UNIVERSITY OF BERN

BACHELOR THESIS

Video Delivery with Multi-Access Edge Computing

Author:
Remo RÖTHLISBERGER

Mentor:
Dr. Eryk SCHILLER

Supervisor:
Prof. Dr. Torsten BRAUN

A thesis submitted in fulfillment of the requirements for the degree of B. Sc. in Computer Science

in the

Communication and Distributed Systems (CDS)
Institute for Computer Science (INF)

May 22, 2018

Declaration of Authorship

I, Remo RÖTHLISBERGER, declare that this thesis titled, "Video Delivery with Multi-Access Edge Computing" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

UNIVERSITY OF BERN

Abstract

Faculty of Science

Institute for Computer Science (INF)

B. Sc. in Computer Science

Bachelor Thesis: Video Delivery with Multi-Access Edge Computing

by Remo RÖTHLISBERGER

This bachelor thesis designs, implements, and evaluates video content delivery in mobile networks using Multi-Access Edge Computing (MEC). We provide a Video Streaming Service on the network edge, which resides close to a typical evolved Node B (eNB) of the Long Term Evolution (LTE) mobile network. The Video Streaming Service delivers video content towards mobile video consumers. Using a Radio Network Information Service (RNIS), we equip the Video Streaming Service with information about the momentary state of wireless channels. Using this information, the Video Streaming Service serves a video towards a mobile user with quality matching her momentary radio link quality and channel capacity. We present a novel algorithm, which improves the state of the art Dynamic Adaptive Streaming over HTTP (DASH) by providing a better quality of video streaming experienced by the mobile video consumer. A real experiment performed in a Long Term Evolution (LTE) femto-cell scenario proves that our new algorithm indeed improves the video quality in terms of video representations delivered, buffers established, and adaptation frequency experienced for different adaptation algorithms in DASH.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

According to Cisco [1], global IP traffic will account for 82% of all consumer traffic by 2021. Cisco also predicts that the Video-on-Demand (VoD) traffic will nearly double by 2021. Moreover, the traffic from wireless and mobile devices will account for more than 63% of total IP traffic in the Internet then. This means that the infrastructure of Mobile Network Operators (MNOs) has to support video delivery. Furthermore, the video delivery rates have to adapt to the network capacity (e.g., congestion) and the state of air interfaces (e.g., channel quality, interference, etc.) to provide the best Quality of Experience (QoE) to the consumer.

1.1 Overview

There is an on-going European Telecommunications Standards Institute (ETSI) Multi-Access Edge Computing (MEC) [2] (c.f., Chapter 2) initiative that can change the ecosystem of future mobile networks also in video delivery. Please notice that in September 2016, ETSI Mobile Edge Computing group changed its name from Mobile Edge Computing to Multi-Access Edge Computing. MEC extends intelligence at the network edge through computing and storage facilities deployed in the close vicinity of the Radio Access Network (c.f., Chapter 2). The MEC architecture (c.f., Chapter 3) is characterized by ultra-low latency high bandwidth as well as real-time access to radio network information. It provides a platform for Mobile Edge (ME) services (c.f., Chapter 3), which are often time critical applications having large data rates or high computation demands.

This thesis studies the video delivery in future mobile networks equipped with Multi-Access Edge Computing (MEC). We provide a MEC-based recommendation system for video delivery in such networks. In our work, the video service personalizes and limits the available video qualities using the momentary channel conditions experienced by the Users Equipment (UE). To this end, we observe the channel of every user by using a MEC Radio Network Information Service (RNIS) (c.f., Chapters 3) and compute the per-user radio channel capacity using novel Fourier-based traffic analysis (c.f., Chapter 4). We, then, dynamically present different video qualities towards the video consumer matching the users link capacity and radio connectivity (c.f., Chapters 5 and 6). The source code for the video service as well as the modified components and configurations of the mobile network are provided in the Appendices (c.f., Appendices A and B).

1.2 Contributions

The goal of this thesis is to provide, evaluate, and implement an adaptation algorithm for a video service that runs as an application on the ME. The video service assesses the momentary quality of the radio signal, reported by the UE, as well as the capacity of the wireless link between the UE and the base station. It provides the user with an adapted set of video qualities. The video service should work in a coexisting way with regular Adaptive Bitrate Streaming (c.f., Chapter 2) and adaptation mechanisms at the client. We compare it to a regular video delivery schemes that use momentary buffer fill level or experienced throughput at the client as a mechanism to define the appropriate video quality (c.f., Chapter. 2). We demonstrate that our video service results in a faster and better adaptation of the optimal video quality and therefore provide a better video quality to the user. We also improve the subjective QoE to the user by limiting the disruptions through providing the lower video quality, when a user is moving away, which should allow for a better anticipation of worse radio signal conditions and, thus, prevent the video stream from low buffer levels and possible stalls in spite of user mobility.

1.3 Thesis Structure

This work has the following organization. First, we present an overview of current mobile networks, describe MEC, and explain an idea behind the Adaptive Bitrate Streaming (ABR) video delivery using Dynamic Adaptive Streaming over Hypertext Transfer Protocol (HTTP) in Chapter 2. Second, we discuss the high level architecture and general concept of video delivery with MEC, and present an actual setup of the proposed architecture (c.f., Chapter 3). Third, we present a study of DASH traffic profiles and develop a novel MEC-assisted adaptation algorithm for controlling the available video qualities in DASH video streaming using Fourier-based traffic analysis (c.f., Chapter 4). Fourth, we design an experiment on MEC-assisted video delivery (c.f., Chapter 5). Fifth, we evaluate the algorithm developed in a real-world femto-cell scenario (c.f., Chapter 6). Finally, we conclude and suggest ideas for further research in Chapter 7.

Chapter 2

State of the Art

In the following sections, we present the architecture of cellular communication systems and the state-of-the-art of video delivery in mobile networks. First, the Long Term Evolution Network (LTE) is described in Sec. 2.1. Second, we elaborate on Multi-Access Edge Computing (MEC) in Sec. 2.2. Finally, we present Dynamic Adaptive Streaming over HTTP (DASH) and an overview of the current research in Sec. 2.3.

