future internet

MDPI

Article

Cache-Enabled Adaptive Video Streaming: A QoE-Based Evaluation Study

Eirini Liotou *, Dionysis Xenakis, Vasiliki Georgara, Georgios Kourouniotis and Lazaros Merakos

Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, 16121 Athens, Greece; nio@di.uoa.gr (D.X.); en2190002@di.uoa.gr (V.G.); en2190003@di.uoa.gr (G.K.); merakos@di.uoa.gr (L.M.)

* Correspondence: author: eliotou@di.uoa.gr

Abstract: Dynamic Adaptive Streaming over HTTP (DASH) has prevailed as the dominant way of video transmission over the Internet. This technology is based on receiving small sequential video segments from a server. However, one challenge that has not been adequately examined is the obtainment of video segments in a way that serves both the needs of the network and the improvement in the Quality of Experience (QoE) of the users. One effective way to achieve this is to implement and study caching and DASH technologies together. This paper investigates this issue by simulating a network with multiple video servers and a video client. It then implements both the peer-to-many communications in the context of adaptive video streaming and the video server caching algorithm based on proposed criteria that improve the status of the network and/or the user. Specifically, we investigate the scenario of delivering DASH-based content with the help of an intermediate server, apart from a main server, to demonstrate possible caching benefits for different sizes of intermediate storage servers. Extensive experimentation using emulation reveals the interplay and delicate balance between caching and DASH, guiding such network design decisions. A general tendency found is that, as the available buffer size increases, the video playback quality increases to some extent. However, at the same time, this improvement is linked to the random cache selection algorithm.

Keywords: HTTP adaptive video streaming; Quality of Experience; caching; video resolution; encoding; stallings

Citation: Liotou, E.; Xenakis, D.; Georgara, V.; Kourouniotis, G.; Merakos, L. Cache-Enabled Adaptive Video Streaming: A QoE-Based Evaluation Study. *Future Internet* **2023**, *15*, 221. https://doi.org/10.3390/fi15070221

Received: 10 April 2023
Revised: 10 June 2023
Accepted: 19 June 2023
Published: 21 June 2023

1. Introduction

With the widespread penetration of the broadband Internet, multimedia services are getting increasingly popular among users. Video streaming is considered to be a major source of Internet traffic today, and its usage continues to grow at a rapid rate. Being a very bandwidth-consuming kind of application, video traffic is estimated to account for around 70% of all mobile data traffic, a share that is forecast to increase to 80% in 2028. Service providers expect additional traffic growth with the introduction of new video services, such as HD video and XR services [1].

On the one hand, to cope with this new and massive source of traffic, network providers have proposed methods to reduce the amount of traffic traversing their networks and serve all the customers in a better way. One of the basic techniques to decrease the volume of video traffic circulating in the networks is ***caching***, which allows every single user to efficiently reuse previously retrieved data. On the other hand, to facilitate smooth video streaming without interruptions and in a way that guarantees a high user experience, the concept of ***Dynamic Adaptive Streaming over HTTP (DASH)*** has been introduced and is widely used by video service providers nowadays, such as YouTube.

Furthermore, it is important to investigate video quality from the end users' perspective, which is widely known as ***Quality of Experience (QoE)***. Currently, there is no strict

definition of QoE. The International Telecommunication Union (ITU-T) defines QoE as "*the overall acceptability of an application or service, as perceived subjectively by the end user.*" Similarly, in the European "Qualinet" community, QoE is defined as "*the degree of delight or annoyance of the user of an application or service. It results from the fulfilment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state.*" [2]. However, the general understanding of QoE is the same to a large extent: QoE is a new measurement approach for communication services, and it is determined by the interactions between users and services.

Caching and DASH have been studied together in the current state of the art, but from different perspectives. Many papers try to jointly optimize caching and video segment selection. In [3], the tradeoff problem between hit ratio and content quality in edge caching systems for multiuser Adaptive Bitrate Streaming (ABS) services is examined, wherein both decisions are tightly integrated into a flexible policy. In this case, an edge server inquires the original content from the content providers and encodes the content at the assigned bitrate, while in our case, a caching policy has already pre-allocated video segments in proxies in the network (not an on-demand basis). In [4], a dynamic caching scheme for adaptive streaming is proposed wherein the video quality adaptation at the application layer and the cache placement at the base station are performed together at a large time scale. Other works combine QoE and caching. The work in [5] optimizes the video segment selection to be cached at each edge server in order to effectively reduce the service load of the base station while maintaining a high QoE, while the work in [6] jointly provides proactive caching, power allocation, user association and adaptive video streaming, forming a QoE-aware throughput maximization problem. In another series of papers, the end user video segment selection decisions are overwritten by an intelligent network-side control logic running at an Access Point or base station/edge server, in order to facilitate the use of cached video content [7] or in general in case the alternative quality provides better QoE to the end user [8]. Similarly, network assistance is considered in [9], wherein routers perform bandwidth estimation and notify the user's application of cached segment information followed by a user QoE-aware video segment selection that matches the bandwidth of the bottleneck link. Finally, a different category of works formulates the video segment caching as an optimization problem with the objective to maximize the average video bitrate for all streamers (with respect to the limit capacity of the storage size and computing capacity) [10].

In our paper, we differentiate from the previous works in the sense that we do not consider network assistance (e.g., SAND architecture) or a global (optimized) view of the network. Instead, we "take a step back" and focus on a more "practical" approach of investigating the interplay and impact when both DASH and caching are present in the network, but these two technologies are not cooperating. Therefore, we evaluate such an interplay in realistic situations, i.e., in networks as they are today, revealing some valuable experimental insights when it comes to parameterization and design decisions to the owners of the infrastructure, as of today.

Moreover, we study how QoE-related metrics, such as the video resolution and number of stallings, among others, are influenced, while streaming video from a server to a client using DASH when caching is also available. Specifically, we investigate the scenario of delivering DASH-based content with the help of an intermediate server, apart from the main server, to demonstrate possible caching benefits for different sizes of intermediate storage servers. For this reason, we create a simple client-cache proxy server topology to implement the DASH logic, wherein if the cache proxy server contains some segments, it serves them immediately.

The novelty of this paper can be summarized as follows:

- The interplay and "indirect" dependencies between DASH-related decisions (video segment selection) and caching (cache hit or miss) are investigated and revealed through an extensive evaluation study.

- Such dependencies are quantified, guiding the network operators of current infrastructures to make more informed decisions when dimensioning their networks (i.e., selecting the appropriate bandwidth allocation related parameters).

Additional contributions include the following:

- The proposal of metrics that can capture the QoE of end users in these kinds of scenarios.
- The proposal of a "Send While Get" DASH-based approach, wherein the segments are fetched from the server simultaneously and are sent to the client without the need for any queue. The impact of this approach on channel utilization is also explained while it is compared with a "Get Before Send" approach.
- The design of a complete, end-to-end emulation framework, which (a) configures the DASH manifest file and prepares the multi-resolution video segments, (b) caches these segments using a random algorithm at involved servers, (c) performs the DASH-based algorithmic logic through a VLC client and (d) supports the caching hit/miss processes and respective segment forwarding.

This study is therefore applicable in cases of clients video streaming in a particular region where there are cache servers in the proximity. These cache servers receive popular videos proactively from a main, backhaul server that has all videos available (e.g., Netflix service). In this way, these clients can be served by the cache, reaping the benefits of proximity (such as lower delay in receiving the segment requests), while at the same time the backhaul network is significantly reducing its overhead. Especially in cases of mobile streaming, wherein the channel conditions are unpredictable and fluctuating, cache-aided mobile-edge computing (MEC) can lead to even more noticeable benefits.

The structure of this paper is as follows: Section 2 introduces the concept and implementation idea behind DASH, while Section 3 describes the concept of caching, focusing on caching for video streaming purposes. Next, Section 4 discusses the concept of QoE in more detail presenting its concept as well as the main QoE-influencing factors for video streaming. Then, Section 5 describes the environment setup for the purposes of this work, while Section 6 presents and analyses the emulation results. Finally, Section 7 concludes the paper. (Note: this work is based on an MSc thesis [11].)

2. HTTP Adaptive Video Streaming

2.1. Concept

Adaptive streaming is a new streaming approach that is designed to deliver multimedia to the user in the most efficient way possible and in the highest available quality for each specific user. Video content is encoded at different quality levels, and each quality level is determined by its corresponding average bitrate. In order to adaptively stream media, the media are split up into chunks, so the content is divided into segments that have a typical duration (most commonly one to ten seconds). Each segment can be decoded independently of other segments. A DASH client initiates a new session by downloading a manifest file. This manifest file is used to hold the information and description of the various streams and the bandwidths they are associated with. Based on the network conditions and the current buffer filling level, the Rate Determination Algorithm (RDA) in the DASH client determines the quality for the next segment download. The objective of the RDA is to optimize the global QoE determined by the occurrence of video freezes, the average quality level and the frequency of quality changes. This procedure is shown in Figure 1.

The main advantage of DASH over the traditional download and common real-time streaming is that it is able to adapt video quality (and bitrate) according to the available bandwidth so as to avoid video stallings. Consequently, DASH facilitates video streaming over a best-effort network. In addition, HTTP-based video streams can easily traverse firewalls and reuse the already deployed HTTP infrastructure such as HTTP servers, HTTP proxies and Content Delivery Network (CDN) nodes. Because of these advantages, major players such as Microsoft, Apple, Adobe and Netflix have massively adopted the adaptive streaming paradigm.

