AGH

AGH University of Science and Technology

Implementation of Network Distortions Simulating Quality Changes Observed by YouTube Users

Implementacja zniekształceń sieciowych symulujacych zmiany jakości obserwowane przez użytkowników aplikacji YouTube

Mykyta Muravytskyi

Supervisor:

PhD, Lucjan Janowski

Krakow, 2021

Contents

1 Introduction

Streaming of multimedia content is becoming more popular than ever, its growth has reached an astonishing 266% over the past three years [16]. And is expected to grow even further by approximately 21% every year [15]. The vast majority of this growth is associated with streaming services such as YouTube, Netflix, or Hulu, expanding on a global scale and acquiring more and more clients throughout the world [18].

The large amount of data associated with multimedia streaming needs to be efficiently delivered to a constantly growing number of clients over networks of unstable conditions. To address these challenges, streaming technologies have adopted both highly flexible and scalable approach of Dynamic Adaptive Streaming over HTTP (DASH). The main purpose of which is to dynamically adapt the video stream to the network conditions.

There are a few aspects to this. The first is the progressive download, where each video is encoded in small atomic segments called 'chunks' which are delivered separately on request. Secondly, adaptability, videos are encoded at different bitrates, quality levels, and resolutions using different codecs. The client-side player is responsible for deciding chunk of which quality to request next. This approach is known as *client-driven rate adaptation* and it allows for uninterruptible playback even if network bandwidth changes in time. The last aspect for streaming over HTTP is the Hypertext Transfer Protocol itself - infrastructures of modern networks have adapted to support a tremendous scale of HTTP usage, thus making it a perfect fit for large-scale multimedia streaming.

Nevertheless, this technology is far from ideal and has its drawbacks related to both infrastructure and end-user experience. Whereas constantly growing demand for multimedia streaming poses certain difficulties onto network infrastructure, end-users expect stable high quality and low delays. For streaming providers serving many users globally, providing good Quality of Experience (QoE) is no trifling matter. They are interested in mediating a perfect balance between end-user QoE and operational costs.

The International Telecommunication Union Telecommunication Standardization Sector (ITU-T) defines Quality of Experience as *"the overall acceptability of an application or service, as perceived subjectively by the end-user"* [22]. This concept reaches far beyond the technical side of streaming and includes aspects of human psychology such as motivation, engagement, affective states, and viewing conditions. Another definition of QoE from a profound paper [12] defines Quality of Experience as *"the degree of delight or annoyance of a person whose experiencing involves an application, service or system. It results from the person's evaluation of the fulfillment of his or her expectations and needs concerning the utility or enjoyment in the light of the person's context, personality and current state."* By its very nature, QoE is exceptionally hard to measure and verify because of its human-centric nature, therefore more approachable objective metrics are commonly used as a substitute to estimate QoE. The relation between Objective and Subjective qualities remains unclear and attracts lots of research to the field [34]. Being able to

accurately estimate QoE based on measurable objective metrics would be an essential tool for efficient streaming.

As an intermediary step between Quality of Service (QoS) (network metrics such as bandwidth, latency, and jitter) and QoE resides objective quality. It is undoubtedly influenced by QoS factors and plays a crucial role in the human experience [11]. Yet it could be easily monitored and supervised, which might be troublesome in the case of subjective metrics.

1.1 Work objectives

The goal of this work is to implement and better understand how network distortions such as bandwidth fluctuations impact video quality. In particular, video quality as an objective quality metric is studied as a function of available bandwidth. YouTube being the most popular and known streaming service, was selected as a testing platform. The goals mentioned above imply different stages of the following work. The first is an analysis of a streaming adaptation algorithm in the implementation of YouTube, to better understand how its design helps to maximise the QoE. Second stage - a study of video quality changes while simulating different network conditions and bandwidths.

The results of this work will be used as a part of a bigger project named "Towards Better Understanding of Factors Influencing QoE" [40] which aim is to better understand the Quality of Experience.

1.2 Work layout

The rest of this work is organised in the following order:

First section contains an introduction to this work, states the problem, and defines main objectives.

Second section is introducing State of the Art of the Adaptive Streaming over HTTP, video compression, and rate adaptation. At the end of this, section there is an overview of existing publications and related work in the field.

Third section explores a few approaches for the problem and presents a plan for the tests. In the second part it covers methodologies used for gathering data and its initial processing.

Fourth section introduces three test cases and provides an overview of the results.

Final fifth section provides a summary of the whole work and explains how achieved results can be used for QoE measurements and evaluation.

2 State of the Art

This state-of-the-art section gives detailed information on video compression, multimedia streaming, and rate adaptation mechanisms. It explains the essential aspects in order to help understand the current state of streaming technology and its principal components.

2.1 Introduction to video compression

In its essence, any video is a set of sequentially played images. It was treated like that for a long time, but this definition cares one significant problem. It is practically irrational to transfer videos as a set of images. Consider a video with 1920x1080 resolution at 60 frames per second. A single minute of such a video will use up around 22GB which is about 3Gb per second. Therefore, efficient and fast compression is essential for any multimedia streaming.

In 1993, the Moving Picture Experts Group (MPEG) released the MPEG-1 standard. Three main pillars were at the foundation of the MPEG compression: temporal redundancies as most of the consecutive frames differ very little from one another; spacial redundancies meaning, images tend to have repetitive areas e.g. wall in the background during Zoom call; and imperfections of the human eye e.g. subtle differences in colors and textures that human eye cannot perceive.

MPEG's core idea was based on encoding not images themselves but the differences between the consecutive frames with additional usage of image compression techniques. It is remarkably simple: get keyframes (encoded directly from the source) with a specified interval and based on those frames predict gaps in-between them. Those self-contained frames are called *Intra frames* or *I-frames*, the frames that are predicted based on the previous frames are named *Predicted frames* or *P-frames*; *Bidirectionally predicted frames* or *B-frames* use both previous and posterior frames for prediction. To achieve an even greater compression ratio, I-frames are additionally compressed to minimize spatial redundancy. Figure 1 presents an example of two keyframes (I-frame), one P-frame and one B-frame, and their relations.

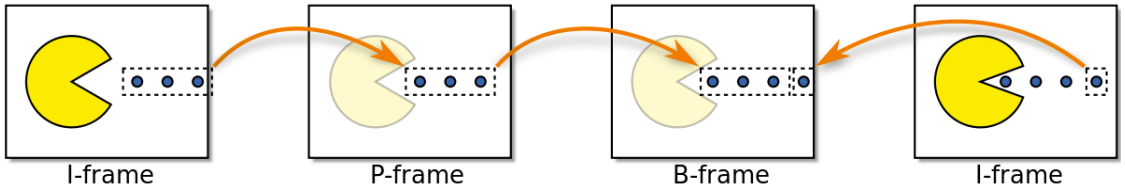

Figure 1: An example of video frames: two keyframes (I-frame), one forward-predicted frame (P-frame) and one bidirectionally predicted frame (B-frame) [10]

Keyframes carry the most information and take up the most space compared to other frame types because they cannot be predicted and need to be encoded from the source video. Understanding this concept explains why seeking or reading a random frame in a video is not a fast and straightforward process, additional frames need to be decoded in order to get the sought one.

Additionally, to make the predictions easier and more precise, a frame is split up into segments of different sizes depending on the detailedness of the region, this concept is known as *block motion*

compensation. Motion vectors are calculated for each segment, showing how its position has changed in-between frames, thus only parameters of the motion vector can be stored, not blocks themselves. This approach causes some difficulties when it comes to decompression. In order to correctly recreate P-frame, Intra frames are required, therefore if I-frame is corrupted it affects related P-frames. In the case of streaming, when a video is delivered in chunks, those chunks are required to be atomic, meaning a chunk should contain all the frames needed for the correct chunk decompression and should not depend on any other chunks which may be delivered later or even get lost during the transmission.

2.1.1 CODEC's

A video CODEC (Compressor/Decompressor) is a dedicated device or a piece of software used for encoding and decoding a video stream. In order to make a fully functional CODEC, a few additional steps need to be added to the process described above. The generic algorithm starts with splitting frames up into blocks, similar to segments from MPEG-1, then calculating predictions based either on the current frame (intraframe prediction) or other already encoded frames (interframe prediction). The following steps involve a transform and quantisation of so-called residual i.e. predictions subtracted from the current block. After quantisation only the coefficients that carry the most information remain, thus resulting in a lossy yet effective compression. Removal of components (e.g. high frequencies in Discrete Cosine Transform) may not be even noticeable due to the imperfections of the human visual system. After the data is quantised it is combined with the information essential to enable the decoder to re-create the predictions and then gets converted into a binary sequence. To reconstruct the original video, the decoder performs the same operations in the opposite order. Many of the modern codecs adopt this approach but may differ in implementation.

Some of the presented above operations may be computationally expensive, therefore codecs e.g. H.264 may be accelerated using dedicated hardware.

2.1.2 Important conclusions

There are two major factors when it comes to codecs' performance, those are quality and size after the compression. The trade-off between those two is such that if one gets better the second inevitably gets worse. To put it into other words, the closer the encoded video to the original is, the larger the compressed file will be. Both end quality and size are affected by the source video features and its format. For instance, dynamic video has significantly more differences between consecutive frames, thus resulting in larger intermediate frames and larger final file in the end. The same goes for detaildeness: compression rate is to be sacrificed to keep the quality of the details. It means that two videos of the same duration and quality parameters after compression may differ significantly in size.

2.2 Introduction to multimedia streaming

Tools that allow users to watch a video stream in a browser or an application can be divided into HTTP and non-HTTP categories. Meaning the usage of HTTP as a protocol for media transportation. The following section gives an overview of both of these approaches.