2.1 Mobile Telecommunication and Long Term Evolution (LTE)

This section explains the building elements of the LTE mobile network. The LTE/4G network consists of two main building blocks:

- **The Radio Access Network (RAN)** provides the wireless radio connection between the mobile devices and a base station.
- **The Evolved Packet Core (EPC)** or also System Architecture Evolution (SAE) is responsible for connecting mobile devices to packet networks such as the Internet.

We distinguish the following components of the minimal LTE network:

In the LTE RAN:

- **User equipment (UE)**: UE is a communicating device (e.g., smart phone, tablets, laptop with mobile network interface); it is also referred to as the LTE mobile terminal. Each UE carries a Universal Subscriber

Identity Module (USIM), which is a separate entity placed on a removable smart card. The USIM is used for the identification, authentication, and provides security keys, which protect radio transmissions over RAN.

- **Evolved Node B (eNB)**: eNB sends and receives the radio signal and connects the mobile terminal with the Core Network (CN). It allows for various signaling functions, such as the Radio Network Control (RNC) or Radio Resource Control (RRC), and is also responsible for the reliable delivery of data packets through the air towards several mobile terminals at the same time.

In the LTE EPC:

- **Home Subscription Server (HSS)**: HSS is a database server, which is maintained centrally at the MNO that contains the information about all mobile users subscribed to the network. It stores subscriber profiles, e.g., access to services, roaming access in other networks, etc.
- **Mobility Management Entity (MME)**: MME is a central control-plane node responsible for signaling, i.e., authentication, authorization, and mobility management.
- **Serving Gateway (S-GW)**: The S-GW is responsible for user-plane management. It transports user traffic between mobile terminals towards external networks; it connects the RAN and P-GW. S-GW is also a mobility anchor for user terminals.
- **Packet Data Network Gateway (P-GW)**: The PDN-GW or short P-GW is the edge router between the CN and external data networks (not operated by the MNO). A P-GW allocates IP addresses to UEs as well as performs traffic filtering and routing. The P-GW/S-GW use the GTP-based S1-U data-plane protocol that matches the IP data flows of users with GTP tunnels to deliver packets towards mobile terminals.

In this work, we refer to the S-GW and P-GW as to one entity and call it Serving/Packet Gateway (S/PGW). Fig. 2.1 depicts the LTE architecture and shows the relation between the involved and discussed components.

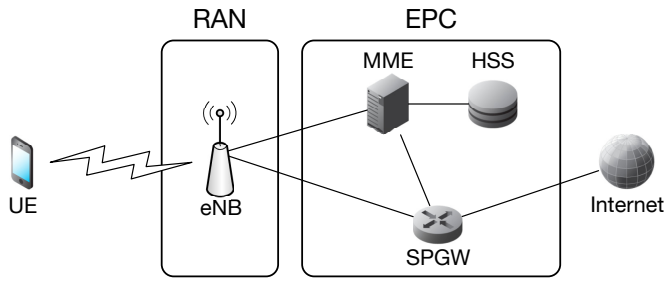


FIGURE 2.1: A simplified LTE system.

For a more in-depth discussion of the LTE functionality, please refer to [3] and [4]. In the following section, we describe the architecture of ETSI Multi-Access Edge Computing (MEC) allowing for video optimization that takes into account the state of user air interfaces.

2.2 Multi-Access Edge Computing (MEC)

The European Telecommunications Standards Institute (ETSI) defines a Multi-Access Edge Computing (MEC) network ecosystem, which enriches the network edge (i.e., the close vicinity of RAN) with cloud-computing capabilities [5, 2]. It offers application developers and content providers an environment of ultra-low latency and high bandwidth as well as real-time access to radio network information. Therefore, MEC dramatically improves the Quality of Service (QoS) and Quality of Experience (QoE) received by mobile subscribers. ETSI foresees various use-cases such as video analytics, location services, Internet-of-Things, data caching, and augmented reality. ETSI mostly focuses on MEC video transcoding or content optimization for end-users [5], which requires a large number of computing resources at the MEC platform processing the video. MEC platforms of the future (i.e., private clouds of a small size) will possess sparser resources comparing to regular cloud operators, so heavy operations on such infrastructures shall not be executed.

The vision of ETSI MEC [5, 2] employs the concepts of Network Function Virtualization (NFV) and Software Defined Networking (SDN) [6, 7]. There is a general consensus that MEC applications will be provisioned as Virtual Network Functions (VNFs) owned by third parties (i.e., not necessarily the

MNO) or vertical providers (e.g., caching, video delivery, augmented reality) [5]. The traffic will be managed by the unified control plane leveraging SDN [8, 9]. As an example, Schiller et al. [6] present an open-source NFV/SDN-based MEC platform providing application and traffic management in 4G/5G systems. The authors prove an appropriate operation of their platform through two use-cases (i.e., content caching and public safety). A UE communicates with VNF-based Mobile Edge Applications instantiated on the MEC cloud. The SDN-based control plane provides the mobile traffic directly from UEs towards MEC applications avoiding the CN. MEC applications can be aware of the state of the air interface (e.g, capacity, congestion, radio signal quality, etc.) through the RNIS implemented on top of MEC [5, 7]. This allows for many optimization problems taking into account RAN awareness. Several works concentrate on the scope of RAN management and programmability, which is being used to alter the system state according to arbitrary objective functions [10, 11].

In the following section, we describe Dynamic Adaptive Streaming over HTTP (DASH).

2.3 Dynamic Adaptive Streaming over HTTP (DASH)

Dynamic Adaptive Streaming over HTTP (DASH), also known as 3GPP-DASH or MPEG-DASH, is a popular standard for video streaming over the Internet allowing for improved user experience in the presence of variable network conditions, due to Adaptive Bitrate Streaming (ABR) of the video content. It has become the standard of video streaming and most big video platforms and services, such as YouTube and Netflix use it [12].

2.3.1 DASH Advanced Video Coding (AVC)

DASH uses partitioned multimedia files, which are delivered over HTTP. Besides the conventional Hyper Text Transfer Protocol (HTTP), DASH consists of two main components, which are the Media Presentation Description (MPD) file and video segments residing on a HTTP server, i.e., a video server.

- **Media Presentation Description (MPD)**: MPD is a XML manifest describing various characteristics of the video content such as the availability of the stream, different representations, encoding scheme, segment duration, and resource identifiers.
- **Segments**: The entire video content is segmented into segments of a predefined duration (e.g., 2 seconds, 5 seconds, 10 seconds). Each of these segments contains efficiently coded media data, according to or aligned with common media formats, such as MP4. The segments can be encoded with different bit rates and, therefore, provide multiple representations of the same media file. As the segments are all of the same duration, the client can change the representation at any given time in the video stream by downloading the next consecutive segment of any quality, i.e., higher and lower qualities.