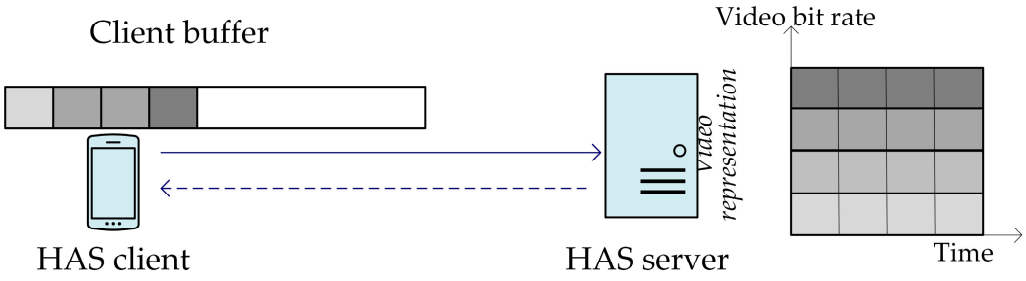

Figure 1. The concept of DASH.

Adaptive streaming is preferred over traditional progressive streaming for two main reasons. The first is quality, meaning that a video that is only 1280 × 720 will never play at correct quality levels on a screen that is 1920 × 1080; it will be stretched and pixelated. On the contrary, adaptive streaming allows the video provider to create a different video for each of the resolutions that he or she wishes to target.

The second problem is buffering, as sequential pauses in the video are unpleasant for all users. Buffering happens commonly when there is an unstable Internet connection. Consequently, the video has to stall several times because it cannot be timely downloaded, in order to wait for more data and then start downloading again. Most videos play at 24 frames per second, so the Internet connection needs to download at least 24 frames every second to avoid buffering. This problem is very common, especially on mobile devices, wherein the connection can vary greatly depending on the user's location and can cause a bad user experience.

Adaptive streaming can resolve this second buffering problem by "adapting" to the user's Internet connection speed. A small video can be downloaded faster than a large video, so if a user has a slow Internet connection, an adaptive video stream will switch to a smaller video file size to keep the video playing. From a user's perspective, it is preferable to watch a few minutes of lower-quality video in order to avoid buffering than to sit and watch a spinning icon until the stream catches up.

MPEG-DASH is a standard flexible bitrate streaming technique. MPEG has developed quite a few extensively used multimedia standards, including MPEG-21, MPEG-7, MPEG-4 and MPEG-2. The newest standard MPEG-DASH is an effort to resolve the intricacies of media delivery to various devices with an integrated common standard. When media content is delivered from conventional HTTP web servers, MPEG-DASH empowers high-quality streaming of this media content over the Internet.

2.2. Implementation

A media streaming scenario between a simple HTTP server and a DASH client takes place as follows. The multimedia content is captured and stored on an HTTP server and is delivered using HTTP. The content exists on the server in two parts: Media Presentation Description (MPD), which describes a manifest of the available content, its various alternatives, their URL addresses and other characteristics; and segments, which contain the actual multimedia bitstreams in the form of chunks, in single or multiple files.

The MPEG-DASH Media Presentation Description (MPD) is basically an XML document containing information about media segments, their relationships and the information necessary to choose among them and other metadata that may be needed by clients. The structure of an MPD is as follows (Figure 2):

- **Periods**, which are contained in the top-level MPD element, describe a part of the content with a start time and duration. Multiple periods can be used for scenes or chapters, or to separate ads from program content.
- **Adaptation sets**, which contain a media stream or a set of media streams.
- **Representations** allow an adaptation set to contain the same content encoded in different ways. In most cases, representations will be provided in multiple screen sizes

and bandwidths in order to allow clients to request the highest-quality content that they can play without waiting to buffer or wasting bandwidth.

- **Sub-representations** contain information that only applies to one media stream in a representation. They also provide information necessary to extract one stream from a multiplexed container, or to extract a lower-quality version of a stream.
- **Media segments** are the actual media files that the DASH client plays, generally by playing them back-to-back as if they were the same file. Media segment locations can be described using BaseURL for a single-segment representation, a list of segments (SegmentList) or a template (SegmentTemplate).
- **Index segments** come in two types: one representation index segment for the entire Representation, which is always a separate file, or a single index segment per media segment, which can be a byte range in the same file as the media segment.

```
<?xml version="1.0"?>
<!-- MPD file Generated with GPAC version 0.5.2-DEV-revVersion: 0.5.2-426-gc5ad4e4+dfsg5-3ubuntu0.1  at 2021-01-05T10:44:13.636Z-->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500S" type="static" mediaPresentationDuration="PT0H10M34.625S" maxSegmentDuration="PT0H0M5.000S" profiles="urn:mpeg:dash:profile:full:2011">
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
  <Title>manifest.mpd generated by GPAC</Title>
 </ProgramInformation>

 <Period duration="PT0H10M34.625S">
  <AdaptationSet segmentAlignment="true" bitstreamSwitching="true" maxWidth="1920" maxHeight="1080" maxFrameRate="24" par="16:9" lang="und">
   <SegmentList>
    <Initialization sourceURL="manifest_set1_init.mp4"/>
   </SegmentList>
   <Representation id="1" mimeType="video/mp4" codecs="avc3.640028" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="4305032">
    <SegmentList timescale="12288" duration="61440">
     <SegmentURL media="video_1080_1.m4s"/>
     <SegmentURL media="video_1080_2.m4s"/>
     <SegmentURL media="video_1080_3.m4s"/>
     <SegmentURL media="video_1080_4.m4s"/>
     <SegmentURL media="video_1080_5.m4s"/>
     <SegmentURL media="video_1080_6.m4s"/>
     <SegmentURL media="video_1080_7.m4s"/>
     <SegmentURL media="video_1080_8.m4s"/>
```

Figure 2. Example of an MPD file.

In order to play the content, the DASH client first requests the MPD file from the server. The MPD can be delivered in various ways, such as via HTTP or email. By parsing the MPD, the DASH client knows all the information about the program timing, media-content availability, media types, resolutions, minimum and maximum bandwidths, the existence of various encoded alternatives of multimedia components, accessibility features and required digital rights management (DRM), media-component locations on the network and other content characteristics. The client uses the information to select the appropriate encoded alternative in order to start streaming the content by fetching the segments using HTTP GET requests.

After appropriate buffering to allow for network throughput variations, the client continues fetching the subsequent segments but keeps on monitoring the network bandwidth fluctuations. Depending on its measurements, the client decides how to adapt to the available bandwidth by fetching segments of different bitrates to avoid buffering.

The MPEG-DASH specification only defines the MPD and the segment formats. The delivery of the MPD and the media encoding formats containing the segments, as well as the client behaviour for fetching, adaptation heuristics and playing content are outside of MPEG-DASH's scope.

3. Caching

3.1. Concept

Caching is an industry-recognized technology that has been employed in various areas of the computer and networking industry for quite some time, and it is widely used to improve user response time and minimize bandwidth utilization. Caching is the process of storing copies of frequently used data in an easily accessible and temporary location so that time and resources are saved due to the fact that data do not have to be retrieved from the original source (e.g., from a server). Because time and resources are always a matter of

great concern in the computer and networking industry, caches exist everywhere and are used in all high-performance systems.

3.2. Caching for Video Streaming

A Video-on-Demand system (VoD) system, which provides a service for users, typically consists of two main components: the central server and clients. The central server has a large storage space to store all the available videos for clients connected via a wide area network (WAN) or a local area network (LAN). In such a framework, all the requests from clients are handled at the central server. The request process starts with generating a request message from clients to the central server. In response to the client's request, the central server serves each request individually with a dedicated channel. Although this operation is simple to implement, the whole architecture is excessively expensive and lacks scalability due to the fact that the bandwidth bottleneck of the central server limits the number of clients it can serve, which will lead to a significant drop in the users' QoE. Furthermore, the introduction of long service latencies is another critical factor affecting the system performance, which is especially crucial when the video is transmitted over a WAN.

To leverage the workload of the central server and reduce the service latencies, an intermediate device called a proxy (or cache) is placed between the central server and clients making requests to download the content of their choice (Figure 3). In the proxy-based architecture, a portion of the video is cached in the proxy. Upon receiving the request, the proxy checks to see if it has a copy of the requested object in its cache. If so, the proxy responds by sending the cached object to the client (***cache hit***). Otherwise, it sends the request to the server (***cache miss***). If the proxy requests the object from the origin server, that object data are cached by the proxy so that it can fulfil future requests for the same object without retrieving it again from the origin server. The result of serving a cached object is that there is zero server-side bandwidth usage for the content and a huge improvement in response time for the end user. Meanwhile, the central server also delivers the uncached portion of the video to the client directly [12].

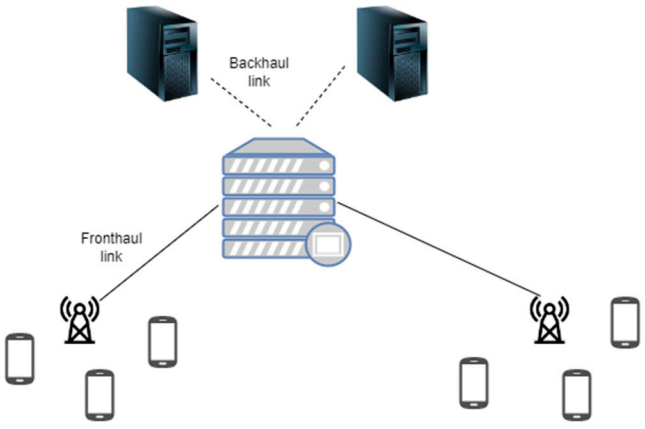

Figure 3. Client–server connection with intermediate proxy.