2.2.1 Non-HTTP Streaming

Non-HTTP Streaming suggests using a dedicated protocol for multimedia streaming. This section gives an overview of WebRTC as an example of streaming without HTTP [19] .

WebRTC is a real-time multimedia exchange technology that allows for peer-to-peer video and audio connections. It utilizes a signaling server as a mediator to establish a peer-to-peer connection. It doesn't perform data transfer on its own but utilizes SRTP (Secure Real-time Transport Protocol) [31] as a transport layer. SRTP is UDP-based and stateful; it requires session establishment for data transmission. WebRTC implements complex feedback signaling about the channel quality, at every moment of time bitrate is calculated, packet losses measured, decisions on their redirecting taken, and audio-to-video synchronization is estimated. All of the calculations required make WebRTC a resource-intensive and complex protocol, however capable of low-latency high-quality transmissions. Those advantages have made it a popular choice for real-time voice communication such as Discord. However, an additional problem arises from UDP usage i.e. many enterprise firewalls are configured to simply block UDP, thus disallowing WebRTC in the network [30].

2.2.2 Streaming over HTTP

The Hypertext Transfer Protocol (HTTP) is considered to be the foundation of the World Wide Web. Its usage is so common, that network architecture has adapted to fulfill the requirements of HTTP.

The concept of utilizing HTTP as a transport for multimedia streaming is supposed to utilize already existing infrastructure. Its elements are nothing new for HTTP services.

Conceptually any HTTP streaming architecture consists of three main parts: the server-side component, the client player software, and the distribution component. The server is responsible for digital encoding of the inputs to desired formats and qualities, segmenting the video into chunks, and composing an index file (also called manifest or description) [38]. So when a video is uploaded to the server it is being processed in order to prepare it for distribution. The client player is responsible for fetching an index file of the desired stream, determining the appropriate media to request based on it, fetching the media, decoding it, and finally presenting content to the user. User requests are handled by the distribution system i.e. an HTTP server or an HTTP-based web cache that serves both index files and content that has been encoded by the server.

2.2.3 Dynamic Adaptive Streaming over HTTP (MPEG-DASH)

MPEG-DASH (Motion Picture Experts Group-Dynamic Adaptive Streaming over HTTP (Hypertext Transfer Protocol)) has been standardised in [20]. According to the standard, a video file is being split up into atomic parts of equal duration called chunks. Chunks are encoded using different codecs and compression ratios, which allows shifting dynamically between qualities during the playtime if network conditions change. The player software may start requesting chunks of lower quality if bandwidth gets limited and vise versa. The user gets to know what qualities are available and their location by receiving so-called Media Presentation Description (MPD) from a content server. MPD is a hierarchically structured XML file that contains information about media as a whole, about available chunks, and their qualities. Fig. 2 shows a hierarchical structure of an MPD file.

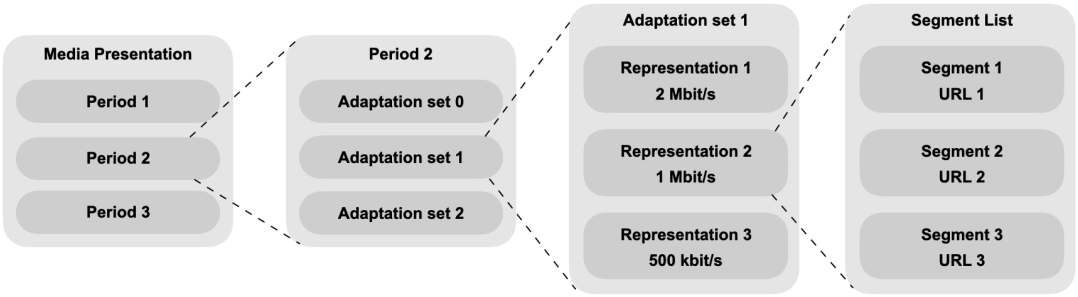

Figure 2: The hierarchy of MPEG-DASH Media Presentation Description file.

At the very top of the MPD file hierarchy, the *Media Presentation* section is located. It contains information about all the different types of media in the content e.g. video, audio, and captions. At the lower level, there are *Periods* - logical segments of the video, which make navigation through the video easier for the user. Periods level consists of *Adaptation Sets* which typically contain multiple *Representations* for each codec, quality, or bandwidth available. Separate *Adaptation Sets* are used for captions, audio, and video. At the bottom-most level, *Adaptation Set* consists of *Segments* i.e. elements containing the information required to construct the actual URL for downloading the content.

From the user's perspective, the following happens when a video is requested. First things first, an index file for the requested video is fetched and parsed. Thus player software gets to know general information about the requested content and where to find it. Next, the player selects an appropriate representation and starts downloading the content by fetching the segments. Content is downloaded progressively, meaning segments are requested in advance and stored in a buffer until played, so if the available network throughput drops, playtime would remain unaffected, at least until the buffer runs empty.

The way the player selects which representation to request is defined by *rate adaptation algorithms*, which are described in more detail in the following section 2.2.5.

2.2.4 HTTP Live Streaming (HLS)

HTTP Live Streaming (HLS) [33] is one of the most widely-used multimedia streaming protocols, it has been designed by Apple in 2009. In November of 2021, a draft of the 2nd edition [32] was released and is supposed to obsolete the previous edition.

The server-side of the streaming in HLS is quite similar to the one described in DASH. Content gets split up into integral pieces, encoded and stored in different qualities. Differences become obvious when it comes to an index file. Primary *Master Playlist* (in [32] is called *Multivariant Playlist*) is only fetched once at the start of the session and is equivalent to an MPD in DASH [20]. Yet Master Playlist contains information only on available content variants of a particular bandwidth and encoding and Uniform Resource Identifiers (URIs) to separate *Media playlists* for each of the options and not the segments themselves. Media playlists are fetched on request and contain the information required to download content segments.

Both Master and Media playlists are M3U format files which could either have URIs to other resources or tags. Fig. 3 and fig. 4 show an example of master and media playlists.

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=150000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/low/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=240000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/lo_mid/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=440000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/hi_mid/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=640000,RESOLUTION=640x360,CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/high/index.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=64000,CODECS="mp4a.40.5"
http://example.com/audio/index.m3u8
```

Figure 3: An HLS Master playlist example.