The regular procedure in DASH is organized as follows. In a first step the client requests the MPD file from the video server. The MPD file contains information about the stream; the available representations, their average bit rate and resource identifiers of the segments in given representations. Then, the client requests each segment individually through the unique resource identifier using a simple HTTP GET request (c.f., Fig. 2.2). At the beginning, the player may be downloading a few segments of lower quality consecutively (c.f., Fig. 2.2 segments in light blue) in order to fill the buffer. The video player then starts displaying the video and the remaining segments are continuously fetched from the video server. The client also adapts the video quality (c.f., Fig. 2.2 each color is representing a distinct video quality), while streaming the video depending on different metrics, such as the buffer level, the experienced throughput while downloading the last segment, or CPU usage. This is completely up to the implementation of the video player at the client.

DASH provides an enhanced Quality of Experience (QoE), as it dynamically adapts the video quality using the adaptation algorithm on the client side; it also maintains an appropriate buffer fill level to avoid a stall of the video stream. Operating with a low buffer fill level, when only a few upcoming segments are cached on the client, leads to a high risk of not receiving the required portion of the video stream on time in the case of changing network conditions. When the buffer drops down to 0, the client is forced to stop displaying the video, rebuffer, i.e., download a few segments that refill the buffer, and continue with displaying the video after a short break.

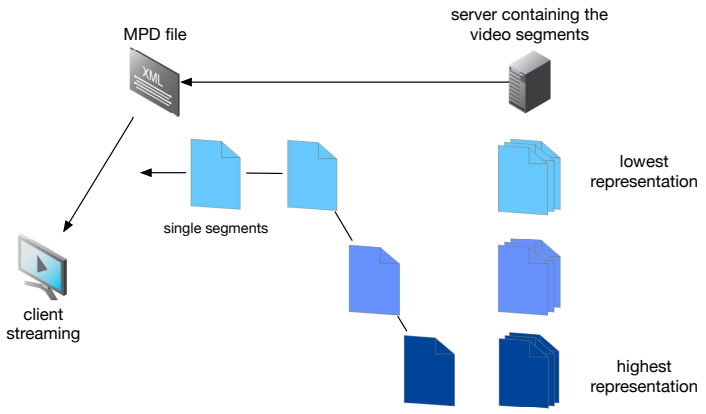


FIGURE 2.2: A simplified illustration of the DASH procedure.

Notice that a non-continuous video stream provided to the user negatively impacts the QoE. Naturally, there is a trade-off between the buffer level and the representation provided, i.e., a lower representation will generally cause a higher buffer level and, thus, continuous and smooth video stream. The research methods on video adaptation suggest us to use the maximal video quality guaranteeing a buffer fill level that minimizes the chance of re-buffering so that the client can continuously stream a video of high quality (c.f., Sec. 2.3.4).

For a more technical description of DASH AVC, please consider the full technical specification [13] of the 3GPP.

2.3.2 DASH Scalable Video Coding (SVC)

It is worth noting that there exist different kinds of video encodings and this work focuses on video delivery with DASH Advanced Video Coding (AVC). Many projects, especially the ones with improved caching strategies, work with DASH Scalable Video Coding (SVC) [14]. The key difference between these two DASH types is that in DASH SVC, the representations are encoded in a layered and dependent way. In order to display a segment of quality $n+1$, we need to possess the same frame of quality n (meaning all qualities lower than $n+1$). The client then needs to decode the video frame from multiple files one by one, which leads to higher workload. The main advantage is that if the frame of quality $n+1$ is not delivered on time at the client, the player can still play the video of quality n. However, due to the decoding of multiple files, this could results in a higher resource utilization at the

UE. Thus, DASH SVC can be considered less mobile friendly, while sparse resources are available on mobile devices.

2.3.3 DASH Server and Network Assisted (SAND)

In DASH AVC as well as DASH SVC, service providers have a little control over the client behavior. It is, therefore, difficult to offer a consistent quality of service. Furthermore, if there is a network failure or the network is reconfigured, outdated resources in MPD files result in misdirected and unsuccessful segment requests. In order to counteract such drawbacks of client-driven DASH, the Moving Picture Experts Group (MPEG) proposed an extension to DASH called *Server and Network Assisted DASH* (SAND) [15].

The DASH SAND architecture [16] provides a set of well-defined SAND messages that allow for the exchange of client and server-side information through the network. The standard provides four categories of network elements. The network is separated into clients, servers, DASH aware network elements (DANEs), and regular network elements (RNE). In the DASH SAND architecture, DANEs and DASH clients can exchange message between each other. Furthermore, DANEs themselves can exchange information about the current status. The messages can be of different types, such as metrics, status, parameter enhancing delivery or parameter enhancing reception, and are well defined in the DASH SAND specification [16]. The DANEs can, thus, share information about the QoS they provide, e.g., cached segments, guaranteed bitrate, maximum bitrate. Similarly the DASH client shares information about its requirements for the best QoE, e.g., deadlines and accepted alternatives for the next DASH segment or maximum round trip time (RTT). The new architecture of DASH SAND should compensate for the limitations of the client-controlled DASH. Furthermore, it provides a concept with a context aware network that allows for better QoS and QoE of DASH.

However, as the DASH SAND architecture is still in the early stages of the development and fairly used, we decided to work with DASH AVC, which has proven to be one of the most popular ABR streaming techniques in the Internet.

2.3.4 Video Adaptation Mechanisms

In our work, we only use DASH AVC, which is described in Section 2.3.1. DASH AVC uses a simple operation model, which requires a complex adaptation mechanism at the client. Thus, let us now survey the video adaptation algorithms implemented in the video player. In the following sections, we discuss the two most important video adaptation algorithms, which are also implemented in the GPAC Project on Advanced Content (GPAC; a recursive acronym). MP4Client[1]. The GPAC MP4Client is of great importance for this work, while we use it in our experiments (c.f., Sec 3.9) as a video player.