3.3. Caching in DASH

Caching may improve the overall performance of streaming protocols like DASH. As explained previously, in DASH, the same media are encoded at different resolutions, bitrates and frame rates, called representations, and each representation is virtually or physically divided into segments. The client, via this protocol, can download the appropriate segment according to its available bandwidth in order to have the best sustainable experience. Cache servers are deployed within the access network to cache and store the available DASH segments according to the cache's buffer size. This allows clients to be served from the cache when we have a cache hit rather than retrieving every video segment from the

origin server. This leads to higher speeds, lower latency, traffic avoidance and better QoE. However, the interplay between caching and DASH is known to be intricate, and possibly detrimental to QoE.

Consider, for example, the scenario wherein the DASH algorithm ***overestimates*** the available bandwidth when a video segment is found on the cache server. In such an occasion, the client may upshift to a higher bitrate representation; however, if the subsequent segment for the higher bitrate representation is not already stored in the cache proxy server, the proxy will have to retrieve it from the origin server. As a result, an increased delivery delay will be observed by the client as a lower throughput. The client may then downshift to a lower bitrate representation in response to the reduction in the throughput. If the subsequent segment for the lower bitrate representation is stored in the cache server, the observed throughput will increase. Consequently, an overestimation may be observed again. This undesired repeated cycle is known as ***bitrate oscillation***. Moreover, such bitrate oscillations will lead to undesirable quality switches, namely the selected segments will continuously change resolution, which is detrimental to QoE [13]. This oscillation can deplete the playback buffer and cause the client algorithm to take drastic measures to refill it at the expense of video quality. One way to solve this problem is to use a DASH algorithm that is cache-aware in order to benefit from the already cached video chunks. Those bitrate oscillation problems could be easily eliminated if the client foreknew which of the chunks are closest to him (cache hit) just so that the video bitrate could be increased, and which are furthest from him (cache miss), so that the video bitrate could be decreased.

4. Quality of Experience

4.1. HTTP Video Streaming Influenced by QoE Factors

HTTP video streaming is a combination of download and concurrent playback. It transmits the video data to the client via HTTP, where they are stored in an application buffer. After a sufficient number of data have been downloaded (i.e., the video file download does not need to be complete yet), the client can start playing out the video from the buffer. However, network issues are possible, and they will decrease the throughput and introduce delays at the application layer. Such network issues include packet loss, insufficient bandwidth, jitter and delay. Consequently, the client's buffer will not fill up as quickly as desired, so the video has to be interrupted until enough data have been received. These interruptions are known as stallings or rebuffering events, and they are one of the most important factors that influence QoE.

In general, the QoE influence factors can be categorized into technical and perceptual influence factors. The perceptual influence factors are directly perceived by the end user of the application and are dependent but decoupled from the technical development [13].

QoE in video streaming is a topic handled in the literature from various angles. One category is the proposal of SDN-based QoE experimental platforms for video streaming that integrate QoE assessment methods so that they support researchers in implementing QoE-aware management approaches [14,15]. Other works focus on the server selection problem using QoE criteria. For instance, in [16], reinforcement learning techniques are applied to the server selection problem to achieve optimal QoE in mobile video streaming, which is approached by considering the predicted QoE for each server as the "reward". Similarly, ref. [17] proposes a QoE-aware MEC-based peer-offloading scheme for DASH that is based on a reverse-fuzzified PSO algorithm for real-time QoE maximization. Finally, another line of work evaluates or proposes new QoE assessment methods/analytical models for video streaming. The goal is to quantify the impact of input parameters, such as quality switches, initial buffer level, maximum buffer and video characteristics, on QoE-relevant metrics for DASH-based scenarios [18,19].

4.2. QoE Metrics Related to Video Streaming

QoE is highly dependent on the application type. Since in our work we focus on DASH, we propose the use of the following metrics related to QoE:

- The **average resolution** refers to the average resolution over all segments that have been selected by the media player, where, as explained later, in our work each segment can take resolution values from the set {2160, 1440, 1080, 720, 480, 360}.
- The **sum of switches** refers to the sum of changes in resolutions that have occurred during the video playback.
- The **average altitude** refers to the average value of the jumps between the previous and the current segment over all segments that have been selected by the media player, wherein each segment can take altitude values from the set [0, 6]. For example, if the first segment is 480p and the second is 1440p, then the altitude, or the jumps, are 3.
- The **average video bitrate** (Mbps) refers to the average bitrate over all segments that have been selected by the media player, after which each segment size is divided by the segment duration, which in our work is 5 s. The common length of chunks ranges from 2 s (i.e., Microsoft Smooth Streaming) to 10 s per segment (recommended by Apple HTTP Streaming) [20]. We chose a compromise between these two values, selecting 5 s as the chunk length in our scenarios. For this reason, since our video is 5 min long, for each resolution, 61 segments are created. We avoid the two "extreme" values for the following reasons: low values: (a) having a very small chunk length would lead to far too many segments and (b) the chunk size would be too small to "stress" the network bandwidth-wise (small chunk size leads to low data rate requirements)/ high values: we would have just a few segments, not allowing for the fine interplay between segment selection (due to ABR) and caching.
- The **network usage time** (seconds) refers to the duration of the video playback, from the time of delivery of the first segment to the time of delivery of the last segment.
- The **mean opinion score (MOS) for stallings** refers to the numerical measure of the overall quality of an event, system or experience in networks and telecommunications. It is a crucial parameter for the domain of QoE and is expressed as a single rational number, typically in the range 1–5, where 1 is the lowest perceived quality and 5 is the highest perceived one. This MOS value, based on stallings, is computed with the formula [21]

$$MOS_{stallings} = 3.5 * e^{-(0.15M+0.19)N} + 1.5$$

where M = mean stalling duration and N = number of stallings.

- The **mean opinion score (MOS) for resolutions** has been defined by ITU-T in several ways. We will use the ITU-T P.1203 Standalone Implementation for the MOS based on resolutions. In order to use this tool, we install it from the official repository of GitHub [22]. The ITU-T Rec. P.1203.1 model is restricted to information provided to it by an appropriate bit stream analysis module or set of modules. The model applies to progressive download and adaptive streaming, where the quality experienced by the end user is affected by audio coding and/or video degradations due to coding, spatial re-scaling or variations in video frame rates, as well as delivery degradations due to initial loading delay, stalling and media adaptations. The model's input consists of information that is obtained through a prior stream inspection. There are four different modes of inspection, which result in models of different complexity. In our work, we use Mode 0, wherein the information is obtained from available metadata during the adaptive streaming (for example, from manifest files used in DASH about codec and bitrate, and initial loading delay and stalling). Due to our mode, we do not have score fluctuations among the same resolution.
- The **number of stallings** refers to the sum of stalling events. A stalling occurs when a segment has been delivered in more than 5 s, which is its duration.
- The **total stalling time** refers to the sum of stalling durations of all the segments.
- The **mean stalling time** refers to the total stalling time over all stallings that occurred in a video playback.

- The **initial playback delay**, in seconds, refers to the time of delivery of the first segment from the time of the arrival of the manifest file, which is the start of the experiment.
- The **number of cached bits** delivered refers to the sum of the size in bits of the segments that were already stored in the cache.
- The **number of non-cached bits** delivered refers to the sum of size in bits of the segments that were not stored in the cache, but were served by the main server.
- Last, the **cache hit ratio** refers to the sum of segments that were already in the cache over all the segments cached and non-cached, as a percentage.

5. Experimental Setup

5.1. Network Topology—Scenario

The main actors in our implementation are three, as shown in Figure 4:

(1) The **client**, who wants to see a video at the best available quality.
(2) The **proxy or cache server** (terms used interchangeably), which has stored some of the video segments according to a caching algorithm, which limited by its buffer size, as well as the manifest file.
(3) The **main server**, which is equipped with all the video resolution chunks.

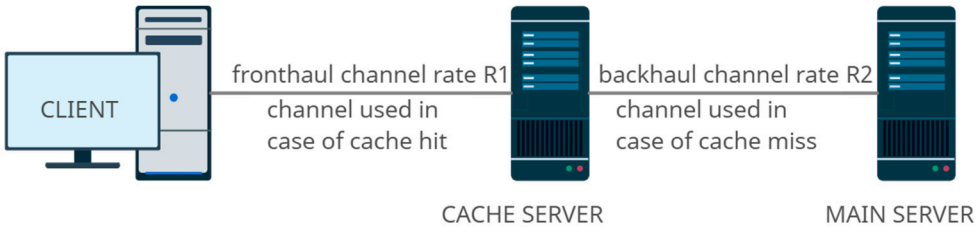

Figure 4. Implementation with an intermediate cache server.

The cache server is designed to be a simple repository for our video content, providing local clients with accelerated access to cached files. The main server is intended to provide a content management service which runs entirely behind the user interface of our custom application, creating the backhaul connection with the cache server, and containing all the given resolutions and the coding rates.

Specifically, the cache server will contain a few segments from some of the video resolutions, meaning that it will have some caching capabilities. So, the basic connection will be between the client and the cache server. The backhaul connection, between the cache and the main server, will be utilized so the cache server will be able to receive chunks with a fixed rate from the main server, which contains all the segments. The number of chunks that will be stored in the cache server depends on its available buffer size L and the implemented caching algorithm. The link rate between the client and cache will be referred to as R_1 and the link rate between the cache and main server as R_2 (see Error! Reference source not found.). Specifically:

- The **mean network throughput R_1** or **"Effective R_1"** (Mbps) refers to the size of all the segments (cached and non-cached) that have been selected by the media player over the network usage time (fronthaul rate).
- The **mean network throughput R_2** or **"Effective R_2"** (Mbps) refers to the size of all the segments not stored in the cache over the network usage time (backhaul rate).
- In this sense, the **backhaul/fronthaul traffic ratio R_2/R_1** is a parameter which may be adjusted by the mobile network operator [23].