```
#EXTM3U
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:4
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
http://example.com/movie1/fileSequenceA.ts
#EXTINF:10.0,
http://example.com/movie1/fileSequenceB.ts
#EXTINF:10.0,
http://example.com/movie1/fileSequenceC.ts
#EXTINF:9.0,
http://example.com/movie1/fileSequenceD.ts
#EXT-X-ENDLIST
```

Figure 4: An HLS Media playlist example.

Every playlist starts with an `EXTM3U` tag, which allows identifying a playlist file. In the case of the *Master playlist* following is a sequence of `EXT-X-STREAM-INF` tags and URIs. `EXT-X-STREAM-INF` contains parameters such as available codec, bandwidth, frame rate, and resolution. URIs are pointing to the locations of corresponding *Media playlists*.

2.2.5 Rate adaptation algorithms

Studies such as [44] have shown that video streaming clients experience highly variable network conditions. The variation can be explained by a number of reasons, such as global network congestions, interference in the wireless access medium, TCP transmission rate drops, or content server overload.

When it comes to non-adaptive streaming, such network distortions may lead to video freezing in the middle of the playback as the rate at which the video is being played exceeds the rate at which the video is downloaded. Playback freezing of that kind is called *stalling* or *rebuffering* and plays a key role in how video quality is perceived [24]. Therefore rate adaptation algorithms are essential to adapt the video stream to the available network resources and minimise stalling and rebuffering. The general purpose of such an algorithm is to select the best content representation based on the network conditions the player operates in [27]. In this context "best" means a quality level that guarantees the highest QoE.

Proprietary implementations of adaptation algorithms by streaming provides such as YouTube or Netflix are not available for the public and can only be empirically analysed and tested (see section 4.1).

In this section, a very basic rate adaptation algorithm is described based on [44] and [27], however, the concepts found in this solution can be applied to the commercial-scale algorithms (see section 4.1).

In order to minimise the impact of network bandwidth fluctuations on the playtime, video segments are downloaded beforehand and stored in a client's buffer, then played from it. The buffer gets refilled when video chunks are downloaded, and steadily emptied during the playback. The measure of buffer's fullness is *Buffer capacity* or *Buffer health*; it is measured in second of a video residing in the buffer. Buffer fills up when network conditions are amicable, and empties when bandwidth drops, thus allowing to minimise the impact of variable network conditions. One of the concerns when it comes to buffer usage is the potential of wasting buffered video. As [17] shows, users rarely watch the entire video, the average view duration at YouTube is 50-60% of the total video length. So when a user decides to stop watching the video and switches to something else, all the buffered video gets wasted. This puts an unnecessary load on both the content delivery and the client's networks. To fight this, a limitation is implemented within a buffer, which buffer capacity cannot exceed. This solution helps to minimise the potential wasting of network resources. Fig. 5 presents a scheme of data flow filling and leaking from a buffer.

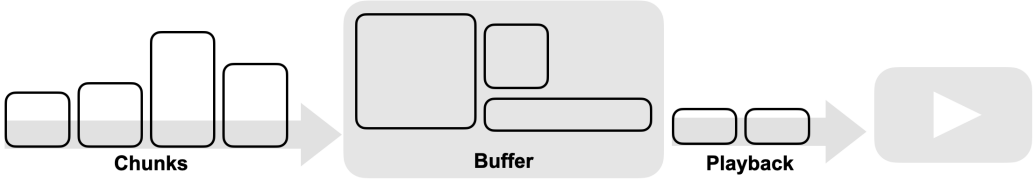

Figure 5: Buffer flow

It is paramount to understand that as a software application a player has no knowledge about the state of the network. Therefore the decision on which quality to request next can only be based on the metrics player has generated itself. The most fundamental metric used for quality selection is the available bandwidth. A player can estimate it by dividing segment size by the time it took to download this segment. This value is used as a reference point and is compared to bandwidth parameters from the index file e.g. see figure 4. To prevent the edge case when bandwidth fluctuations cause constant switches between the qualities, a correction to the estimation is applied, basically underestimating the bandwidth. This simple approach has one significant drawback i.e. if network variations are strong the algorithm will frequently switch between quality levels, emptying the buffer and disturbing the playback.

The bandwidth estimation is recalculated on every chunk and if it exceeds the threshold the switch-up or switch-down occurs i.e. segments of higher or lower quality are starting to be requested. In the case of a switch-up, this inevitably leads to an unused buffered video of lower quality from before the switch. Therefore player may implement the conservative strategy for the switch-ups: take buffer fullness as an additional parameter, thus lowering the probability of a quality switch when a buffer is filled up. For the switch-down, the situation is quite the opposite, buffered higher quality video is not wasted and is shown to the user. In both cases to transition smoothly from one quality to another, the player starts fetching segments of another quality in advance.

Overestimation of the capacity will lead to buffer running empty and as a result to the rebuffering, meaning playback will be stopped until the buffer refills. If the player underestimates the capacity, the potential of better quality is being wasted which is not desired from the perspective of the user experience.

According to the suggested above, rate adaptation algorithms may utilize a combination of parameters such as bandwidth estimation or a buffer level. The buffer-based approach from [44] suggests using a buffer growth rate to make a decision for a quality switch. Using buffer as the main metric results in a more conservative, yet more stable playback. For instance, an extra reservoir is created within the buffer to absorb variations. The player maintains its buffer level above the reservoir and only if the level is higher, quality up-switches are possible.

The described approach faces difficulties during a startup phase e.g., at the beginning of playback or after jumping to a new point. During which the buffer is empty and there is no bandwidth estimation to base upon. A generic algorithm would follow its usual behaviour and start with the lowest quality and step by step switch up to the optimal one. Yet if we assume, that the reservoir condition has to be met to switch up, then it will significantly slow down the process and lead to requesting unnecessary content that is going to be dropped on the next quality up-switch. Therefore during the startup phase, simple capacity estimation is used, allowing for more aggressive and even multiple-level switches. The startup phase lasts about 60 seconds, after which the algorithm has enough information to continue switching based on more stable metrics.

The algorithm described above is a very simple one and has its drawbacks such as no protection

against short-time switches, lack of buffering limit, and different chunk sizes. Commercially used algorithms such as YouTube's one are increasingly more complex and put focus on improving Quality of Experience, applying dynamic requesting rates, and even utilizing Machine Learning for bandwidth predictions [37].

2.3 Streaming on YouTube

At the beginning of the playback, when a user selects a video to watch, an index file is fetched for the desired video. The file is only requested once before the playback. Details of this request and the file itself could be inspected using HAR tracing e.g. Networking tools in Google Chrome browser (sought file is called `player` and is in JSON format [1]). Code below presents an example of how this file is structured:

```
"body": {
    "responseContext": {...},
    "trackingParams" : "CAAQu2kiEwipsIDa6ff0AhXGAIkKHZSvAWY=",
    "playabilityStatus": {...},
    "streamingData": {...},
    "heartbeatParams": {...},
    "playbackTracking": {...},
    "videoDetails": {...},
    "annotations": {...},
    "playerConfig": {...},
    "storyboards": {...},
    "microformat": {...},
    "attestation": {...},
    "endscreen": {...},
    "frameworkUpdates": {...}
}
```

For the scope of this work only few of the fields are relevant and related to streaming.

`videoDetails` contains general-purpose data about the video e.g. title, description, thumbnail, and video duration.

```
"videoDetils": {
    "videoId": "K78jqx9fx2I",
    "title": "Jamie Metzl: Lab Leak Theory | Lex Fridman  Podcast #247",
    "lengthSeconds": "17709",
    "keywords": [...],
    "channelId": "UCSHZKyawb77ixDdsGog4iWA",
    "isOwnerViewing": false,
```

```
        "shortDescription": "...",
        "isCrawlable": true,
        "thumbnail": {...},
        "allowRatings": true,
        "viewCount": "267215",
        "author": "Lex Fridman",
        "isPrivate": false,
        "isUnpluggedCorpus": false,
        "isLiveContent": false
    }
```


`playbackTracking` - a set of values and URLs, that player uses to send playtime statistics e.g. Quality of Experience metrics.


```
    "playbackTracking": {
        "videostatsPlaybackUrl": {...},
        "videostatsDelayplayUrl": {...},
        "videostatsWatchtimeUrl": {...},
        "ptrackingUrl": {...},
        "qoeUrl": {...},
        "videostatsScheduledFlushWalltimeSeconds": [...],
        "videostatsDefaultFlushIntervalSeconds": 40
    }
```


`playerConfig` contains player configurations parameters such as lower and upper buffer boundaries (15 and 120 seconds accordingly), loudness levels, and maximal bitrate for the content.


```
    "playerConfig": {
        "audioConfig": {
          "loudnessDb": -2,
          "perceptualLoudnessDb": -16,
          "enablePerFormatLoudness": true
        },
        "streamSelectionConfig": {
          "maxBitrate": "10650000"
        },
        "mediaCommonConfig": {
          "dynamicReadaheadConfig": {
            "maxReadAheadMediaTimeMs": 120000,
            "minReadAheadMediaTimeMs": 15000,
            "readAheadGrowthRateMs": 1000
```

```
        }
    }
```

`streamingData` contains parameters such as average bitrate, codec, itag, URL, and duration for all the available representations. This section of the player file closely resembles manifests in DASH or HLS. `itag` is a format code unique for every codec-quality combination, [2] presents a table of these combinations. Listing below presents an example of such a representation section. This example video had 22 representations: encoded with H.264, AV1, and VP9 in a range of qualities.

```
{
  "itag": 397,
  "url": "https://...",
  "mimeType": "video/mp4; codecs=\"av01.0.04M.08\"",
  "bitrate": 624954,
  "width": 854,
  "height": 480,
  "initRange": {...},
  "indexRange": {...},
  "lastModified": "1639089123665964",
  "contentLength": "40169605",
  "quality": "large",
  "fps": 24,
  "qualityLabel": "480p",
  "projectionType": "RECTANGULAR",
  "averageBitrate": 429025,
  "colorInfo": {...},
  "approxDurationMs": "749039"
}
```

YouTube applies an amalgam of HLS and MP4 streaming technologies, the first one being used for live streaming and the second one for general videos on request. HLS streaming has been covered in section 2.2.4. However, MP4-based streaming deserves additional attention as it will be studied in the following sections of this work.

In MP4 streaming both video and audio tracks are requested in segments of bytes. It means that two requests need to be made separately for audio and video. Player requests bytes from a specific range as shown in the example below. `itag` 140 being ACC encoded 128 Kbps audio and `itag` 140 - 1080p video encoded with AV1. This approach allows the player to request segments of different sizes depending on the available network capacity.