There are two main categories of video adaptation algorithms, which are buffer-based and throughput-based. A comparison between the most relevant algorithms can be found in the work of Karagkioules et al. [17]. According to their comparative case, the buffer-based adaptations tend to outperform other adaptation algorithms in mobile networks. However, they state that a robust adaption mechanism for high QoE under variable network conditions, as experienced in mobile networks does not exist yet.

Throughput-Based Video Adaptation

In throughput-based adaptation, the video quality is determined using the experienced throughput downloading the previous segment and the advertised bit rates of the different representations in the MPD file. At the beginning, the lowest representation quality is used.

Moreover, two different switching strategies can then be distinguished, i.e., aggressive and passive switching. In passive switching, the adaptation occurs stepwise; the client switches from the current representation to another one, which has only a slightly higher bit rate (i.e., a representation that the guarantees the lowest bit rate increase is selected in the following step). In contrast, aggressive switching almost immediately adapts to the best possible representation having the closest bit rate to the momentarily experienced channel capacity.

GPAC MP4Client implements throughput-based adaptation, buffer-based adaptation, and other switching algorithms (c.f., dash_client.c subsystem

[1]The MP4Client is developed as part of the GPAC multimedia open source project: https://gpac.wp.imt.fr/

in the GPAC MP4Client source code). Algorithm 1 presents a simplified description of a throughput-based adaptation implemented in GPAC MP4Client. The throughput-based adaptation, `dash_do_rate_adaptation_legacy_rate`, switches the video quality based on the previously experienced throughput (i.e., the throughout experienced during the download phase of the last segment) and the advertised bit rates in the MPD file. GPAC distinguishes between aggressive and passive switching strategies. The client decides whether a higher video quality should be requested. On the one hand, if the client decides to increase the video quality (i.e., the experienced download rate is higher than the required bit rate of a currently used representation), the client selects the highest or next higher representation that the network can handle depending on the aggressive or passive switching strategy respectively. On the other hand, if the bandwidth is smaller than the bit rate required by the current representation, the client immediately selects the highest video representation having a smaller or equal bitrate to the experienced throughput.

Buffer-Based Video Adaptation

Buffer-based video adaptation adds another decision making layer, as it uses the buffer fill level as the indication of the video quality. The client keeps track of the minimum and maximum buffer fill levels experienced during the reception of the current video stream. The buffer-based adaptation uses the observed buffer fill level as an indication on how to adapt the quality of the video stream. On the one hand, if the algorithm experiences minimum buffer levels, it goes down with the video quality and switches to lower representations. On the other hand, it waits with going up with the video quality until the buffer level surpasses a threshold (e.g., a fraction of the maximum observed buffer level). Optimally, the buffer level shall not decrease during the downstream of the segment, i.e., the refill process should be always faster than the use of the buffer. A simplified version of the buffer-based adaptation algorithm implemented in the GPAC MP4Client is described in Algorithm 2.

The buffer-based adaptation algorithm basically distinguishes three different cases (c.f., Algorithm 2). First the buffer can be too low, and we need to pro-actively go down with the video quality, as we risk running out of video

Algorithm 1 Throughput-based adaptation implemented in MP4Client

```
1: procedure DO RATE ADAPTATION RATE BASED(dl_rate, go_up, aggressive_switching)
2:     ▷ The experienced download rate in bits per sec, a boolean variable go_up; default true and aggressive_switching; default true
3:
4:     R = set of all available representations advertised in the MPD file
5:     current_rep ← the current representation
6:     new_rep ← null
7:
8:         ▷ If the current representation requires a higher bandwidth than available
9:     go_up = dl_rate < current_rep.bitrate ? false : true
10:
11:    if go_up then
12:        if aggressive_switching then
13:            for each rep ∈ R do
14:                if rep.bitrate >= new_rep.bitrate ∧ rep.bitrate <= dl_rate then
15:                    new_rep = rep
16:        else
17:            for each rep ∈ R do
18:                if rep.bitrate >= current_rep.bitrate ∧ rep.bitrate <= dl_rate ∧ rep.bitrate <= new_rep.bitrate then
19:                    new_representation = rep
20:        if new_rep = null then
21:            new_rep ← lowest available representation
22:    else
23:        for each rep ∈ R do
24:            if rep.bitrate <= dl_rate ∧ rep.bitrate >= new_rep.bitrate then
25:                new_rep = rep
       return new_rep
```

Algorithm 2 Buffer-based adaptation as implemented in the MP4Client

```
1: procedure DO RATE ADAPTATION BUFFER BASED(dl_rate, go_up, aggressive_switching)
2:     ▷ The experienced download rate in bits per sec, a boolean variable go_up; default false and aggressive_switching; default false
3:
4:     go_up = dl_rate > current_rep.bitrate ? true : false
5:     buf_high_threshold ← 2/3 of the max. observed buffer level
6:     buf_low_threshold ← the min. observed buffer level
7:     buf_curr ← the current buffer level
8:     occ ← difference between the buffer level after and before the
           download of the last segment
9:     if buf_curr < buf_low_threshold then
10:        DO RATE ADAPATATION RATE BASED(dl_rate-10, false,
               aggressive_switching)
11:    else if buf_curr > buf_high_threshold ∧ occ > 0 then
12:        DO RATE ADAPTATION RATE BASED(dl_rate, true,
               aggressive_switching)
13:    else
14:        keep the current representation
```

segments and, thus, stalling the video. Second, the buffer can be above a certain level and we managed to fill the buffer during the download of the last segment, so we can securely go up with the video quality without risking a stall. The appropriate representation is then selected, by using the previously discussed throughput-based adaptation (c.f., Algorithm 1) with the provided parameters. Third, the buffer level is somewhere in between the two thresholds, which means that we should keep the current representation.

Due to the introduction of the buffer level in the decision making process, buffer-based algorithms tend to have fewer changes in the video qualities and provide a better QoE to the user.

In the following subsection, we survey the state of the art of DASH video delivery in mobile networks equipped with SDN, NFV, and MEC.

2.3.5 DASH improved by SDN and MEC

Han et al. [18] designed an MP-DASH algorithm, which explores the concept of multi-homing. The authors use Multipath TCP (MP-TCP) to improve the definition of video streaming experienced on a User Equipment (UE) having both WiFi and mobile network interfaces (4G/5G). A deadline scheduler

is developed taking into account interface preference and delay tolerant nature of the video traffic. MP-DASH reduces the use of the mobile network interface by 99% and the UE battery consumption by up to 85% with little video quality degradation when compared to the off-the-shelf implementation of the Multipath TCP (MP-TCP). Xu et al. [19], developed a novel Quality Aware adaptive Concurrent Multipath Transfer solution (CMT-QA), which explores multi-homing SCTP transmissions in heterogeneous networks. The aim is to improve the quality of video streaming and best-effort data transmissions through an appropriate path quality assessment, an algorithm distributing data over paths, and a novel acknowledgment mechanism.