The exact roles of the three aforementioned actors in our problem statement are the following:

- **Client**: The client requests a video. The video has to be displayed at the best resolution based on the available bandwidth. Practically, the client requests the manifest file from

the cache server, which contains all the available video details. For this reason, the cache must have available information about the manifest.

- **Proxy/cache server**: The proxy server has some chunks of the video in specific resolutions according to its caching policy. It interacts both with the client at one end and with the main server at the other end. The cache server needs to set up a session with the main server that has all the video chunks in order to cache as many chunks as it can. As long as the segments are stored in the cache, the client requests the video and a new fronthaul connection with the cache server is set up. Each time a request is received, the cache checks the cache buffer and if there is a ***cache hit***, the cache server sends it directly to the client and the channel cache-main is not utilized at all. Otherwise, in the case of a ***cache failure***/miss, the cache asks from the main wanted chunk in order to serve the client's request.
- **Main (VoD) server**: The main server has all the possible resolutions of the video (360p, 480p, 720p, 1080p, 1440p, 2160p) that are defined by the manifest file. This server interacts only with the cache server providing specific video chunks. In our setup, its role is only to give the manifest file to the cache (only in cases wherein the cache is not already equipped with it) and serve the client when there is a cache miss. In case the cache has all the available chunks of the video, meaning that it can fully serve the client, the backhaul link is not used at all.

The flow of information between these nodes is depicted in Figure 5.

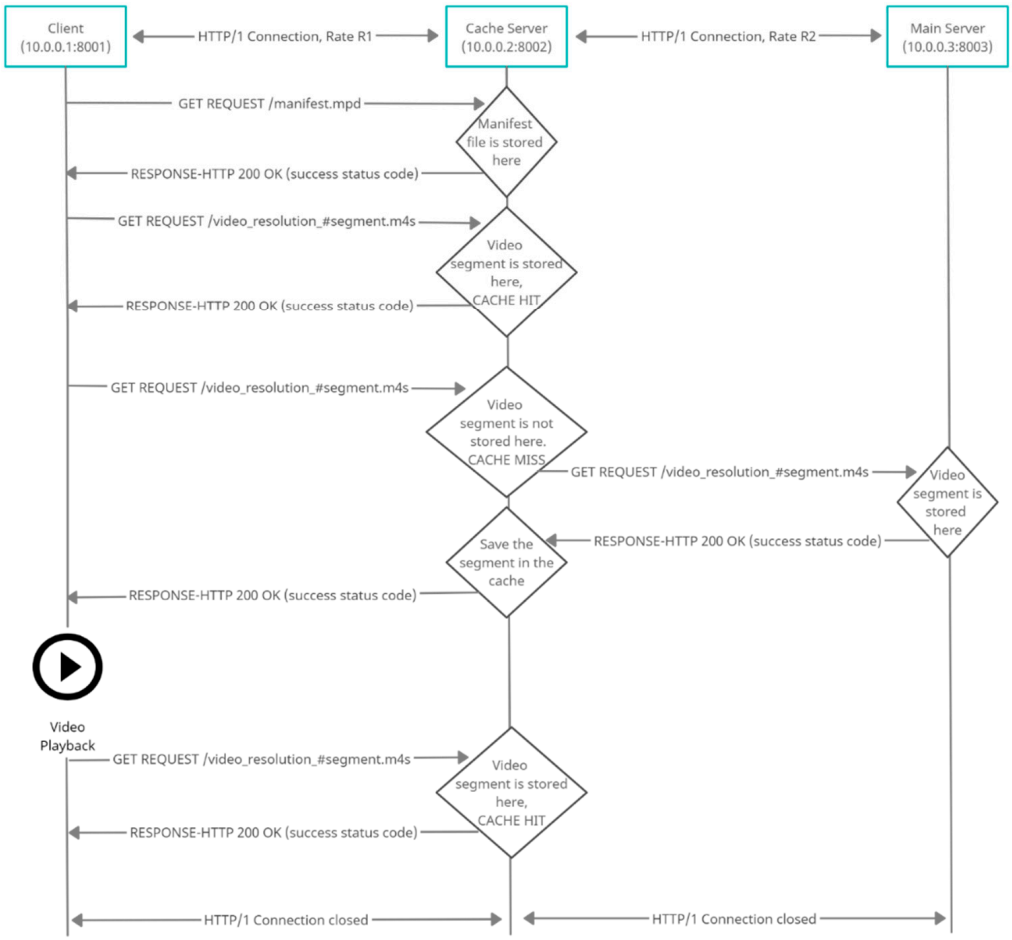

Figure 5. Flow of information.

The proxy server is the cache of our topology with the available buffer sizes of 0, 100, 1000 and 3000 MBs, and the main server is the database that contains all the segments of all the different resolutions, 3000 MBs in total.

5.2. Content Preparation

For our experiments, we used a video with a resolution of 3840 × 2160 (4K-UHD), which we recorded with a mobile phone. The video size was 1.8 GB (we employed a large file in order to deliberately stress the network simulations). We created a bash.sh script, which converts every mp4 file in the current folder and subfolder to a multi-bitrate video in MP4-DASH. This script created six versions of the video of varying quality, an audio file and an mpd manifest file with all the appropriate information about the given resolutions. Then, we saved the name of every file without the file's extension, and we started the procedure of converting each file to a multi-bitrate video in MPEG-DASH.

The first step was to convert the audio file with the help of the FFmpeg commands for the audio and video codec.

Next, we converted the video file to six different-quality versions. The video codec that was used was H.264. We know that in this codec, the frames are organized in I, P and B frames, and that I frames appear every 30 frames. A series of grouping frames is called a GOP (Group of Pictures). The first frame in a GOP is an intra-frame called an I frame, which contains most of the vital information for the following sequence of P frames, which are forward-predictive, and B frames, which are both forward- and backward- (bi-directional) predictive. A B frame estimates changes to the frame based on the previous and following frames. I frames contain more data than P and B frames.

Finally, we obtained all the required video and audio files, and using the MP4Box tool, we produced the following files:

- A manifest-mpd file, which stored all the needed information about the video. This file contained all the available video resolutions, their bitrates and their size. In our implementation, a client, who wants to see a specific video, requests the manifest file.
- All the video segments had a duration of 5 s for all the available resolutions and audio. For this reason, because our video was 5 min long, for each resolution, 61 segments were created (5 min = 300 s and 300 s/5 s = 60 segments, and 1 more segment for initialization).
- An initialization file in order to start loading the segments (manifest_set1_init.mp4).

Table 1 below presents the video bitrates and respective sizes after the encoding and segmentation processes.

Table 1. Video bitrates and sizes after the segmentation.

Resolution	Video Bitrate (Mbps)	Size (MBs)
2160p	45	1700
1440p	16	597.6
1080p	8	299.1
720p	5	187.3
480p	2.5	94.1
360p	1	37.7

5.3. Caching Algorithm

An important factor in our video streaming environment is the caching policy. For our work, we chose "random caching" for performance benchmarking. In this method, the proxy downloads segments of all possible resolutions and caches them until the cache capacity is fully exploited.

The algorithm generates two random numbers, one for the resolution and one for the segment. So, the resolution can be chosen to be 360, 480, 720, 1080, 1440 or 2160, and the range to choose for the segment is [1, 61], because each resolution has 61 segments. After randomly choosing the file, it checks if it fits into the buffer. If it fits, it copies it in from

the main server's public folder. If it does not, it moves to the next one only if the available buffer size is less than the size of the smallest available non-cached segment.

6. Experimental Study Results

6.1. Set of Experiments

For the purposes of emulation and analysis, we conducted two different sets of experiments. The first set, given the name "***Get Before Send***", or GBS for short, is an approach wherein the data from the server are fetched serially, meaning that the second segment is sent to the client only after the first segment has been fetched. In the case of GBS, if a cache hit takes place, the proxy server forwards the video segment (locally cached) directly. If a cache failure takes place, the proxy downloads the full video segment before forwarding it to the client. This approach will inevitably cause the traffic R_1 that we applied in the fronthaul channel and the traffic R_2 that we applied in the backhaul channel to be under-utilized.

The second set of experiments, named "***Send While Get***", or SWG, as the name suggests, is more efficient than the first; the segments are fetched from the server simultaneously and are sent to the client without the need for any queue. In this approach, given the fact that it is smarter than the first, we expected the channels to be utilized in a better way, leading us to better results for all the QoE metrics, which we exported after conducting a variety of experiments.

Therefore, in the SWG technique, the proxy server does not wait for a single segment to be delivered; rather, while a segment is sent, the next one is already beginning the process of being requested and delivered. If a cache hit takes place, the proxy server forwards the video segment (locally cached) directly. If a cache failure takes place, we configured the proxy server to read the proxy-to-main socket packet per packet and forward the packets directly upon reception.

In the case of SWG, there are no stallings observed, which means that the QoE is increased for the user. Also, the bitrate oscillations are fewer as compared to in GBS, resulting in the client better exploiting the cache hits. However, the most significant observation is that both the fronthaul and the backhaul channels are utilized to a better extent, due to the fact that no channel stays idle for a long period of time.

6.2. Simulation/Emulation/Programming Tools

In the following Table 2, we present the tools and libraries that we used to set up our experiments in a Software-Defined Networking (SDN) environment, and we explain their roles.

Table 2. Tools used for experimentation.

Tool	Purpose
OpenJDK	Java Development Kit
Mininet (only in Part C)	Network emulator
MiniEdit (only in Part C)	Mininet's graphical interface
VLC media player	Streaming media server
FFmpeg	A suite of libraries and programs for encoding and handling video
Python	Programming language, to configure a web video server
MP4Box—GPAC	Multimedia packaging (segments a video file and creates a Media Presentation Description (MPD))

Concerning the adaptive bit rate algorithm used, this is selected in VLC, as shown below (Figure 6). "Bandwidth Adaptive" means that the segment request decision is taken based on the rate at which the previous segments were actually received.