```
  first request
https://...googlevideo.com/videoplayback...itag=140&range=0-87487
https://...googlevideo.com/videoplayback...itag=137&range=0-581908
  second request
https://...googlevideo.com/videoplayback...itag=140&range=87488-161919
https://...googlevideo.com/videoplayback...itag=137&range=581909-1707659
  third request
https://...googlevideo.com/videoplayback...itag=140&range=161920-320886
https://...googlevideo.com/videoplayback...itag=137&range=1707660-3624048
```

The URL used for the requests is retrieved from the representation section of the manifest. There is only one URL for the content and segments are requested by passing rage of bytes as a parameter. It is the streaming server that is responsible for sending the requested segment. What representation to request and in how big byte segments is decided by the rate adaptation algorithm described in the following section.

2.4 Related Work

A lot of existing studies on YouTube streaming focus on user Quality of Experience. Studies such as [14, 35, 41] provide a comprehensive overview of quality metrics but fail to examine the internals of YouTube's DASH implementation. However, for the scope of this work studies such as [29, 28] might be more applicable, as they investigate various aspects of YouTube beyond video quality. Among everything else, [29] covers characteristics of YouTube's adaptive streaming and how it behaves when network conditions are unstable. [25] provides an evaluation of YouTube's performance in adaptive streaming approach and claims that *"YouTube gains 83%-95% in terms of bandwidth by switching from progressive download to DASH"*. The above-mentioned studies underline a potential traffic wastage in video streaming, [36] covers this issue in great detail.

In [21] authors compare traffic patterns and test service performance in constrained conditions for YouTube and Netflix. They test the impact of bandwidth constraints, packet losses, and delay on the streaming process and concluded that *"losses have negative effects, but bandwidth restrictions are far more damaging"* and *"results indicate that YouTube is not much sensitive to delay."*

3 Work Layout

This work aims to implement network distortions and analyse their impact on video quality. Common network distortions include packet loss, jitter, transmission delay, bandwidth fluctuations, etc. When it comes to multimedia streaming, [21] has shown that bandwidth distortions are the most detrimental to the streaming performance both for YouTube and Netflix. Cisco report from 2021 has also stated that network speed is still an issue for developing countries of Africa and Asia [15]. Therefore, among all the network distortions this work will focus on bandwidth as the most relevant for the video streaming.

There is a need for a reference point to understand how changes in bandwidth might influence streaming. Within this work, the reference point is referred to as a *baseline test*. It is supposed to show how YouTube streaming behaves in conditions close to ideal, meaning that available bandwidth is practically unlimited and all the external factors are minimised. The baseline test aims to understand the rate adaptation mechanism at YouTube and what role bandwidth estimation plays in it.

In the main part of this work network distortions are to be implemented and their impact on video quality selection studied. As for the first part of this i.e. applying bandwidth distortions, several different tools could be used. For instance, Chrome-based browsers such as Google Chrome or Chromium provide a testing tool called "throttling" it could be found in the developer's tools (*DevTools*) section. This tool allows limiting bitrate and network delay with per-tab precision. The limit is set at the application level and *de facto* applies to limiting HTTP throughput. What it means is that TCP/UDP transmission remains unaffected and therefore this tool could not be used for the experiments as the main focus of this work is on the network distortions overall.

A more precise approach would be implementing bandwidth limitations at the network level using a switch or a router. Network devises of vendors such as Juniper, Cisco, or MikroTik can be configured to limit throughput on both layers 2 and 3 of the ISO/OSI model. This approach while being very efficient requires a special lab environment with open access to the device's configuration. The third approach utilises system-wide throttling on a socket level. Unix-base systems have an advanced traffic control tool named "`tc`" [3]. It allows for manipulating traffic control settings within the system kernel and can be applied to selected ports and addresses. The last option is to be used as it doesn't require a special environment, is accessible and adjustable.

By applying different bandwidth constraints for the same video sample it is possible to study how quality levels depend on available bandwidth. Within this work, the scenario of analysing the correlation of bandwidth and quality is called *Quality from bandwidth*. Results of this testing scenario are to be compared to the baseline to see how adaptation techniques behave in restricted network conditions.

As a supplementary step to better understand the impact bandwidth estimation has on quality selection, a *Bitrate estimation* is introduced. The idea is to estimate the bitrate requirement for the specific video quality. Bitrate requirement is a minimum bandwidth required for the video

transmission. It defines a lower boundary for the *Quality from bandwidth* scenario.

Figure 6 shows a diagram of all the testing scenarios: baseline test, bitrate estimation, and quality from bitrate.

Figure 6: Work layout

In this work two closely related terms are used frequently i.e. bitrate and bandwidth. They have the following meaning: bitrate means the quantity of data of video or audio transmitted in a unit of time, whereas bandwidth is defined as a data transfer rate or network capacity and refers to any kind of data. For instance, if a video's bitrate is higher than available bandwidth a single second of video will take more than one second to download.

This experiment could not be conducted for one selected video as the results would be exceptionally biased. Videos may have different encoding, quality levels, bitrate requirements, and so on. However, for some testing scenarios single video would be a more convenient option, for instance, analysis of streaming flow in the baseline test. For such cases, a video with frequent frame changes and a moderate amount of details was selected (video title: *"BTS Dishes on Touring and Working with Ed Sheeran — The Tonight Show Starring Jimmy Fallon"*, duration: 6 min. 55 sec., views: 5,500,000) [13].

To ensure the validity of the result, tests should be replicated over the dataset of different videos. The following section provides a comprehensive overview of the dataset and data used for testing.

3.1 YouTube Video Dataset

YouTube provides billions of videos for testing, as practically any of the videos could be used. There are two major factors when it comes to video selection; they are video compression and content delivery.

As was described in great detail in section 2.1.1, the compression achieved by codecs is not even among the videos. It means that two videos of equal duration may differ significantly in size after the compression. Several parameters such as content dynamics, colour space, detailedness influence how effective the compression is. To address this aspect there is a need for a diverse set of testing videos that would represent a range of values for each of the parameters.

The second aspect is content delivery. Streaming services widely use Content Delivery Networks (CDNs) in their streaming infrastructure. A CDN sits between the clients and the origin servers to deliver video content across different geographical regions and scale efficiently to ensure a smooth viewing experience for the clients [39]. It may have a significant impact on the delay before the

video start due to the transmission latency between a content server and end-user device. At the scope of this work primary focus is put on bandwidth, so minimising the impact of other parameters such as start-up delay is necessary.

Those two aspects could be extended to a list of requirements to be met by a video in order to qualify for the experiment. The requirements include:

- Representations availability - a video should be available in all the qualities and encoding format required in testing scenarios;
- Duration - a video should be at least 1 minute long. Shorter videos are shown to have a different download flow when compared to the longer ones (see section 4.3);