There are several different approaches improving the video delivery in networks studied through simulations [20, 21, 22, 23]. Cetinkaya et al. [20] uses SVC-DASH and Software Defined Networking (SDN) to improve video streaming. The authors suggest routing video flows through the underlying infrastructure taking into consideration the capacity of the backhaul network and a bit rate of particular flows registered in the system. The controller allocating flows to the infrastructure does not exchange information with the video server, but instead analyzes the flow information reported by the SDN switches through OpenFlow [24].
Li et al. [21] propose a Mobile Edge Computing (MEC) approach to improve fairness and overall video definition among UEs sharing the same channel. An Integer Linear Program (ILP) has been developed to select appropriate video representation (data rates) for video consumers. They use a dynamically generated MPD file at the MEC server to adapt the bitrate to momentary network conditions. When a MEC infrastructure discovers potential congestion, the MPD file is updated and higher bit rates are removed. The clients, therefore, move to lower bit-rates overcoming congestion.
Lai et al. [22] propose a method for improved video delivery in heterogeneous networks with SDN. They use two modes of operation called buffering and streaming modes. In the buffering mode, the client is able to buffer video representations of good quality. The authors propose two methods for scaling video representations up and down. When the buffering mode is unable to refill the buffer that drops to 0, the buffering mode is abandoned, and the streaming mode is used. In the streaming mode, the authors use a polynomial bandwidth estimation to select the most appropriate representation.
Fajardo et al. [23] propose a network assisted HTTP streaming mechanism based on MEC. Their mechanism is able to adapt DASH streams to different channel conditions of users based on periodical measurements of Channel

Quality Indicators (CQIs) and adaptation algorithms matching experienced CQIs to video definitions. The main parts of their system are: the DASH manager adapting content towards end-users, the DASH cache to handle previously retrieved segments, and the RAN monitoring module, which is fed by the channel state information (CSI) established by means of CQI reported by UE at the eNB within the LTE system.
Foukas et al. [11] prove a similar approach in a real-experiment by using FlexRAN. In their use-case, CQIs statistics reported by a UE are gathered at the FlexRAN controller. A DASH-based video streaming server uses information gathered at the controller to match the CQIs to video representations (i.e., bitrates). It then provides the UE with an appropriate video definition matching the momentary channel conditions.

Novelties in Comparison to the State of the Art

Although a lot of research has already been performed in the field of DASH video delivery, our work significantly differs from existing solutions. In contrast to Fajardo et al. [23], who introduces a DASH manager operating at the level of the eNB scheduler, we do not interact with the radio scheduler and focus only on passive RAN observations (i.e., the information about throughput and radio quality metrics received from RNIS). We, therefore, only introduce a video service that takes into account the status of the radio link belonging to a given UE. Also, in contrast to Foukas et al. [11], we do not use Channel Quality Indicators (CQIs, c.f., Sec. 4.2), which depend on hardware, vendor, and environmental specifics, but instead, we use experienced throughout and received signal power as the radio quality measure. Furthermore, we perform our experiments using a real radio equipment, i.e., we do evaluate our findings in simulations, while [25] uses an emulation of the network (i.e., Mininet[2]).

[2]http://mininet.org/

Chapter 3

Architecture and Implementation of the Adaptive Video Delivery

This chapter presents an overview of the architecture and the implementation of the MEC platform and the video delivery, including the video server and client. First, we provide the high level architecture of the MEC platform in Sec. 3.1 and then discuss the interaction and information flow between these components in Sec. 3.2. Second, we introduce the implementation of our MEC platform (c.f., Sec. 3.4) and provide details about the modifications of necessary third party components (c.f., Sec 3.6). Third, we describe and explain the provisioning of the video service on the MEC platform (c.f., Sec. 3.8) and the use of the video client (c.f., Sec. 3.9).

3.1 High Level Architecture

The high level architecture of our network follows the ETSI MEC [2] model. It consists of a base station providing radio access network (RAN), a MEC cloud, and a core network (CN). As in the regular LTE, the RAN provides a radio signal towards the UEs and connects them to the CN of the MNO. The UEs benefit from the connection to the Internet through the CN (c.f., Fig.3.1).

A Software Defined Radio Access Network (SD-RAN) platform provides a global view of the radio network status and manages radio resources among eNBs [10]. The SD-RAN platform is, therefore, acting as an RNIS [5], a service providing authorized applications with low-level, real-time, radio network information, which can be used by MEC applications (Apps) to calculate derived secondary information (e.g., location of the UE can be derived from the

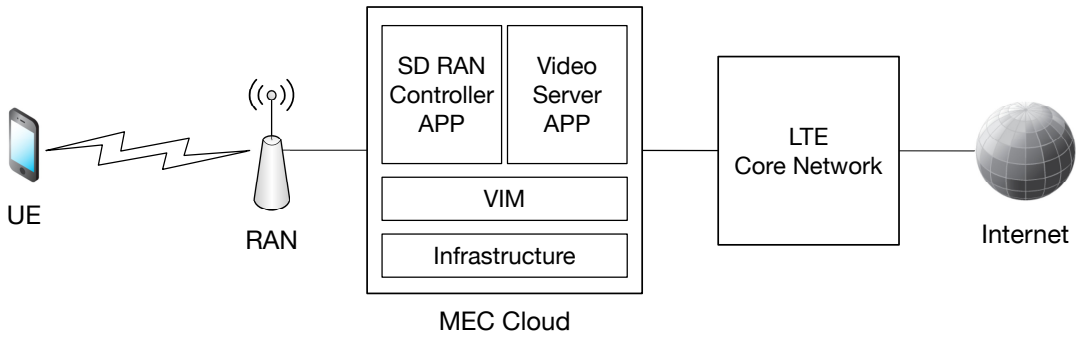


FIGURE 3.1: A depiction of the system architecture.

power information through a trilateration process [26]). Furthermore, as the MEC cloud is installed very close to the RAN, the traffic can avoid the CN and directly access MEC Apps. Therefore, the MEC Apps benefit from the close vicinity of the eNB and, thus, the UE experiences low latency, which is crucial for low-delay tolerant applications. MEC Apps instantiated on the MEC cloud receive traffic directly from the user plane (e.g., video service) or other MEC related services (e.g., SD-RAN platform), dealing with radio control and management planes. Moreover, MEC Apps can exchange information between one another, or create advanced application chains. MEC Apps are typically hosted as VNFs on top of abstracted virtual infrastructures controlled by Virtual Infrastructure Managers (VIMs).