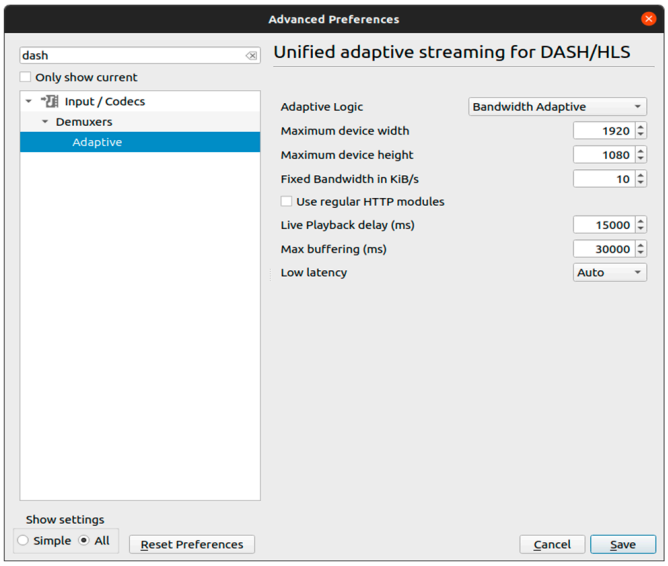

Figure 6. VLC preferences.

6.3. Mininet Experiments

We first attempted to conduct our evaluations using Mininet in an SDN setting, with the topology shown below (Figure 7).

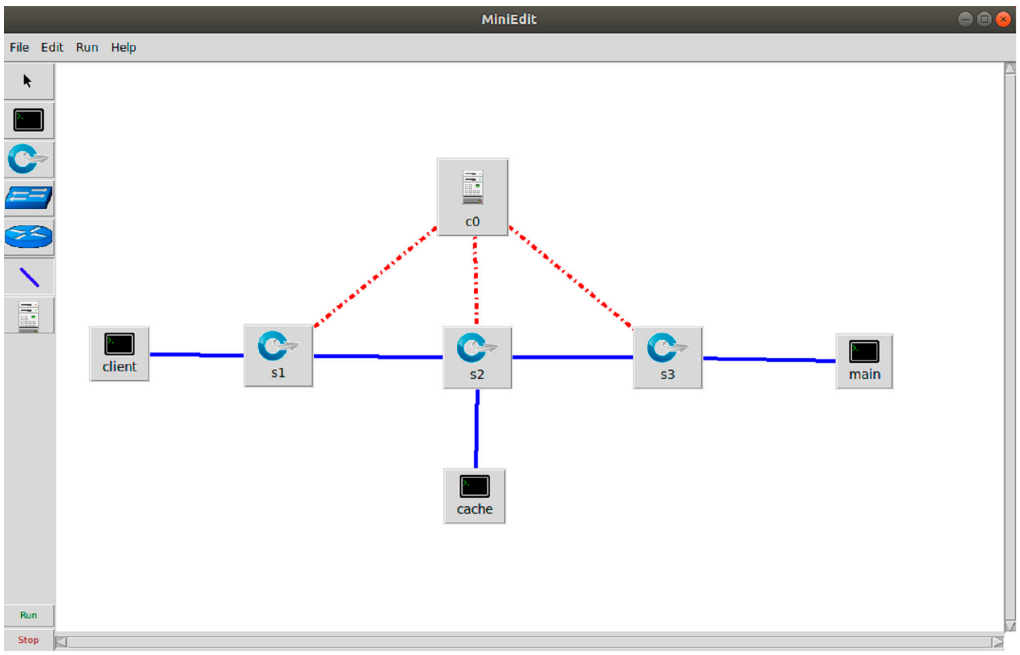

Figure 7. Enhanced topology with cache server.

For our simulations, we created the cache and the main server (shown in Figure 7). The main server contains the manifest file as well as every segment for each quality. On the other hand, the cache folder contains the manifest file, and some video segments according to the appropriate caching policy.

We started running the main server as well as set up the cache server to listen for any requests. Finally, we started the video player in the client's Xterm window in order to request the manifest file from the cache server.

The procedure was then as follows: The video starts playing due to a request for the manifest from the cache server. Then, sequential requests for each segment move towards the cache server. The start always has the lowest quality, so the initialization file is asked first, and the first segments (m4s files) are asked second (those files are all in the cache). In the case a file is requested by the client from the cache and the cache does not have it (cache miss), the main server is enabled, and the cache requests it from it.

After conducting many experiments with different combinations of (R_1, R_2, buffer size L), we concluded to the fact that this approach had many drawbacks that led to a result that did not meet realistic scenarios. The restrictions that we met were the following:

- The channels were not utilized to a full extent even though there were enough resources available. This problem persists mostly due to the limitations of the Mininet interface but also in the case of the Get Before Send technique.
- The rate control was not accurate. We noticed that the Mininet interface had a very high default bandwidth value (Gbps). What we observed was that while the rate was changing in our scripts, due to the application of one rule after the other, there was a gap between these rules. During this gap, the default value of the Mininet interface was applied, as it was the dominant one. This resulted in a higher data rate than the one anticipated by the experiment setup.
- The Mininet interface restricted the automation of this procedure.

In the next sections, we present first the QoE metrics related to DASH as a function of the fronthaul channel R1 and as a function of the cache buffer size L.

6.4. QoE Metrics vs. R_1 Throughput

Given the limitations explained previously, our new approach was implemented in regular gnome terminals with the IPs of the localhost 127.0.0.1 (Figure 8). In our experiments, we changed the rate that is applied to the fronthaul channel with the Rayleigh distribution per second.

Figure 8. Revised topology.

In Figures 9–11, we can observe how the average resolution improves with the increase in the buffer size L for different values of the fronthaul channel R_1.

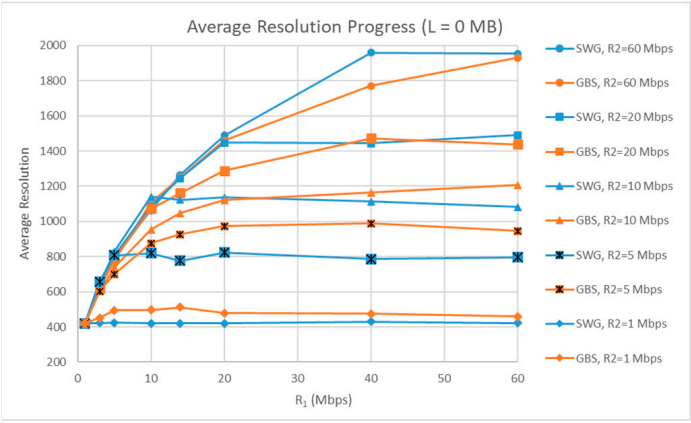

Figure 9. Average resolution vs. R_1 (L = 0 MB).

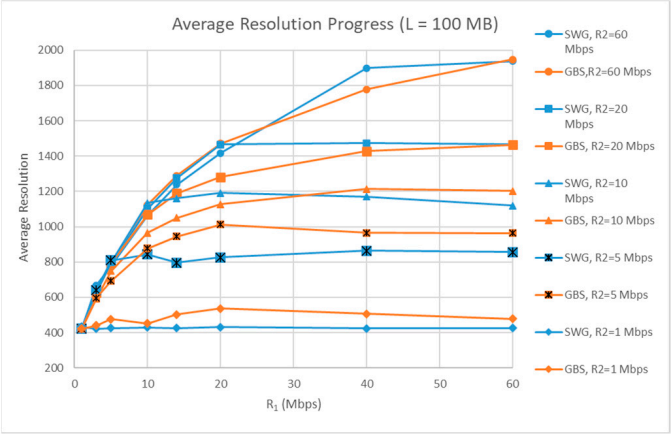

Figure 10. Average resolution vs. R_1 (L = 100 MB).

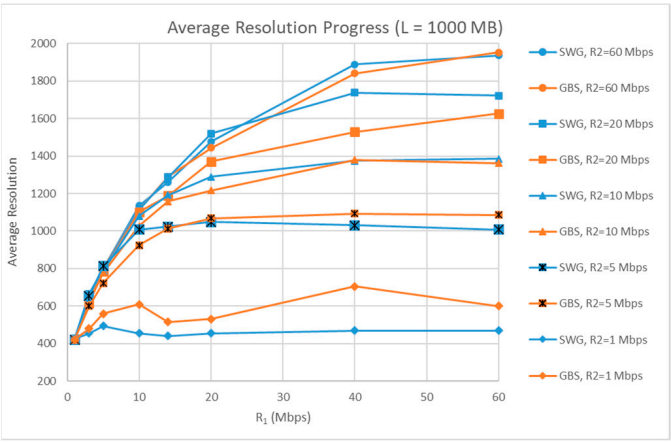

Figure 11. Average resolution vs. R_1 (L = 1000 MB).