- Views - a video should have at minimum 500,000 views;
- Popularity - a video should reside on a list of YouTube trending for a given region. According to Variety magazine, *"to determine the year's top-trending videos, YouTube uses a combination of factors including measuring users interactions (number of views, shares, comments and likes). Note that they're not the most-viewed videos overall for the calendar year"* [4];
- Language - a primary language of the video should be English.

The last three requirements are closely related to CDNs. YouTube content is placed in regional CDN based on its popularity within the region. Taking trending videos with high views threshold ensures that content will be present in the CDN and thus start-up delay minimised.

YouTube Trending is a dataset of daily trending YouTube videos for a number of regions [42]. Videos selected from this dataset satisfy the popularity requirement, but need to be additionally filtered by views, duration, and representations. Moreover, the subset of selected videos has to be representative and diverse as mentioned above to take into account compression performance.

According to all the above mentioned, sample videos for testing were selected manually covering 15 different categories. As videos of every category have a different set of parameters. Below there is an overview of the categories. Figure 7 presents a pie chart of all of the categories.

- Sports, Autos & Vehicles, Music, Entertainment - represent videos of real-world origin, high dynamics, and moderate detailedness.
- People & Blogs, Howto & Style, Pets & Animals, News & Politics, Nonprofits & Activism, Comedy - represent videos of little dynamics and moderate detailedness.
- Film & Animation, Gaming - a category covering videos of both very high dynamics and with elements of computer graphics.
- Travel & Events - a category with very detailed real-world images of moderate dynamics.

Finally a subset of 120 videos was prepared; 8 videos per category based on the *YouTube Trending* dataset. Every video is available in 1080p, 720p, 480p, 360p, and 240p qualities and encoded with AV1.

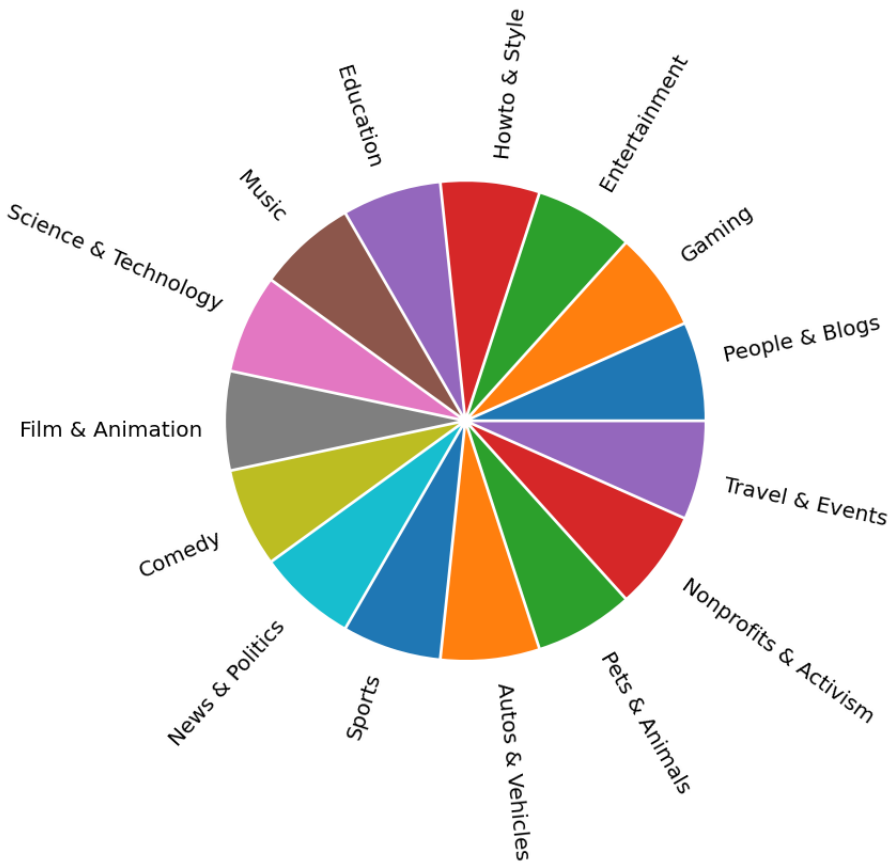

Figure 7: Dataset video types

3.2 Statistics for Nerds

For advanced users, YouTube provides an interface called *Stats for Nerds* for retrieving information on streaming statistics. It is available under right click on the player and includes the following parameters:

- `VideoID` - a unique alphanumeric ID number that is associated with the video and can be used to access it. [9] goes in detail on how ids are structured and used as identifiers in caching system.
- `sCNP` - a unique string that allows to identify that specific video playback. It is commonly used for debug purposes.
- `Viewport` - an actual resolution of the player window.
- `Frames` - the amount of played and dropped frames in the video so far.
- `Current Resolution` - the resolution in which the video is being played at that moment.
- `Optimal Resolution` - a target resolution estimated by player.
- `Volume / Normalized` - volume level set in player (in percents) and volume level in dB after normalization.

- `Codecs` - an information on codecs and its versions used for video and audio; also itags (see section 2.3) for audio and video tracks.
- `Connection Speed` - value of estimated available network bandwidth measured in Kbps. It is calculated by dividing the size of last chunk (from Network Activity) by the time it took to download it. This estimation is used by the player to in rate adaptation (see section 2.2.5).
- `Network Activity` - size in kilobytes of incoming video chunks. This value shows raw incoming data and do not take any overhead into account.
- `Buffer Health` - amount of buffered video measured in seconds. For example 30 seconds value means that player has received and decoded 30 seconds of video. Buffer is emptied with a steady pace as a result of the video playback.
- `Mystery Text` - additional information on video and player states and current timestamp, used for debugging purposes.

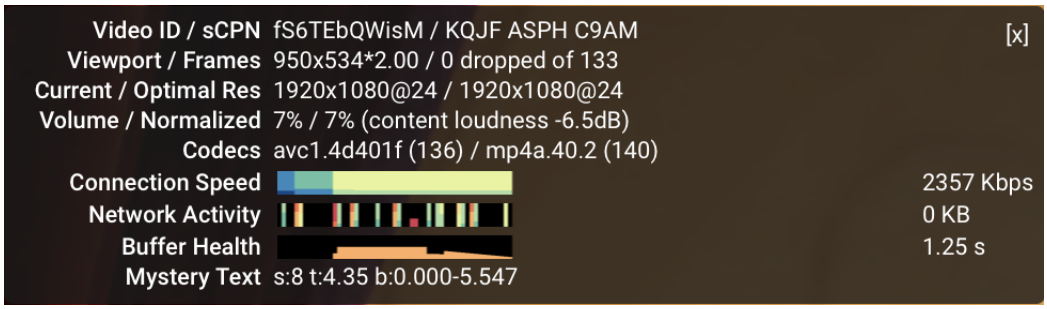

Figure 8: An example of *statistics for nerds*

Changes of *Network activity* and *Buffer health* in time give enough information on player's behaviour for analysing how it adapts to new network conditions. Additionally, *Current resolution* might be useful to observe what changes in other metrics correspond to a quality switch.

As *Statistics for nerds* are available only during the playtime, there is a need to collect them once to process further offline. YouTube provides a very simple visualization for some of the metrics, but this is not sufficient for the scope of this work. A dedicated Chrome extension was used created specifically for this purpose [5]; it is called *NerdCoder* and was developed for the needs of TUFIQoE project.

4 Tests and results

This section presents and discusses three different testing scenarios and provides an overview of the results. The first scenario (I - Baseline Test), gives a deeper understanding of YouTube's DASH streaming flow in the conditions of unlimited bandwidth. The following scenario (II - Bitrate Requirements Test) aims to accurately estimate the average bandwidth required for streaming with the specific video quality. The obtained results from test II are then used as threshold values in the last scenario (III - Quality from Bandwidth), where video quality is analysed as a function of available bandwidth. All the tests were replicated over the same 120 videos from the prepared dataset. To ensure equal conditions for every test, the player window size was fixed, as it might affect the playback (see section 3.2). The browser used in all the experiments was Google Chrome with enabled QUIC [23] as it becomes a widely used option [26].

At the end of this section an example of YouTube index file is analysed. It shows how YouTube player gets to know what video representations are available and where to find them.

4.1 Baseline test

The main objective for the first test is to understand the central concepts behind YouTube's implementation of the rate adaptation algorithm. The very basic example of such an algorithm has been presented in section 2.2.5. For this test Nerd statistics (section 3.2) were collected during the entire video playback. The conditions were unrestricted, no bandwidth limitations were applied and the player was set to the `Auto` option, thus enabling the rate adaptation. Figure 9 presents the Buffer State and Network Activity metrics as a function of video duration. In order to obtain a clearer picture, the following graph was composed for the sample video.

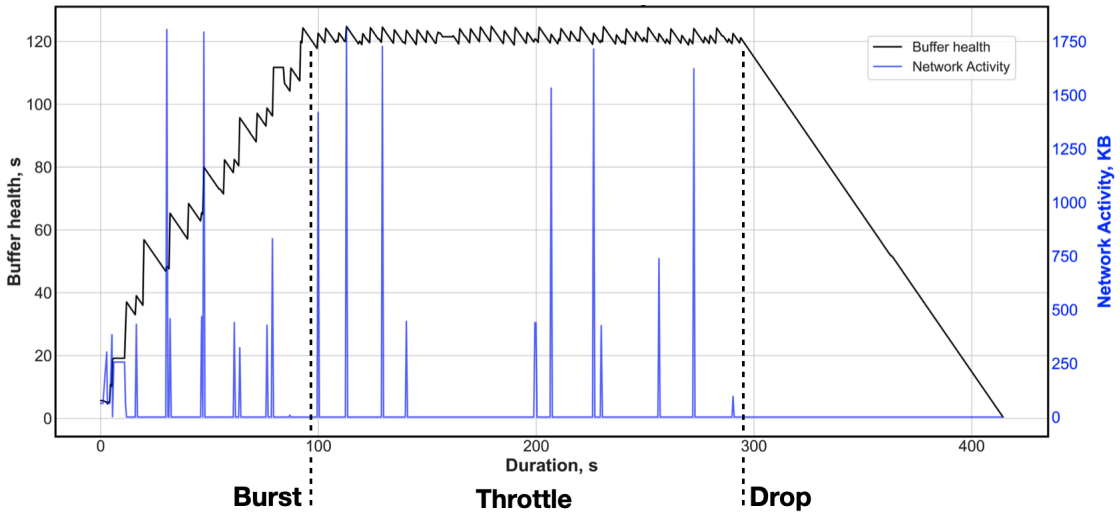

Figure 9: Baseline test - video streaming phases.

Three phases can be clearly distinguished in the playback. First i.e. from the second 0 to approximately second 100 is the so-called *burst phase*. Throughout this phase, the YouTube player requests chunks of video with high frequency trying to fill up the initially empty buffer. Correspondingly, the network activity during this phase is significantly denser and the buffer level rises rapidly.

At the beginning of the playback requested chunks are smaller and therefore are downloaded faster. Presumably, this feature is supposed to decrease the video start-up delay. Smaller chunks are downloaded and shown faster, thus starting the playback with a smaller delay. Meanwhile, the player starts requesting chunks of a bigger size as high peaks in network activity come right after the small ones. Figure 10 adds the third parameter i.e. quality switches, and shows that at the beginning player requests chunks of lower quality and then switches to higher. It seemingly has two goals: to bring startup delay even more down as lower qualities can be downloaded faster. Moreover, as a player starts with an empty buffer and no information on available bandwidth so it cannot accurately select a representation. Therefore, the second reason is that the algorithm starts with lower quality and after obtaining the first few chunks can estimate the bandwidth more correctly. In close-to-ideal circumstances of this experiment, the burst phase ends when buffer health reaches the level of 120 seconds, which means that buffer now contains enough data for the following 2 minutes of the video.

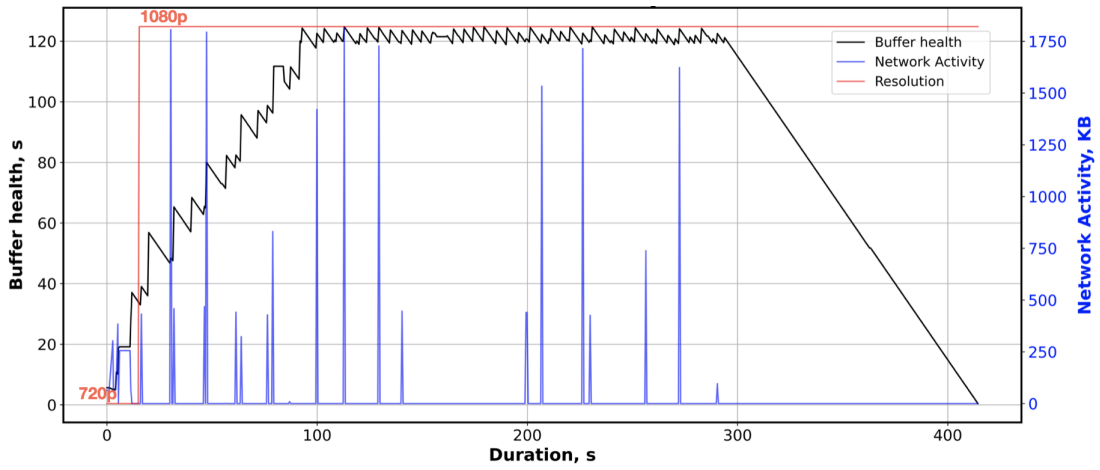

Figure 10: Buffer health, network activity, and quality switches in the baseline test.