3.2 Information Flow

The streaming process and information flow between the system elements is illustrated in Fig. 3.2. Initially, the UE starts the information cycle by initiating the video streaming form the video server residing at the ME cloud. The user is given an initial MPD file, which contains all available representations. The video player requests the segments of a given quality, based on its local adaptation algorithms. The requested segments are sent from the video server over the network and the established radio link between the eNB and the UE to the user. The RNIS registers the traffic occurrence as well as the momentary network status (e.g., signal quality at the UE) and provides this information to the video service, in which the statistics are processed and used for the video-stream adaptation (i.e., to restrict or expand video qualities) of the UE's personal MPD file. Finally, the UE refreshes its personalized

MPD file and continues streaming using the qualities provided in the dynamically generated MPD file. This information flow, between the components involved in the video streaming, can be understood as a continuous cycle, which is maintained as long as the video is playing or the user terminates the stream.

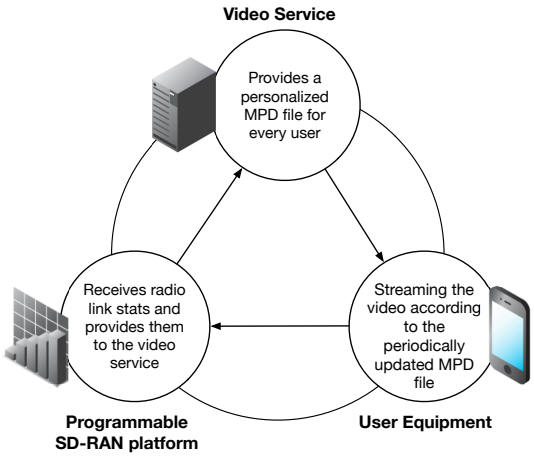


FIGURE 3.2: Illustration of information flow between the main components in our video service architecture.

3.3 Statistics Distribution from the SD-RAN Controller

As the communication between the base station and the video service (ME App) is of great importance, we present a general approach for accessing statistics obtained from the SD-RAN controller. This work uses a simple nested request-response model, c.f., Fig. 3.3, in which an ME App accesses the statistics through the northbound application interface (API) of the controller. Typically, the northbound API of the SD-RAN controller is used for communication purposes between the controller and higher-level modules, i.e., services or applications. To receive radio information, the SD-RAN controller requests the statistics from the eNB through the southbound API used to communicate with the network components (e.g., eNB). It then processes the results and finally responds to the requests of the ME App trough the northbound interface.

This scheme could be easily extended to the publish/subscribe model, in which the ME App is periodically provided with the RAN information about

UEs momentarily connected to the MEC video service. However, we mainly concentrate on video delivery, and do not develop elastic Pub/Sub systems for flexible information dissemination. Therefore, the SD-RAN controller is directly used in our design.

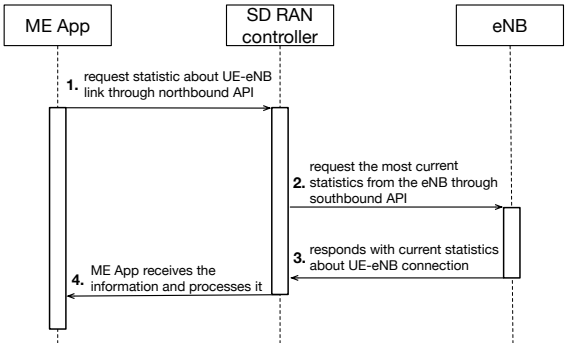


FIGURE 3.3: Request-response model between the ME App and the SD-RAN platform.

3.4 Platform Implementation

The platform developed in this work (c.f., Fig. 3.4) is based on previous work of the CDS group [6]. We work with a fully cloudified core network, i.e., all the components of the mobile network, except the eNB, are fully virtualized. The mobile network follows the current 4G mobile telecommunication standards and uses the OpenAirInterface (OAI) [27, 28], which implements HSS, MME, and S/PGW of the minimal LTE CN and the eNB for RAN. The configuration files of these components are included in Appendix A.2. OAI provides the LTE mobile network and the OpenAirInterface Base Band Unit (BBU), which processes and provides the radio signal through radio equipment (i.e., the USRP B210 board[1]). The setup provides an LTE Frequency Division Duplex (FDD) transmission in band 7 (2.5 GHz/2.6 GHz) using 5 MHz channels and the Single-Input Single-Output (SISO) mode. To receive the LTE signal, we use a smartphone Moto 2[2], which connects to the mobile network.

As SD-RAN, we use FlexRAN [11], which extends OAI with a flexible SD-RAN solution. FlexRAN also provides an interface to get information about

[1]https://www.ettus.com/product/details/UB210-KIT
[2]https://www.motorola.com/us/products/moto-z-play-gen-2

the RAN and, thus, implements an RNIS-like service, which allows us to obtain information about the current network status (e.g., radio signal quality or channel usage). FlexRAN consists of an agent co-located with the OAI BBU, which is implemented in the OAI `feature-68-enb-agent`[3] branch and the external SD-RAN controller `develop-uplink`[4] branch. In our setup, the FlexRAN controller module as well as the video service are instantiated on the OpenStack-based MEC cloud and are running on Ubuntu virtual machines (c.f., Fig. 3.4). The traffic at the network edge is managed by the Open vSwitch (OVS) [29], which is part of OpenStack and forwards traffic towards the CN or MEC Apps according to a flow table [6]. The OVS switch handles IP traffic between the UE, MEC Apps, and the Internet accordingly.