From these graphs, we infer the following conclusions: First of all, the higher the backhaul R_2 (for the same R_1), the higher the average resolution playing for the client. Moreover, we noticed a logarithmic increase in the average resolution at every curve of the figure, while $R_2 < R_1$. When $R_2 \geq R_1$ is valid, the value of the quality of the video stabilizes and cannot be further improved. For R_2 = 1, 5 or 10 Mbps, a false image is given, so the GBS approach gives better results. In this case, the DASH algorithm is over-optimistic, making the media player increase the quality. However, this increase is not justified in terms of bandwidth, and this is the reason for the continuous occurrences of bitrate oscillations and, as a result, stallings. On the other hand, the SWG approach has fewer stallings than the GBS one, so it is the one that gives better results in terms of QoE. The right approach, showing the dominance of the SWG technique, is mostly shown in the curves of the figures for R_2 = 20 and 60 Mbps, regardless of buffer size. Furthermore, when the buffer size L is increased, we can see an increase in the average resolution for the same R_1 and R_2 values. The positive impact of caching is clearly noticeable. To make the example more understandable, in the case of an R_1 = 20 Mbps and R_2 = 10 Mbps, if the buffer size L = 0, then the average resolution ranges between 1100 and 1200, but if the L = 1000, then the average resolution is increased at about 100 units, reaching a value close to 1300.

In Figures 12–14, we can see the average altitude for two distinct values of R_1 in the three different buffer sizes that we had available in our experiment.

Figure 12. Average altitude vs. R_1 for R_2 = 5 and 20 Mbps (L = 0 MB).

Figure 13. Average altitude vs. R_1 for R_2 = 5 and 20 Mbps (L = 100 MB).

Figure 14. Average altitude vs. R_1 for R_2 = 5 and 20 Mbps (L = 1000 MB).

The peak of the curve in the figure for R_1 = 5 Mbps is a good example of the over-optimism of the DASH algorithm, which causes many bitrate oscillations, leading to the increase in the altitude value. If the channel is of a higher rate, this impact is eliminated, and the average altitude reaches very low values. In order to fully extinguish the impact and cause it to reach lower values, a higher R_2 is needed, as shown by comparing the figures above. Moreover, for a higher R_2, the SWG approach remains the best option. Finally, the increase in the buffer size does not demonstrate any big difference in the behaviour of the curve in the figure. A quite general conclusion is that the higher the buffer size L, the lower the average altitude.

In the next experiment (Figures 15–17), we studied the relationship between the reserved R_1 and the actual mean network throughput R_1 (called effective R_1). There is an important distinction between these two terms, as they are not identical: reserved R_1 represents the performance guarantees from the fronthaul link (reserved capacity), while the effective R_1 quantifies the actual rate achieved (and measured using this evaluation study). Thus, with the Y axis, we plotted the actual R_1 recorded/experienced by the end user in our experiments, whereas with the X axis, we tracked the reserved R_1 capacity referring to the nominated a priori throughput that was reserved between the HTTP client (video consumer) and the HTTP proxy base station. We observed the following for buffer sizes 0, 100 and 1000 MB:

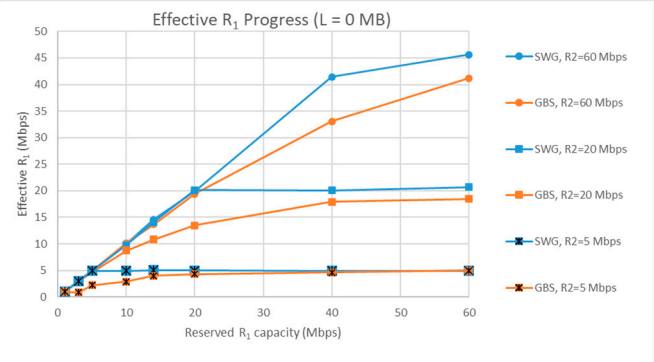

Figure 15. Effective R_1 vs. reserved R_1 capacity for L = 0 MB.

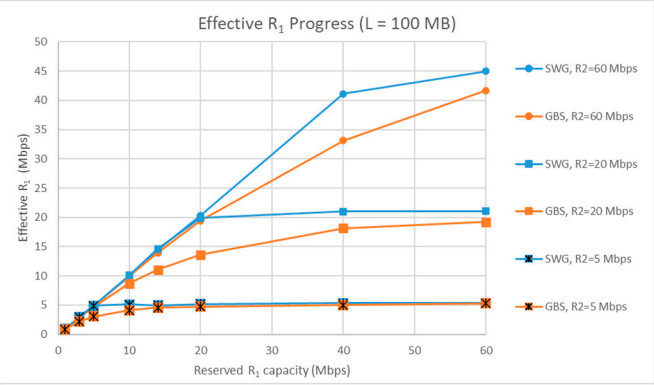

Figure 16. Effective R_1 vs. reserved R_1 capacity for L = 100 MB.

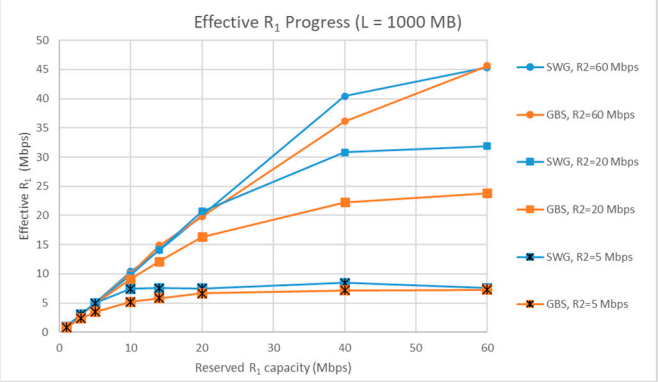

Figure 17. Effective R_1 vs. reserved R_1 capacity for L = 1000 MB.

The results are mostly as expected, as the throughput is increased linearly until the point where $R_1 = R_2$, where it reaches the highest value. For R_2 = 60 Mbps, the throughput is not 60 as expected, but it is lower because the highest available quality (2160p) has an average bitrate of almost 45 Mbps. Also, the higher the buffer size, meaning that the fronthaul channel R_1 is used more often than the backhaul, the higher the mean throughput R_1. Another useful insight is the better utilization of the channels in the SWG approach, especially when the buffer size is higher, while in the GBS approach, the network resources remain unutilized, something that is prohibited for any vendor. Finally, the gap shown in the diagram between the two approaches for the same R_2 value reaches higher values of R_2. The R_2 increase is responsible for the change in place of the gap in the right direction.

Next, in Figures 18 and 19, we validate the behaviour of the backhaul R_2 for buffer sizes 100 MB and 1000 MB.

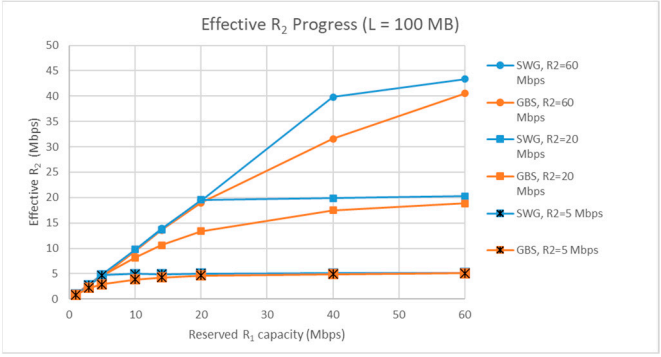

Figure 18. Effective R_2 vs. reserved R_1 capacity for L = 100 MB.

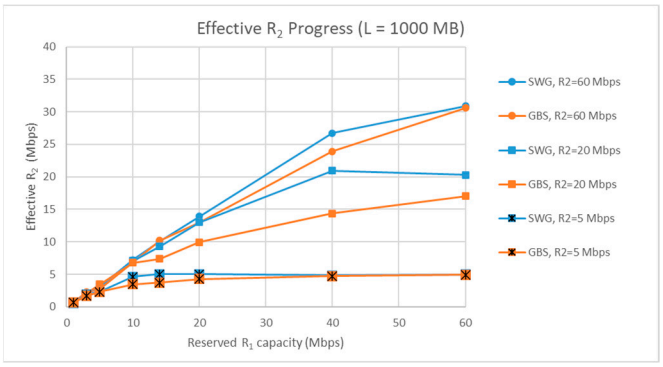

Figure 19. Effective R_2 vs. reserved R_1 capacity for L = 1000 MB.

We can see that the curve is linear until it reaches an upper limit, and that, again, better utilization of the channel is shown in the SWG approach compared to the GBS approach, which leaves resources even more unutilized. However, the use of a non-smart algorithm and the non-existence of a multithread approach makes the backhaul channel idle for a long time, in the case of a cache hit. Ideally, in case of a cache hit where the fronthaul channel would serve the client, the backhaul channel should download new segments to the cache in order to benefit later, in terms of QoE and network resources. Furthermore, the SWG approach is generally more beneficial, but the biggest benefit comes in the points from R_1 = 0 to 10 Mbps, where there is a range of available resolutions. When the R_1 is higher than those values, there is no more benefit, due to the absence of another higher resolution. (Note: R_1 = 0 is not meaningful per se, but shows the tendency immediately after the fronthaul rate is changed to >0.)

The MOS that a client may extract if he watches a video is depicted below for three different kinds of cache servers (Figures 20–22).

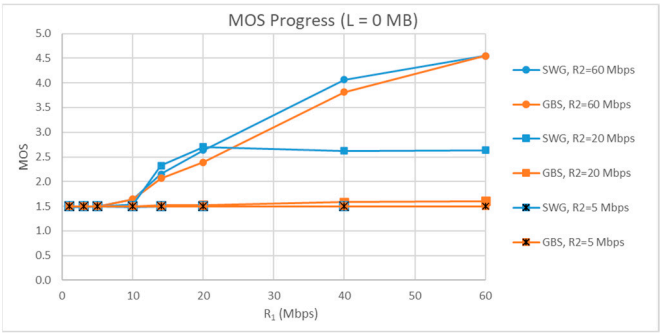

Figure 20. MOS vs. R_1 for L = 0 MB.