At this stage player transitions into the *throttling* or "conservative" phase, maintaining buffer health on a steady level until it reaches the last stage. During the second stage of the process, chunks are requested less frequently but still are large in size of approximately 1.6MB. Those huge chunks are being decoded in the background and the buffer is incrementally updated. Therefore there is very little visible correlation between network activity peaks and fluctuations in the buffer health. Different chunk sizes may be also explained by a lower compression ratio.

The final stage occurs when the player's buffer contains enough data for the rest of the video. In circumstances of unlimited bandwidth, it usually occurs at the point of 2 minutes before the end of the video, which corresponds to the buffered 120 seconds. During this stage, the buffer

is emptied at a steady pace until the end of the playback. There is no more data to download, therefore network activity stays at zero level.

This testing scenario has shown that given unrestricted conditions YouTube's algorithm tends to request fewer bigger chunks, thus minimising the number of requests to the streaming server. Apparently, this strategy aims to decrease the load on content servers. What is more, it might not be as effective when a bandwidth drop occurs in the middle of the request and slows it down. So the request needs more time to finish and meanwhile, the player potentially can run out of buffered video. In unstable network conditions, this problem could be practically eliminated by requesting smaller segments. This approach is described in more detail in the following section 4.3.

4.2 Bitrate estimation

This scenario aims to estimate the bandwidth requirements for the specific quality levels. The test was conducted with no restrictions or bandwidth boundaries and was repeated for every video from the dataset. Tested quality levels included 320x240, 640x360, 720x480, 1280x720 and 1920x1080 for short referred to as 240p, 360p, 480p, 720p and 1080p. Rate adaptation was disabled and the quality level was selected manually. Figure 11 presents an example of a buffer state and network activity for the first 250 seconds of video playback. For this specific example, a 1080p quality level was selected.

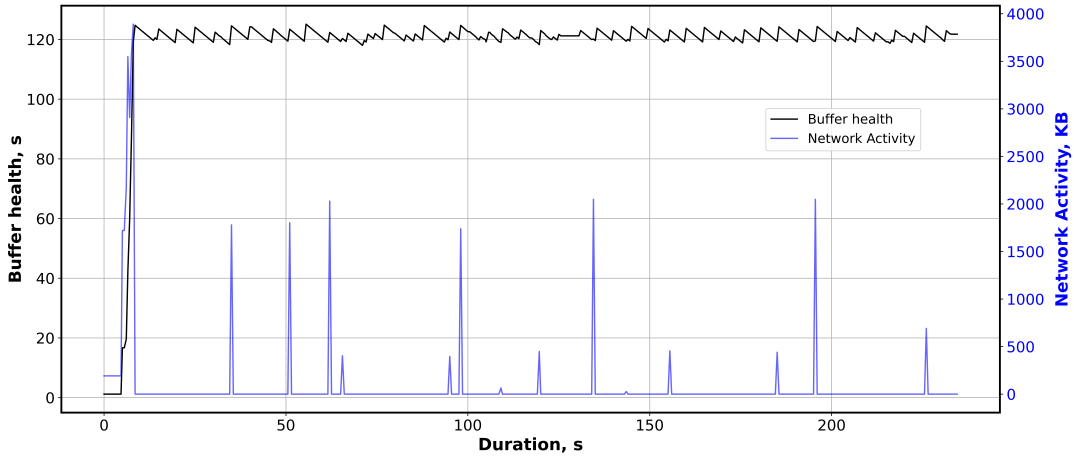

Figure 11: Buffer health and network activity with disabled rate adaptation.

As was described in the previous section 4.1, at the start of the playback player enters the burst phase and aggressively fills up the buffer. The corresponding massive peaks of network activity can be seen in the first 10 seconds of the playback. As quality level was predefined and set to 1080p, only chunks of that quality are requested, therefore resulting in a stiffer growth of the buffer health. The throttling phase follows the burst, keeping the buffer level steady.

The bandwidth requirements can be calculated using the following formula:

$$bandwidth = \frac{\sum_{t=0}^{t_0} N(t)}{t_0 + B(t_0)}$$

Where $N(t)$ is the network activity at the second t, and $B(t)$ is the buffer health at the second t. t_0 being the last timestamp of the recorded data.

The above formula was applied for every video from the dataset and for every quality level. Figure 12 presents a boxplot with bandwidth distributions for each of the quality levels.

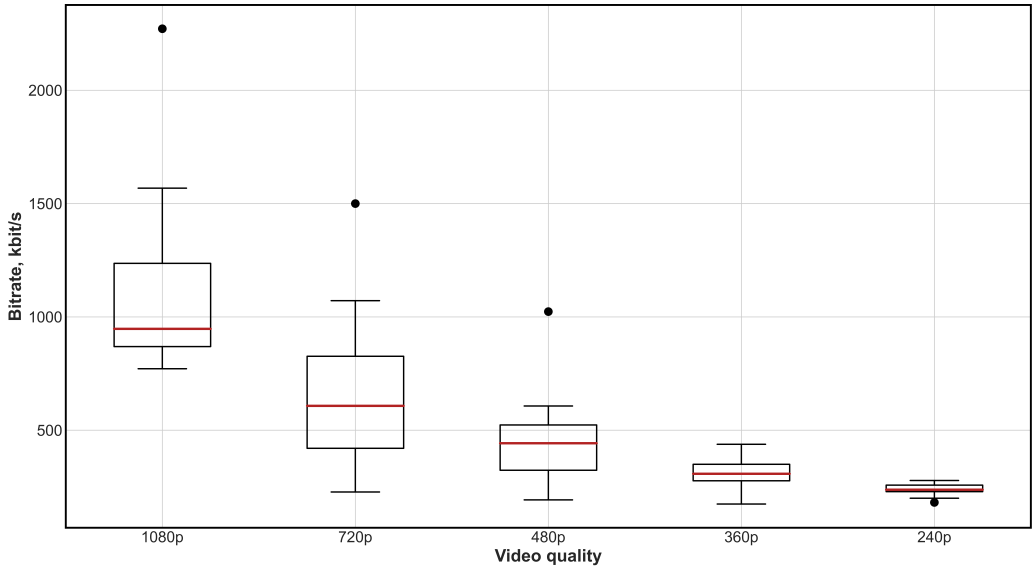

Figure 12: A boxplot of distributions of bitrate requirements for quality levels: 1080p, 720p, 480p, 360p, 240p

The data show, as expected, that bandwidth requirements are closely related to the quality levels. The correlation between quality levels and bandwidth can be noticed. Higher qualities have proportionally higher bandwidth requirements. Another important moment is that the variation grows with quality, meaning that high qualities have a significantly wider range of possible bitrates. Such data distribution can be explained by codec performance, when compression ratios are exceptionally high more movement and details are dismissed. Yet there could be noticed some outliers which bandwidth value is outstanding, it appears in each of the quality levels. This one sample is a trailer of an action-shooter game Call of Duty: Black Ops Cold War [43]. It is exceptionally dynamic and detailed, therefore presumably hard to compress, which explains it being far from the rest of the samples.

The values presented in Figure 12 do not include transmission overheads and represent requirements for the AV1 codec exclusively.

4.3 Quality from bandwidth

This test case aims to analyse YouTube's performance in conditions of limited network resources and map bandwidth to objective quality. For this experiment rate adaptation was enabled by selecting the `Auto` option.

Figure 13 shows three metrics: network activity, buffer health and quality switches. The `tc` tool has been used to limit throughput to 1Mbit/s; playback is shortened to the first 5 minutes.

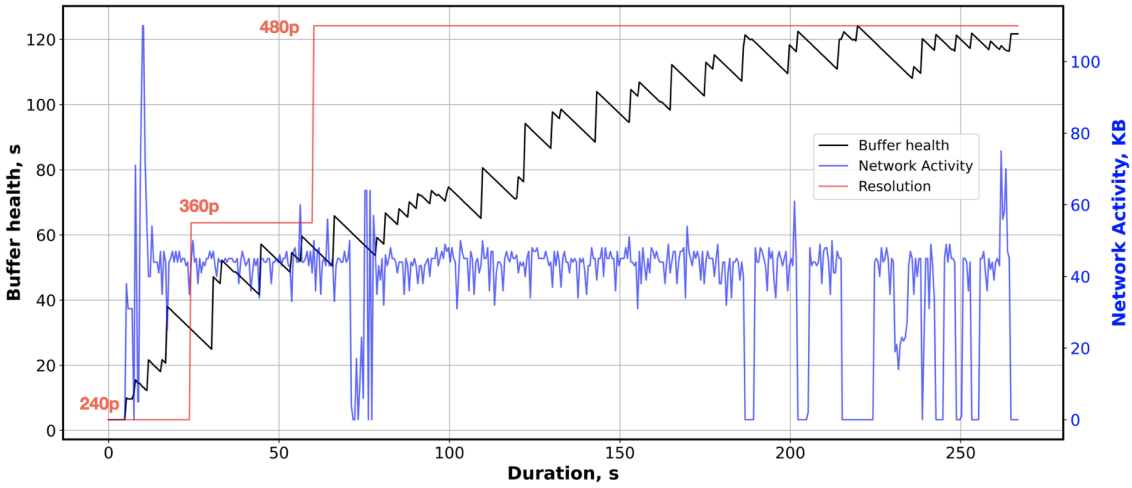

Figure 13: Buffer health, network activity, and quality switches in conditions of limited bandwidth

The most radical and noticeable changes, compared to Figure 10, are in the network activity domain. The incoming data is spread out evenly in time, not concentrated in bursts. The player makes frequent requests but for shorter byte ranges. Therefore the growing rate of buffer health is lower, yet the algorithm still tries to bring the level up to the 120-second threshold. Few quality switches are happening early in the playtime and corresponding buffer drops can be seen. The drops do not empty the buffer completely because chunks of better quality were requested in advance and are already present in the buffer. Different tests have shown, that in restricted conditions YouTube's algorithm requires about 60 seconds to estimate the bandwidth and start requesting chunks of optimal quality. As it can be seen in the example of the baseline test (section 4.1), the network activity in unconstrained conditions is rather bursty and not frequent. It makes sense if a streaming server is considered, fast and intensive connections are preferable to the long-lasting and slow ones [39].

Figure 14 shows another important issue: connection speed. This value is estimated by dividing chunk size by its download time. The full potential of the download is never actually achieved, it is not a problem for the content server because, in case of other limitations, higher speeds are reached. The described phenomenon can be explained by introducing the transmission overhead. Raw bytes of data are transported using HTTP and therefore have an HTTP header attached to each package of data. As Figure 15 shows, the average HTTP header size is about 1.05KB. As

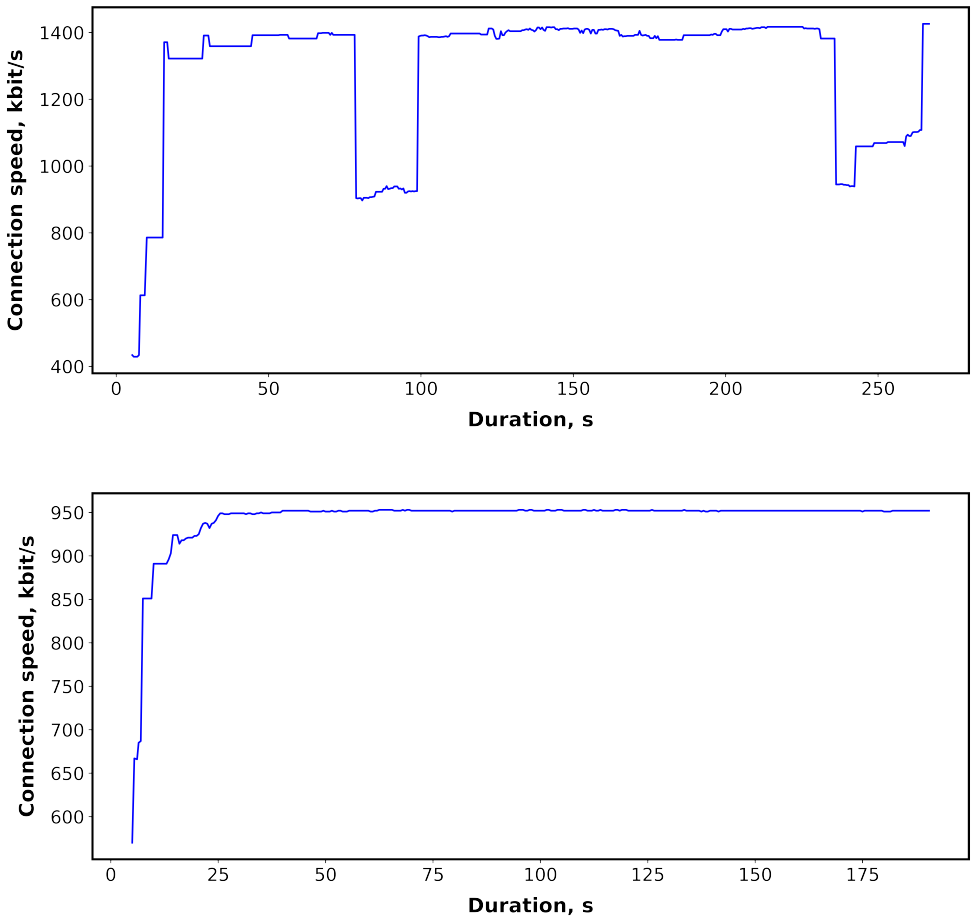

Figure 14: Connection speed estimated in a player for 1500kbit/s and 1000kbit/s throttling applied

video and audio tracks are requested separately this overhead doubles. In the case of high available bandwidth, this overhead is tiny in comparison to 1.5MB chunks of data. The opposite is true for the constrained bandwidth, as requests are more frequent and requested segments are smaller, the ratio of HTTP overhead to data gets bigger. For the examples presented in Figure 14, headers take up about 10% of the available bandwidth.

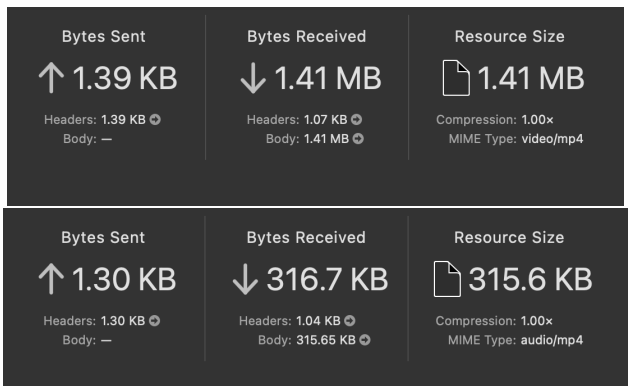

Figure 15: Summary of HTTP content request showing header size

In order to test how bandwidth limitations impact quality selection, a range of bandwidth limits were applied from 200 kbit/s up to 2Mbit/s for every video sample. As the previous experiment have shown, bitrate distributions fit well inside this range. Bandwidth was studied with a 100 kbit/s step. Figure 16 presents gamma distributions of the results.

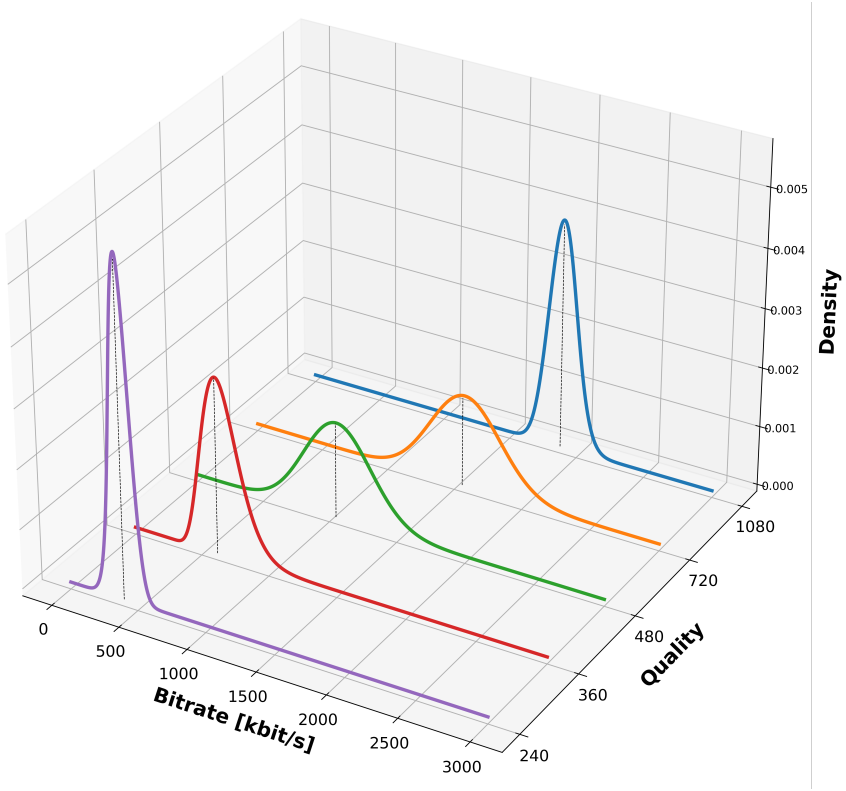

Figure 16: Gamma distributions of qualities achieved with specified bitrate

Values on the Figure 16 could be used for estimating the probability of achieving a certain quality level with the given available bandwidth.

4.4 Automation and implementation

The second and the third testing scenarios required a test to be repeated for every video sample from the dataset. Overall, it is 600 tests for the *Bandwidth estimation* scenario and 2040 for the *Quality from bitrate*. The time of the tests couldn't be reduced as a video needs to be played to gather the data, therefore it couldn't be sped up.

In order to automate the process of data collection, a script was written in the Python programming language [6]. The scripts aim is to sequentially open a video sample in a new tab of the Google Chrome browser, wait until the video ends or specified amount of time, close the tab, thus keeping only one tab open at a time, and go through all of the videos. Listing 1 shows a piece of code that implements the above-mentioned actions. A `webbrowser` Python library was used to implement opening URLs in the browser.

