" />
     <Representation id="480x360 369.0kbps" mimeType="video/mp4"
       ↪  codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1"
       ↪  startWithSAP="1" bandwidth="368912" />
     <Representation id="854x480 503.0kbps" mimeType="video/mp4"
       ↪  codecs="avc1.42c01e" width="854" height="480" frameRate="24" sar="1:1"
       ↪  startWithSAP="1" bandwidth="503270" />
     <Representation id="854x480 569.0kbps" mimeType="video/mp4"
       ↪  codecs="avc1.42c01e" width="854" height="480" frameRate="24" sar="1:1"
       ↪  startWithSAP="1" bandwidth="568500" />
     <Representation id="1280x720 771.0kbps" mimeType="video/mp4"
       ↪  codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1"
       ↪  startWithSAP="1" bandwidth="771359" />
```

```
21        <Representation id="1280x720 987.0kbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="987061" />
22        <Representation id="1280x720 1.2Mbps" mimeType="video/mp4" codecs="avc1.42c01f"
            ↪  width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1"
            ↪  bandwidth="1174238" />
23        <Representation id="1280x720 1.4Mbps" mimeType="video/mp4" codecs="avc1.42c01f"
            ↪  width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1"
            ↪  bandwidth="1431232" />
24        <Representation id="1920x1080 2.1Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="2070985" />
25        <Representation id="1920x1080 2.4Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="2384387" />
26        <Representation id="1920x1080 2.9Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="2884382" />
27        <Representation id="1920x1080 3.2Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="3245900" />
28        <Representation id="1920x1080 3.5Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="3493765" />
29        <Representation id="1920x1080 3.8Mbps" mimeType="video/mp4"
            ↪  codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1"
            ↪  startWithSAP="1" bandwidth="3792491" />
30      </AdaptationSet>
31    </Period>
32  </MPD>
```

D | VM configuration

Listing 5: Shortened client VM configuration.

```
Name:            NewClient
Guest OS:        Ubuntu (64-bit)
Memory size:     1024MB
VRAM size:       12MB
CPU exec cap:    100%
Number of CPUs:  2
NIC 1:           MAC: 080027C6B4CE, Attachment: NAT,
                  Cable connected: on, Trace: off (file: none), Type: 82540EM,
                  Reported speed: 0 Mbps, Boot priority: 0, Promisc Policy: deny,
                  Bandwidth group: none
NIC 1 Settings:  MTU: 0, Socket (send: 64, receive: 64),
                  TCP Window (send:64, receive: 64)
NIC 1 Rule(0):   name = ssh-nat, protocol = tcp,
                  host ip = , host port = 3024, guest ip = , guest port = 22
NIC 2:           MAC: 0800270E2572, Attachment: Bridged Interface 'tapc1',
                  Cable connected: on, Trace: off (file: none), Type: 82540EM,
                  Reported speed: 0 Mbps, Boot priority: 0, Promisc Policy: deny,
                  Bandwidth group: none
NIC 3:           MAC: 080027064001, Attachment: Bridged Interface 'tapc2',
                  Cable connected: on, Trace: off (file: none), Type: 82540EM,
                  Reported speed: 0 Mbps, Boot priority: 0, Promisc Policy: deny,
                  Bandwidth group: none
Audio:           enabled (Driver: ALSA, Controller: AC97, Codec: AD1980)
```

Listing 6: Shortened server VM configuration.

```
Name:            NewServer
Guest OS:        Ubuntu (64-bit)
Memory size:     1024MB
VRAM size:       12MB
CPU exec cap:    100%
Number of CPUs:  2
NIC 1:           MAC: 08002796E151, Attachment: NAT,
                  Cable connected: on, Trace: off (file: none), Type: 82540EM,
                  Reported speed: 0 Mbps, Boot priority: 0, Promisc Policy: deny,
                  Bandwidth group: none
NIC 1 Settings:  MTU: 0, Socket (send: 64, receive: 64),
                  TCP Window (send:64, receive: 64)
NIC 1 Rule(0):   name = ssh-nat, protocol = tcp,
                  host ip = , host port = 3042, guest ip = , guest port = 22
NIC 2:           MAC: 0800271721CC, Attachment: Bridged Interface 'taps1',
                  Cable connected: on, Trace: off (file: none), Type: 82540EM,
                  Reported speed: 0 Mbps, Boot priority: 0, Promisc Policy: deny,
                  Bandwidth group: none
NIC 3:           MAC: 080027610131, Attachment: Bridged Interface 'taps2',
                  Cable connected: on, Trace: off (file: none), Type: 82540EM,
                  Reported speed: 0 Mbps, Boot priority: 0, Promisc Policy: deny,
                  Bandwidth group: none
Audio:           enabled (Driver: ALSA, Controller: AC97, Codec: AD1980)
```

E | Network performance

Tables E.1 to E.6 give the detailed ping and iperf results for two network test runs between the client and server VMs. MPTCP is disabled, the congestion control is set to New Reno. The configured delay and bandwidth are added with tc NetEm on the host.