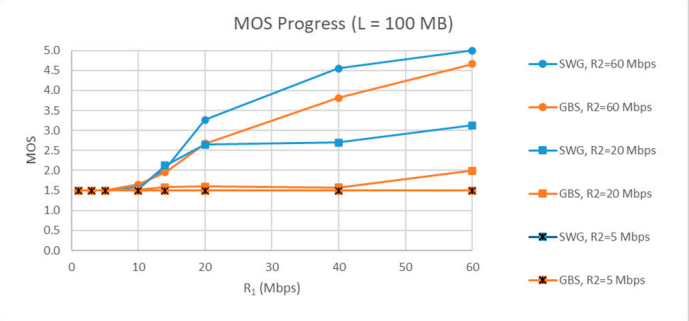

Figure 21. MOS vs. R_1 for L = 100 MB.

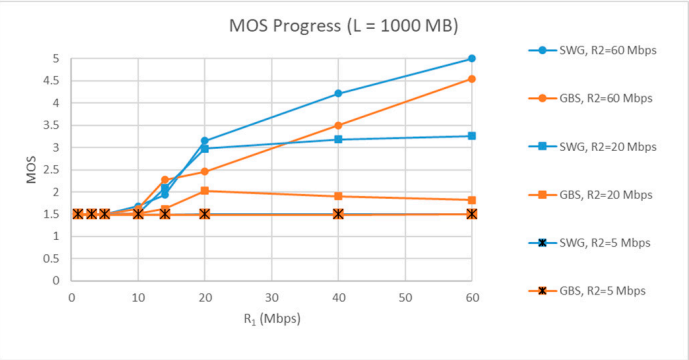

Figure 22. MOS vs. R_1 for L = 1000 MB.

We can conclude that the MOS metric demonstrates the client's preference for the SWG approach. The DASH algorithm, due to the fact that it sometimes acts over-optimistically for the available bandwidth, leads to the video freezing many times (stalling), and, as a result, the client is not satisfied. The bottom line is that the caching algorithms and the DASH algorithms are always ***interdependent*** because if the DASH algorithm is not cache-aware, then caching does not help very much. In other words, we are wasting resources for storage, without it being beneficial. In more detail, the higher the R_2 value, the less severe the bottleneck, so there is a benefit in the MOS value. For R_2 = 5, so many stallings occur that they lead to the worst value of the MOS. As a result, the approaches do not differ that much in terms of what the client feels while watching the video. (Note: In Figures 21 and 22 the SWG and GBS for R_2 = 5 Mbps coincide.)

Lastly, as regards the number of stallings in our experiments, we conclude with the following results (Figures 23–25).

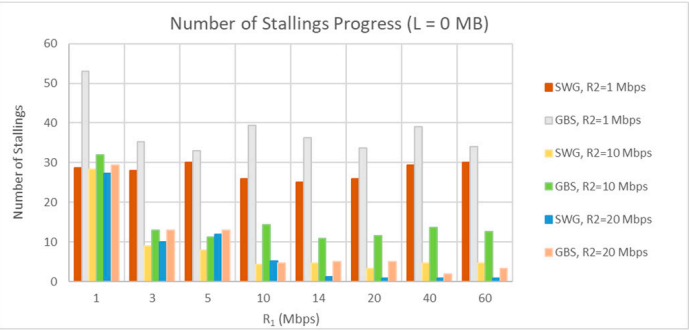

Figure 23. Number of stallings vs. R_1 for L = 0 MB.

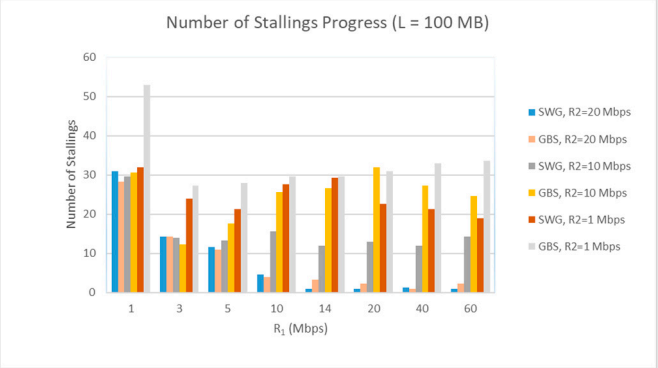

Figure 24. Number of stallings vs. R_1 for L = 100 MB.

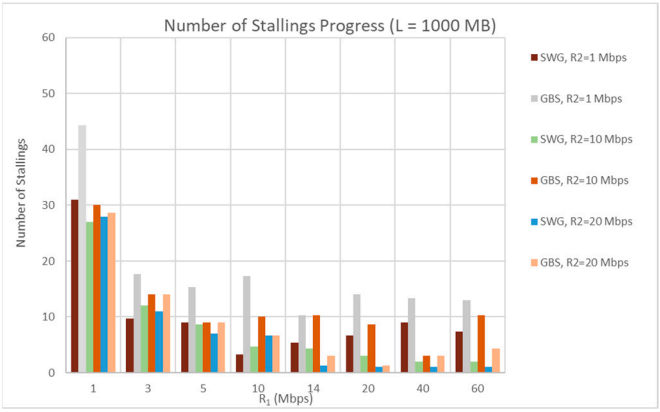

Figure 25. Number of stallings vs. R_1 for L = 1000 MB.

To highlight a clear conclusion, it is possible that more samples were needed. Generally, there is a tendency for stallings to decrease while the R_1 increases and while the buffer increases, since more segments are found cached and the media player uses them to its advantage to avoid freezing. For a stable value of R1, there are more stallings for lower values of R2. The GBS approach, compared to the SWG, presents more stallings in the majority of cases. If we had more samples, the GBS approach would always have more stallings. Finally, the existence of a larger buffer is beneficial in terms of stallings, mostly in cases of higher R_1 values. In cases of lower R_1 values, there is also an improvement in terms of stallings, but it is a smaller improvement compared to that achieved with higher R_1 values.

6.5. QoE Metrics vs. Buffer Size L

The average resolution progress as the buffer size is increased is shown in Figure 26.

We can observe that the increase in the buffer size creates a satisfying profit over resolution in the case of a larger R_1. If R_1 is low, there is a slightly smaller increase, or no increase at all, meaning that caching is not always beneficial, but should be accompanied by high-speed channels. Additionally, the SWG approach is much better than the GBS approach.

The backhaul throughput progress over buffer size is shown below (Figure 27).

From this diagram, we can conclude the utilization of the channels, especially in the GBS approach. This is demonstrated more clearly in the case of the buffer size being 3000 MB, wherein all the segments are cached and the R_2 throughput is zero. Although this is acceptable for lower R_2 values, it is a big waste of resources for larger R_2 values. If we were dealing with a multithread approach or a cache-aware DASH algorithm, then the curve would not fall progressively to zero, but the available R_2 would be used to download even more segments proactively, optimizing the QoE of the client.

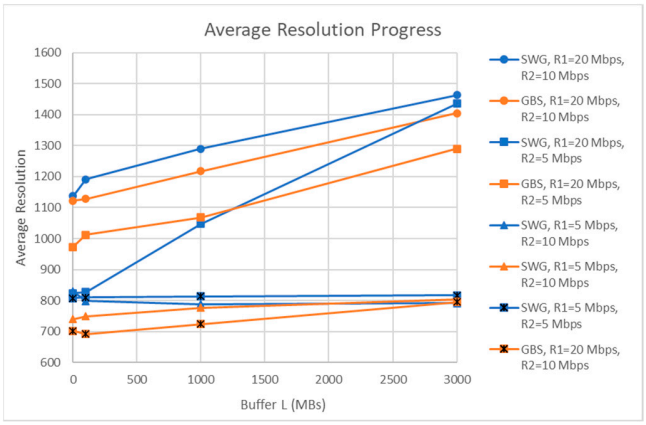

Figure 26. Average resolution vs. buffer size L.

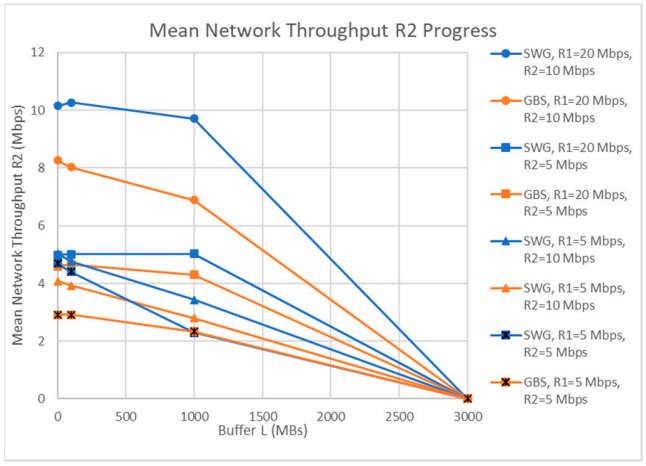

Figure 27. Mean network throughput progress vs. buffer size L.

Now, if we compare the number of stallings that have occurred for two different fronthaul rate values, we can say that in both cases, the GBS approach leads to more stallings than the SWG approach. This is clear for R_1 = 20. Moreover, higher R_1 values can benefit to a larger extent from the increase in the buffer size and decrease the stallings in those cases. If the value of the R_1 is low, there is no reason for any vendor to increase the available resources and provide a larger buffer size, since, at the end of the day, the QoE perceived by the client will not show many differences from the stallings perspective. Stallings will still occur, and they will annoy the client (Figures 28 and 29).