```
for rowi in range(df.shape[0]):
  video = df.iloc[[rowi]]
  web_controller.open(url = YT_URL_FORMAT % video.video_id.item())
  time_start = time.time()
  timeout = video.duration.item() if video.duration.item() <= VIDEO_DURATION_LIMIT
        else VIDEO_DURATION_LIMIT
  print(f"Video: `{video.title.item()}` has started, waiting {timeout} s.")
  # wait until the end of the video, max 3 min
  time.sleep(timeout)
```

Listing 1: Python code for testing automation

For the *Quality from bitrate* scenario, an additional step was added i.e. applying bandwidth limitations. Listing 2 shows a line of code that implements throttling using a `traffic control` tool. It is run as a shell command in a separate process using `subprocess` module [7].

```
subprocess.run(f"tc qdisc add dev eth0 root ingress rate {bw_limit}kbit".split(" ")
    )
```

Listing 2: Python function implementing bandwidth throttling

The above pieces of code only automate the testing, whereas the data and *statistics for nerds* are gathered using a dedicated Chrome extension [5]. It collects all of the necessary statistics and stores them in a locally managed database. Therefore testing results could be retrieved directly from the database after the testing is finished.

Additional automation is required during the data processing. The data from every playtime should be reduced to a single value. The following listing 3 presents a piece of code that calculates bitrate requirements providing data from the whole playback. It implements the formula presented in section 4.2.