Table E.1: ping results (RTT values).

Delay (ms)	**Bandwidth (Mbps)**	**min (ms)**	**avg (ms)**	**max (ms)**	**mdev (ms)**	**avg error margin (%)**
25	1	26.933	26.996	27.085	0.186	7.984
25	10	25.512	25.551	25.638	0.217	2.204
25	20	25.401	25.475	25.613	0.121	1.9
25	50	25.385	25.463	25.688	0.192	1.852
25	100	25.374	25.455	25.645	0.076	1.82
50	1	51.908	51.966	52.044	0.291	3.932
50	10	50.511	50.584	50.648	0.148	1.168
50	20	50.435	50.506	50.625	0.305	1.012
50	50	50.376	50.425	50.504	0.107	0.85
50	100	50.355	50.41	50.455	0.031	0.82
100	1	101.916	102.002	102.176	0.41	2.002
100	10	100.533	100.599	100.767	0.383	0.599
100	20	100.415	100.513	100.614	0.406	0.513
100	50	100.387	100.459	100.563	0.428	0.459
100	100	100.394	100.452	100.549	0.249	0.452
150	1	151.931	152.002	152.103	0.352	1.335
150	10	150.528	150.576	150.692	0.304	0.384
150	20	150.456	150.495	150.617	0.249	0.33
150	50	150.426	150.462	150.553	0.303	0.308
150	100	150.386	150.459	150.512	0.46	0.306

Table E.2: ping results (RTT values).

Delay (ms)	**Bandwidth (Mbps)**	**min (ms)**	**avg (ms)**	**max (ms)**	**mdev (ms)**	**avg error margin (%)**
25	1	26.921	26.991	27.08	0.224	7.964
25	10	25.522	25.595	25.79	0.107	2.38
25	20	25.437	25.521	25.735	0.109	2.084
25	50	25.384	25.448	25.492	0.105	1.792
25	100	25.386	25.449	25.515	0.108	1.796
50	1	51.91	51.995	52.115	0.117	3.99
50	10	50.474	50.546	50.653	0.208	1.092
50	20	50.435	50.511	50.62	0.209	1.022
50	50	50.379	50.452	50.544	0.207	0.904
50	100	50.378	50.42	50.506	0.303	0.84
100	1	101.918	101.965	102.033	0.147	1.965
100	10	100.536	100.607	100.72	0.253	0.607
100	20	100.448	100.514	100.674	0.352	0.514
100	50	100.378	100.467	100.601	0.158	0.467
100	100	100.382	100.797	103.937	1.076	0.797
150	1	151.891	151.957	152.16	0.356	1.305
150	10	150.478	150.567	150.652	0.305	0.378
150	20	150.411	150.463	150.53	0.461	0.309
150	50	150.398	150.483	150.62	0.308	0.322
150	100	150.375	150.434	150.561	0.392	0.289

Table E.3: iperf UDP results. target is the bandwidth at which the iperf client tries to send to the server, as a percentage of the configured bandwidth.