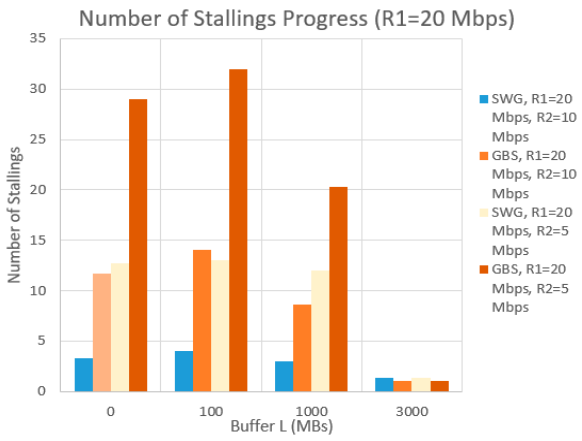

Figure 28. Number of stallings vs. buffer size R_1 = 20 Mbps.

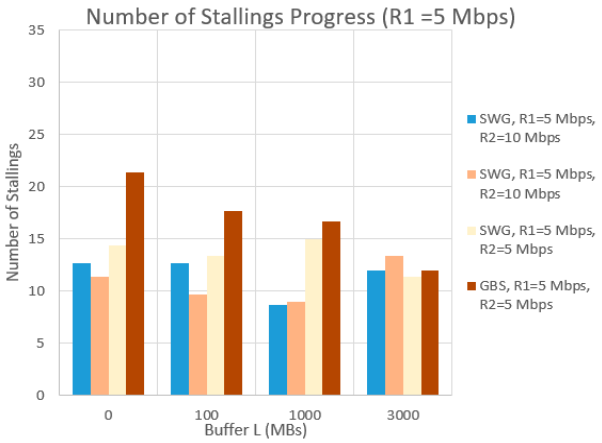

Figure 29. Number of stallings vs. buffer size for R_1 = 5 Mbps.

7. Conclusions

In this paper, we examined how DASH-based content can be delivered with the help of an intermediate server, apart from the main server, in order to demonstrate possible caching benefits for different sizes of intermediate storage servers. The goal was to exploit and take full advantage of the available bandwidth, along with the advantages of the caching approach. For this reason, we created a simple client-cache-main topology to implement HTTP adaptive logic, wherein if the cache server contained some segments, it served them immediately. Otherwise, the main server delivered the missing segments. We compared the basic GBS logic of exchanging requests with a slightly smarter SWG approach, so that it could serve some requests at the same time as requesting the next segment. We aimed to increase the performance of our model to some extent and, consequently, the QoE of the client while watching the video.

The two main findings of our research are that SWG logic enhances the performance of almost all QoE-wised metrics, especially when accompanied by a satisfactory caching policy, and reduces the under-utilization that we observed in our initial approach. We also noticed that the chosen DASH algorithm played an important role in our experiments due to the "dangers" hidden by its over-optimism regarding which segment (in terms of resolution) will be the next to be played. So, if the media player is not aware of what is available on the intermediate server, this may lead to many stallings happening in the experiment, resulting in the loss of any benefit we had due to caching.

Finally, the caching algorithm that we used in all the implementations, which is responsible for filling the buffer of the intermediate server, is random and is not advanced in its method of filling the cache with segments. An implementation of a smarter algorithm for future work is proposed, one which is not based on randomness, but on the possibility of playing certain segments instead of others. For example, if the channels had a traffic of 5 Mbps, then there would be no reason to cache the segments of the 4K resolution, but certainly those smaller than that. This would highlight the assistance provided by the caching policy to all video streaming services.

Author Contributions: Conceptualization & Methodology, E.L. and D.X.; Software & Validation, E.L., V.G. and G.K.; Writing—review & editing, L.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was co-financed by Greece and the European Union (European Social Fund—ESF) through the Operational Programme «Human Resources Development, Education and Lifelong Learning» in the context of the project "Reinforcement of Postdoctoral Researchers-2nd Cycle" (MIS-5033021), implemented by the State Scholarships Foundation (IKY).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jonsson, P. (Ed.) Ericsson Mobility Report. November 2022. Available online: https://www.ericsson.com/4ae28d/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-november-2022.pdf (accessed on 20 June 2023).
2. Le Callet, P.; Möller, S.; Perkis, A. *Qualinet White Paper on Definitions of Quality of Experience*; European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003): Lausanne, Switzerland, 2013; Version 1.2.
3. Dao, N.N.; Ngo, D.T.; Dinh, N.T.; Phan, T.V.; Vo, N.D.; Cho, S.; Braun, T. Hit Ratio and Content Quality Tradeoff for Adaptive Bitrate Streaming in Edge Caching Systems. *IEEE Syst. J.* **2021**, *15*, 5094–5097. [CrossRef]
4. Guo, Y.; Yang, Q.; Yu, F.R.; Leung, V.C.M. Cache-Enabled Adaptive Video Streaming Over Vehicular Networks: A Dynamic Approach. *IEEE Trans. Veh. Technol.* **2018**, *67*, 5445–5459. [CrossRef]
5. Li, C.; Toni, L.; Zou, J.; Xiong, H.; Frossard, P. QoE-Driven Mobile Edge Caching Placement for Adaptive Video Streaming. *IEEE Trans. Multimed.* **2018**, *20*, 965–984. [CrossRef]
6. Huang, D.; Tao, X.; Jiang, C.; Cui, S.; Lu, J. Trace-Driven QoE-Aware Proactive Caching for Mobile Video Streaming in Metropolis. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 62–76. [CrossRef]
7. Bayhan, S.; Maghsudi, S.; Zubow, A. EdgeDASH: Exploiting Network-Assisted Adaptive Video Streaming for Edge Caching. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 1732–1745. [CrossRef]
8. Liotou, E.; Samdanis, K.; Pateromichelakis, E.; Passas, N.; Merakos, L. QoE-SDN APP: A Rate-guided QoE-aware SDN-APP for HTTP Adaptive Video Streaming. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 598–615. [CrossRef]
9. Hayamizu, Y.; Goto, K.; Bandai, M.; Yamamoto, M. QoE-Aware Bitrate Selection in Cooperation with In-Network Caching for Information-Centric Networking. *IEEE Access* **2021**, *9*, 165059–165071. [CrossRef]
10. Tran, A.-T.; Dao, N.-N.; Cho, S. Bitrate Adaptation for Video Streaming Services in Edge Caching Systems. *IEEE Access* **2020**, *8*, 135844–135852. [CrossRef]
11. Georgara, V.D.; Kourouniotis, G.I. Cache-Aware Adaptive Video Streaming in 5G Networks. Master's Thesis, National and Kapodistrian University of Athens, Athens, Greece, 2021. Available online: https://pergamos.lib.uoa.gr/uoa/dl/frontend/file/lib/default/data/2944340/theFile (accessed on 20 June 2023).
12. Lee, D.H.; Dovrolis, C.; Begen, A.C. Caching in HTTP Adaptive Streaming: Friend or Foe? In Proceedings of the Network and Operating System Support on Digital Audio and Video Workshop, Singapore, 19 March 2014; pp. 31–36.
13. Seufert, M.; Egger, S.; Slanina, M.; Zinner, T.; Hoßfeld, T.; Tran-Gia, P. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Commun. Surv. Tutorials* **2015**, *17*, 469–492. [CrossRef]
14. Wang, M.; Wang, Y.; Liu, Y. SDN-Based QoE Evaluation Methods for HTTP Adaptive Video Streaming. In Proceedings of the 2021 7th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC), Beijing, China, 17–19 November 2021; pp. 220–224. [CrossRef]
15. Ahmad, A.; Floris, A.; Atzori, L. Timber: An SDN-Based Emulation Platform for Experimental Research on Video Streaming. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 1374–1387. [CrossRef]
16. Tapang, D.K.; Huang, S.; Huang, X. QoE-Based Server Selection for Mobile Video Streaming. In Proceedings of the 2020 IEEE/ACM Symposium on Edge Computing (SEC), San Jose, CA, USA, 12–14 November 2020; pp. 435–439. [CrossRef]
17. Taha, A.-E.M.; Ali, N.A.; Chi, H.R.; Radwan, A. MEC Resource Offloading for QoE-Aware HAS Video Streaming. In Proceedings of the ICC 2021—IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; pp. 1–5. [CrossRef]
18. Schwarzmann, S.; Zinner, T. Linking QoE and Performance Models for DASH-based Video Streaming. In Proceedings of the 2020 6th IEEE Conference on Network Softwarization (NetSoft), Ghent, Belgium, 29 June–3 July 2020; pp. 65–71. [CrossRef]
19. Seufert, A.; Wamser, F.; Yarish, D.; Macdonald, H.; Hoßfeld, T. QoE Models in the Wild: Comparing Video QoE Models Using a Crowdsourced Data Set. In Proceedings of the 2021 13th International Conference on Quality of Multimedia Experience (QoMEX), Montreal, QC, Canada, 14–17 June 2021; pp. 55–60. [CrossRef]
20. Lederer, S. Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length. 2015. Available online: https://bitmovin.com/mpeg-dash-hls-segment-length (accessed on 20 June 2023).
21. Hoßfeld, T.; Schatz, R.; Biersack, E.; Plissonneau, L. Internet Video Delivery in YouTube: From Traffic Measurements to Quality of Experience. *Data Traffic Monit. Anal. Meas. Classif. Anom. Detect. Qual. Exp.* **2013**, *7754*, 264–301.
22. ITU-T, P. 1203: Parametric Bitstream-Based Quality Assessment of Progressive Download and Adaptive Audiovisual Streaming Services over Reliable Transport. Available online: https://github.com/itu-p1203/itu-p1203 (accessed on 20 June 2023).
23. Mehrabi, A.; Siekkinen, M.; Ylä-Jaaski, A. QoE-Traffic Optimization Through Collaborative Edge Caching in Adaptive Mobile Video Streaming. *IEEE Access* **2018**, *6*, 52261–52276. [CrossRef]