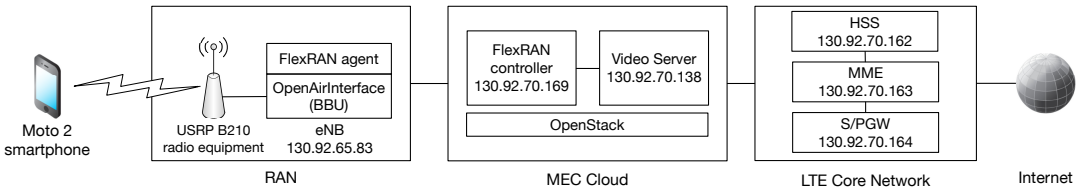


FIGURE 3.4: The MEC platform setup, consisting of the RAN, MEC cloud and LTE Core Network.

3.5 Provisioning of the LTE Network

In order to provision the LTE network, we need to start the entire OAI EPC as well as the FlexRAN controller module and the eNB respectively. As the EPC is fully cloudified, the installation process is fully managed and orchestrated by Juju[5,6], a tool that manages software in the cloud environment. To start the EPC, we launch each module individually (i.e., HSS, MME, and S/PGW), which requires us to execute the following commands on OAI HSS, OAI MME, and OAI S/PGW respectively:

- HSS: `/srv/openair-cn/scripts/run_hss`
- MME: `/srv/openair-cn/scripts/run_mme`
- S/PGW: `/srv/openair-cn/scripts/run_spgw`

[3]https://gitlab.eurecom.fr/oai/openairinterface5g/tree/feature-68-enb-agent
[4]https://gitlab.eurecom.fr/flexran/flexran-rtc/tree/develop-uplink
[5]https://jujucharms.com/
[6]https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/OAIonJuJu

The configuration of each component of the EPC (c.f., Appendix A.2) and the changes the S/PGW (c.f., Appendix B.1) are provided in the appendix. Before starting the eNB, we need to start the FlexRAN controller. The reason for this is that the eNB contains the FlexRAN agent that connects and registers with the controller. The instructions to build and run the FlexRAN controller and the eNB, i.e. FlexRAN agent, are provided in the appendix (c.f., Appendix A.1).

When the CN and RAN components are up and running, we can use any LTE band 7-compatible UE. Having a USIM registered in the HSS database, we attach and connect the UE to the cloudified mobile network. Furthermore, we can request statistics about the eNB and its connected UEs from FlexRAN using the RESTful API at `http://130.92.70.169:9999/stats_manager/all`. A request towards the API is handled by the FlexRAN agent at the eNB, (c.f., `openair2/ENB_APP/flexran_agent_handler.c`) that responds with appropriate information. A detailed description of communication between the OAI-eNB and FlexRAN is provided in subsequent Sec. 3.7.

3.6 Modifications to the Components of the Mobile Network

In order to derive the statistics about the network status, such as radio signal quality at the UE or status of uplink and downlink shared channels, from the SD-RAN platform (c.f., Sec. 3.4), we need to understand the communication between the SD-RAN controller (i.e., FlexRAN controller) and the SD-RAN agent (i.e., FlexRAN agent). We, therefore, discuss the implementation of the communication between the eNB and, furthermore, explain the modifications of the source files (c.f., Appendix B.2, B.3, B.4) to provide statistics according to our needs.

3.7 Communication between FlexRAN agent and server

The communication between FlexRAN server (i.e., SD-RAN controller) and agent (i.e., OAI-eNB) is based upon protocol buffers[7], which allow for an efficient mechanism exchanging serialized data structures. We can, thus, extend the existing communication between the FlexRAN server and agent to include information about the RAN. To this end, we define a new protocol buffer (protobuf) message, describing data structures that will be serialized and exchanged in the network. To this end, we create a new message (c.f., Listing 3.1) in `stats_common.proto`, which needs to be defined at both ends (i.e., the FlexRAN controller and agent). The new message is part of the UE status report (c.f., `stats_messages.proto`). We are free to include as many values and nested messages in our message as we want. In order to request the newly created message from the eNB, we need to set an appropriate flag at the FlexRAN controller (c.f., `stats_manager.cc`) to signal that our message will be transported through the network.

LISTING 3.1: The implemented protobuf message in `stats_common.proto`.

```
1 message flex_dl_video_stats {
2         optional int32 rsrp_value = 1;
3         optional int32 rsrq_value = 2;
4         optional uint32 cqi_value = 3;
5         optional uint64 num_total_bytes = 4;
6         optional uint64 num_acks = 5;
7         optional uint64 num_nacks = 6;
8         optional uint32 mcs_value = 7;
9         optional uint64 timestamp = 8;
10 }
```

In the second step, we fill the corresponding fields of the message. This action takes place at the eNB and is implemented in the FlexRAN agent, (c.f., `openair2/ENB_APP/flexran_agent_mac.c`). Our message (c.f., Listing 3.1) provides statistics about the UE-eNB link status including Reference Signal Received Power (RSRP), Reference Signal Received Quality (RSRQ), Channel Quality Index (CQI), and Coding Modulation Scheme (MCS), which reflect

[7] https://developers.google.com/protocol-buffers/

the signal quality and the coding efficiency, but also the number of total bytes sent on the shared downlink channel as well as the number of acknowledged and unacknowledged data chunks reported by Hybrid Automatic Repeat reQuest (HARQ) on the link layer [30].

Interaction between ME App - FlexRAN - OAI-eNB

Fig. 3.5 illustrates how the ME App, the FlexRAN controller, and the OAI-eNB interact. It explains the basic functionality of a ME App requesting the available statistics about the connected UEs and the eNB. It follows the structure outlined in Sec. 3.3.

The ME App originates an HTTP request (1) towards the northbound API of the FlexRAN controller. FlexRAN controller provides an interface to interact with the FlexRAN agent and, therefore, controls the eNB. The Pistache[8] server provides the REST framework of the FlexRAN controller and handles the request in (2.1). The request is processed at the northbound API (c.f., `north_api/stats_manager_calls.cc`) and the appropriate statistics are requested from the FlexRAN agent by creating the corresponding protobuf message (c.f., `app/stats_manager.cc`). The message is sent over the southbound API (2.3) from the FlexRAN server to the FlexRAN agent (c.f., `core/request_manager.cc`) residing at the eNB. The message is then received and handled at the eNB (3) by a corresponding handler (c.f., `openair2/ENB_APP/flexran_agent_handler.c`), which fills out a response message using information about the state of the LTE Medium Access Control (MAC) layer of the UE and eNB. The information about MAC are gathered at the eNB through corresponding modules (c.f., `openair2/ENB_APP/-CONTROL_MODULES/MAC/flexran_agent_mac.c`). The message is then again sent through the network (4) back to FlexRAN. Finally, the returned result is provided to the ME App (5) as an HTTP response presenting the statistics in either plain text or JavaScript Object Notation (JSON).