```
def calc_normalized_bitrate(df, offset=0):
    playtime = (df.iloc[[-1]].timestamp_utc_ms.values[0] - df.iloc[[offset]].
    timestamp_utc_ms.values[0])/1000
    buf = df.iloc[[-1]].buffer_health.values[0] - df.iloc[[offset]].buffer_health.
    values[0]
```

```
    activity_sum = df.network_activity[offset:].sum()*8
    activity_overhead = df.network_activity[offset:].count() *
    HTML_PLAYBACK_CHUNK_OVERHEAD
    downloaded_sec_of_video = buf + playtime
    return (activity_sum+activity_overhead)/downloaded_sec_of_video +
            TRANSMISSION_OVERHEAD
```

Listing 3: Python function for average bitrate calculation

This and other code for data processing and data visualisation cloud be found in [8]. Its structure is self-explanatory: `notebooks` folder contains Python interactive notebooks and `graphs` folder which contains all the graphs from this work and some additional visualisations. Notebooks have the following structure: `scrap_yt.ipynb` contains code for scrapping YouTube to retrieve additional data on a video e.g. duration; `study_ds.ipynb` contains code for *YouTube trending* dataset exploration; `nerd_stats_processing.ipynb` has code for both data processing and some of the visualisations; and `quality_final.ipynb` contains code for complex 3D visualisations and distribution fitting.

5 Conclusion and future directions

For this work, two main goals were set: to better understand streaming flow in the YouTube application, and to implement and analyse how network distortions influence the selection of video quality. As for the first goal, the YouTube streaming process has been examined starting from fetching an index file up to selecting chunks of what quality to request next. An example of the streaming process has been described in great detail covering burst, throttling, and drop phases.

As for the second goal, the `tc` Linux tool was used to simulate different bandwidths. It allowed creating a distribution of bandwidth for the specific quality level. Given the distribution, it is possible to estimate the quality level based on bandwidth measurements. To ensure valid results and minimise the impact that compression may have, all of the tests were repeated over the dataset of 120 videos.

At the scale of the TUFIQoE [40] project, a correlation between video quality and perceived quality is going to be studied. As this work has shown the connection between network parameters and video quality, it would be possible to estimate QoE based on network parameters as the future step. It will require combining system and human factors to create a comprehensive tool for QoE estimation based on network measurements. This could be done similar to the approach used in the last testing scenario: by composing a distribution function for each of the quality levels.

References

[1] URL: https://www.json.org.

[2] URL: https://gist.github.com/AgentOak/34d47c65b1d28829bb17c24c04a0096f.

[3] URL: https://man7.org/linux/man-pages/man8/tc.8.html.

[4] URL: https://variety.com/2017/digital/news/youtube-2017-top-trending-videos-music-videos-1202631416/.

[5] URL: https://github.com/navuyi/NerdCoder-TUFIQoE-2021.

[6] URL: https://www.python.org.

[7] URL: https://docs.python.org/3/library/subprocess.html.

[8] URL: https://github.com/mmuravytskyi/youtube_video_quality.

[9] Vijay Kumar Adhikari et al. "Vivisecting YouTube: An active measurement study". In: *2012 Proceedings IEEE INFOCOM*. 2012, pp. 2521–2525. DOI: 10.1109/INFCOM.2012.6195644.

[10] Petteri Aimonen. *A sequence of Intra-coded, Predicted and Bi-predicted frames in a compressed video sequence*. 2009. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/6/64/I_P_and_B_frames.svg/1024px-I_P_and_B_frames.svg.png.

[11] Nabajeet Barman and Maria G. Martini. "QoE Modeling for HTTP Adaptive Video Streaming–A Survey and Open Challenges". In: *IEEE Access* 7 (2019), pp. 30831–30859. DOI: 10.1109/ACCESS.2019.2901778.

[12] Kjell Brunnström. "Qualinet white paper on definitions of quality of experience". In: (2013).

[13] *BTS Dishes on Touring and Working with Ed Sheeran — The Tonight Show Starring Jimmy Fallon*. URL: https://www.youtube.com/watch?v=ff6zBfAmX58.

[14] Xianhui Che, Barry Ip, and Ling Lin. "A Survey of Current YouTube Video Characteristics". In: *IEEE MultiMedia* 22.2 (2015), pp. 56–63. DOI: 10.1109/MMUL.2015.34.

[15] *Cisco Annual Internet Report (2018–2023)*. Tech. rep. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf.

[16] Conviva. *State of streaming, Q3 2021*.

[17] Bo Fu et al. "Video WatchTime and Comment Sentiment: Experience from YouTube". In: 2016.

[18] *Global Internet Phenomena Report*. 2021. URL: https://www.sandvine.com/phenomena.

[19] Christer Holmberg, Stefan Hakansson, and Goran Eriksson. *Web Real-Time Communication Use Cases and Requirements*. RFC 7478. Mar. 2015. DOI: 10.17487/RFC7478. URL: https://rfc-editor.org/rfc/rfc7478.txt.

[20] *Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats*. Standard. Geneva, CH: International Organization for Standardization, 2019.

[21] Maria Silvia Ito et al. "Network level characterization of adaptive streaming over HTTP applications". In: *2014 IEEE Symposium on Computers and Communications (ISCC)*. 2014, pp. 1–7. DOI: 10.1109/ISCC.2014.6912603.

[22] *ITU-T Report, Definition of Quality of Experience (QoE)*. 2007.

[23] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000. URL: https://rfc-editor.org/rfc/rfc9000.txt.

[24] S. S. Krishnan and R. K. Sitaraman. *Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs*. 2012.

[25] Dilip Kumar Krishnappa, Divyashri Bhat, and Michael Zink. "DASHing YouTube: An analysis of using DASH in YouTube video service". In: *38th Annual IEEE Conference on Local Computer Networks*. 2013, pp. 407–415. DOI: 10.1109/LCN.2013.6761273.

[26] Etienne Liebetrau. *How Google's QUIC Protocol Impacts Network Security and Reporting*. 2021. URL: https://www.fastvue.co/fastvue/blog/googles-quic-protocols-security-and-reporting-implications/.

[27] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. "Rate Adaptation for Adaptive HTTP Streaming". In: (2011). DOI: 10.1145/1943552.1943575.

[28] Christian Moldovan et al. "YouTube Can Do Better: Getting the Most Out of Video Adaptation". In: *2016 28th International Teletraffic Congress (ITC 28)*. Vol. 03. 2016, pp. 7–12. DOI: 10.1109/ITC-28.2016.309.

[29] Abhijit Mondal et al. "Candid with YouTube: Adaptive Streaming Behavior and Implications on Data Consumption". In: *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV'17. Taipei, Taiwan: Association for Computing Machinery, 2017, pp. 19–24. ISBN: 9781450350037. DOI: 10.1145/3083165.3083177. URL: https://doi.org/10.1145/3083165.3083177.

[30] Md Fahad Monir and Shanjidah Akhter. "Comparative Analysis of UDP Traffic With and Without SDN-Based Firewall". In: *2019 International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST)*. 2019, pp. 85–90. DOI: 10.1109/ICREST.2019.8644395.

[31] Karl Norrman et al. *The Secure Real-time Transport Protocol (SRTP)*. RFC 3711. Mar. 2004. DOI: 10.17487/RFC3711. URL: https://rfc-editor.org/rfc/rfc3711.txt.

[32] Roger Pantos. *HTTP Live Streaming 2nd Edition*. Internet-Draft draft-pantos-hls-rfc8216bis-10. Work in Progress. Internet Engineering Task Force, Nov. 2021. 96 pp. URL: https://datatracker.ietf.org/doc/html/draft-pantos-hls-rfc8216bis-10.

[33] Roger Pantos and William May. *HTTP Live Streaming*. RFC 8216. Aug. 2017. DOI: 10.17487/RFC8216. URL: https://rfc-editor.org/rfc/rfc8216.txt.

[34] L. Sevcik et al. "Prediction of Subjective Video Quality Based on Objective Assessment". In: *2018 26th Telecommunications Forum (TELFOR)*. 2018, pp. 1–4. DOI: 10.1109/TELFOR.2018.8612127.

[35] Christian Sieber et al. "Sacrificing efficiency for quality of experience: YouTube's redundant traffic behavior". In: *2016 IFIP Networking Conference (IFIP Networking) and Workshops*. 2016, pp. 503–511. DOI: 10.1109/IFIPNetworking.2016.7497231.

[36] Christian Sieber et al. "The cost of aggressive HTTP adaptive streaming: Quantifying YouTube's redundant traffic". In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 1261–1267. DOI: 10.1109/INM.2015.7140478.

[37] Tongyu Song et al. "FAIR-AREA: A Fast AI-Based Joint Optimization of Rate Adaptation and Resource Allocation for DASH". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. 2019, pp. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9013477.

[38] Jim Summers et al. "To chunk or not to chunk: Implications for HTTP streaming video server performance". In: *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*. 2012, pp. 15–20.

[39] Ruben Torres et al. "Dissecting Video Server Selection Strategies in the YouTube CDN". In: *2011 31st International Conference on Distributed Computing Systems*. 2011, pp. 248–257. DOI: 10.1109/ICDCS.2011.43.

[40] *Towards Better Understanding of Factors Influencing the QoE by More Ecologically-Valid Evaluation Standards*. URL: https://qoe.agh.edu.pl/tufiqoe/.

[41] Florian Wamser et al. "Modeling the YouTube stack: From packets to quality of experience". In: *Computer Networks* 109 (2016). Traffic and Performance in the Big Data Era, pp. 211–224. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2016.03.020. URL: https://www.sciencedirect.com/science/article/pii/S1389128616300925.

[42] *YouTube Trending Dataset*. URL: https://www.kaggle.com/datasnaek/youtube-new.

[43] *YouTube: Call of Duty®: Black Ops Cold War - Multiplayer Reveal Trailer*. URL: https://www.youtube.com/watch?v=rXRQyd6_5j4.

[44] Mark Watson Te-Yuan Huang Ramesh Johari Nick McKeown Matthew Trunnell. "A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service". In: 44.4 (2014). DOI: 10.1145/2619239.2626296.

List of Figures