Delay (ms)	**Bandwidth (Mbps)**	**Target (%)**	**Throughput (Mbps)**	**Throughput (%)**
25	1	100	0.972	97.2
25	1	200	0.969	96.9
25	10	100	9.72	97.2
25	10	200	9.72	97.2
25	20	100	19.4	97
25	20	200	19.1	95.5
25	50	100	48.4	96.8
25	50	200	48.4	96.8
25	100	100	96.2	96.2
25	100	200	97.2	97.2
50	1	100	0.972	97.2
50	1	200	0.968	96.8
50	10	100	9.72	97.2
50	10	200	9.56	95.6
50	20	100	19.4	97
50	20	200	19.4	97
50	50	100	48.6	97.2
50	50	200	47.7	95.4
50	100	100	96.3	96.3
50	100	200	97.1	97.1
100	1	100	0.972	97.2
100	1	200	0.971	97.1
100	10	100	9.72	97.2
100	10	200	9.72	97.2
100	20	100	19.4	97
100	20	200	19.4	97
100	50	100	48.6	97.2
100	50	200	48.6	97.2
100	100	100	96.6	96.6
100	100	200	94.6	94.6
150	1	100	0.972	97.2
150	1	200	0.957	95.7
150	10	100	9.72	97.2
150	10	200	9.57	95.7
150	20	100	19.4	97
150	20	200	19.4	97
150	50	100	48.6	97.2
150	50	200	47.7	95.4
150	100	100	96.4	96.4
150	100	200	97.2	97.2

Table E.4: iperf UDP results. target is the bandwidth at which the iperf client tries to send to the server, as a percentage of the configured bandwidth.

Delay (ms)	**Bandwidth (Mbps)**	**Target (%)**	**Throughput (Mbps)**	**Throughput (%)**
25	1	100	0.972	97.2
25	1	200	0.956	95.6
25	10	100	9.72	97.2
25	10	200	9.72	97.2
25	20	100	19.4	97
25	20	200	19.4	97
25	50	100	48.6	97.2
25	50	200	48.6	97.2
25	100	100	96.5	96.5
25	100	200	97.2	97.2
50	1	100	0.972	97.2
50	1	200	0.972	97.2
50	10	100	9.72	97.2
50	10	200	9.69	96.9
50	20	100	19.4	97
50	20	200	19.4	97
50	50	100	48.6	97.2
50	50	200	48.5	97
50	100	100	96.9	96.9
50	100	200	97.2	97.2
100	1	100	0.972	97.2
100	1	200	0.971	97.1
100	10	100	9.72	97.2
100	10	200	9.4	94
100	20	100	19.4	97
100	20	200	19.4	97
100	50	100	48.6	97.2
100	50	200	48.5	97
100	100	100	96.6	96.6
100	100	200	97.2	97.2
150	1	100	0.972	97.2
150	1	200	0.972	97.2
150	10	100	9.72	97.2
150	10	200	9.72	97.2
150	20	100	19.4	97
150	20	200	19.4	97
150	50	100	48.6	97.2
150	50	200	48.5	97
150	100	100	96.8	96.8
150	100	200	97.2	97.2

Table E.5: iperf TCP results. The reported throughput is measured at the receiving host.

Delay (ms)	**Bandwidth (Mbps)**	**Throughput (Mbps)**	**Throughput (%)**
25	1	0.788	78.8
25	10	8.89	88.9
25	20	17.8	89
25	50	43.3	86.6
25	100	80.4	80.4
50	1	0.768	76.8
50	10	9.12	91.2
50	20	18.2	91
50	50	45.1	90.2
50	100	83.5	83.5
100	1	0.743	74.3
100	10	9.15	91.5
100	20	17.6	88
100	50	43	86
100	100	76.8	76.8
150	1	0.734	73.4
150	10	9.13	91.3
150	20	17.6	88
150	50	36.9	73.8
150	100	75.1	75.1

Table E.6: iperf TCP results. The reported throughput is measured at the receiving host.

Delay (ms)	**Bandwidth (Mbps)**	**Throughput (Mbps)**	**Throughput (%)**
25	1	0.788	78.8
25	10	9.01	90.1
25	20	17.8	89
25	50	43.1	86.2
25	100	81.2	81.2
50	1	0.773	77.3
50	10	9.1	91
50	20	18.2	91
50	50	43.6	87.2
50	100	84.7	84.7
100	1	0.746	74.6
100	10	9.15	91.5
100	20	17.7	88.5
100	50	41.9	83.8
100	100	80.1	80.1
150	1	0.735	73.5
150	10	9	90
150	20	16.8	84
150	50	35.6	71.2
150	100	74.9	74.9

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**