In the last step, the ME Application can parse and process the statistics for further use, e.g. restricting the available video qualities for video streaming.

[8] http://pistache.io/

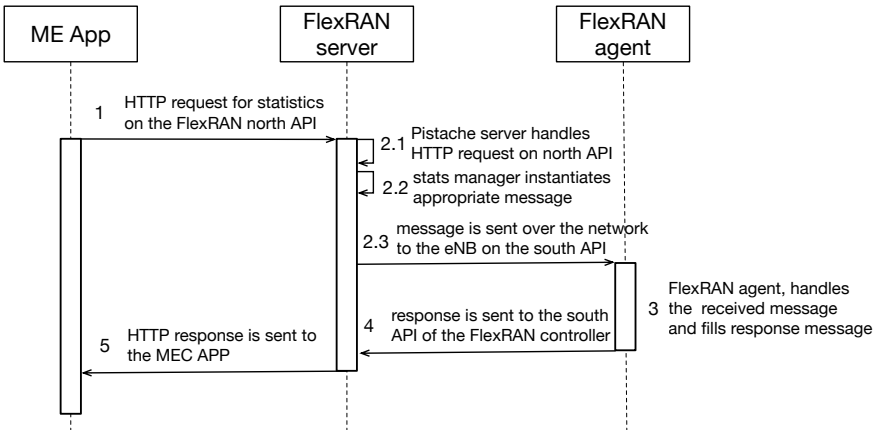


FIGURE 3.5: A sequence diagram describing the procedure of interactions for a UE statistics request.

3.8 Adaptive Video Service

In order to provide Adaptive Bitrate (ABR) video content to the user, we deploy a video server at the network edge within the ME cloud that supports DASH. The video server runs Ubuntu 16.04 LTS and a regular Apache/2.4 web server. We encode a video in different qualities to provide multiple representations according to the DASH standard.

3.8.1 Preparation of the Video

In order to find the appropriate qualities for the video encoding (i.e., bitrates), we assess the channel capacity between the UE and the video server using `iperf -c` on the UE and `iperf -s` on the video server. In our case, this process returns the maximum channel capacity of 8.33 Mbps.

```
remo@ubuntu:~$ iperf -c 130.92.70.138
------------------------------------------------------------
Client connecting to 130.92.70.138, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 172.16.0.2 port 55630 connected with
   130.92.70.138 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.1 sec  10.0 MBytes  8.33 Mbits/sec
```

We identify the bottleneck of the mobile network, which is the wireless channel between UE and eNB, since all other wired Gigabit Ethernet connections provide a much higher bandwidth.

Given the maximum available bandwidth between the UE and video service, we can now encode the video using different bitrates. To fully saturate the link, a video with a bitrate of around 8 Mbps is necessary. We choose a freely available UHD video from Goddard Media Studios (GMS) [31], which has the initial bitrate of 39.9 Mbps. Using the guide from Bitmovin [32], we transcode the video using the `h264`[9] codec (i.e., a popular video compression standard) with ten distinct bitrates (c.f., Table 3.1). For the `x264` transcoding, the command `x264 -output OUTPUT.264 -fps 60 -preset slow -bitrate BITRATE -vbv-maxrate 2*BITRATE -vbv-bufsize 4*BITRATE -min-keyint SEGMENT_DURATION -keyint SEGMENT_DURATION -scenecut 0 -no-scenecut -pass 1 -video-filter "resize:width=WIDTH,height=HEIGHT" INPUT.mp4` (variables in capital letters) is used. After that, the raw `h264` video data is encapsulated in a container, such as `mp4`[10].

TABLE 3.1: The different representations and bit rates of the encoded video.

ID	Resolution [px × px]	Specified max. bitrate [kbps]	min. bandwidth [kbps]
1	320×180	50	54.994
2	320×180	100	105.412
3	320×180	500	509.222
4	640×360	800	811.818
5	640×360	1’000	1’014.234
6	640×360	1’500	1’520.707
7	1280×720	2’000	2’012.579
8	1280×720	4’000	4’011.958
9	1280×720	6’000	6’016.341
10	1280×720	8’000	8’024.380

Finally, we prepare the video for adaptive streaming (i.e., the DASH protocol) with the MP4Box[11] multimedia packager to “dashify” the video content, i.e., segmentation of the video and creation of a DASH manifest file. The command used to generate DASH files of segment

[9] http://www.itu.int/rec/T-REC-H.264
[10] https://mpeg.chiariglione.org/standards/mpeg-4
[11] https://gpac.wp.imt.fr/

duration t_s ms from different video representations is: `MP4Box -dash` t_s `-frag` t_s `-rap -out DASHFILE.mpd -profile live VIDEO_QUALITY#1.mp4 VIDEO_QUALITY#2.mp4 ....`

3.8.2 Adaptation to Current Network Status

The DASH protocol is normally developed for dynamic adaptive streaming with adaptation control at the client side (c.f., Sec. 2.3). As the client is not aware of the momentary link-layer/physical-layer information, it can only adapt the video quality using measured data on the client, i.e., the experienced throughput and the buffer fill level. This information does not allow us to construct an optimal adaptation mechanism for video streaming in mobile networks.

Since the DASH standard allows for dynamical MPD files, the video server can change and update the MPD file when required. We, therefore, provide a `Python` script that parses the data received from RNIS and dynamically updates the MPD file according to the information from RNIS link-specific observations. We discuss these observations in Chapter 4 and present an MEC adaptation mechanism for DASH in Section 4.3. To run the video service and periodically check for radio network information and update the user's MPD file, the command `python main.py -f RNIS_URL -m MPD_input -s MPD_OUTPUT -i Interval` is used. Here, the option `-f` provides the URL of RNIS (the FlexRAN message are discussed in Sec. 3.7), `-m` and -s provide the location of the original and modified MPD file, and -i specifies is the time between the measurements/MPD file updates.

3.9 Video Client

As a video client, we use the open-source MP4Client[12] client, which runs on a laptop connected to a smartphone (i.e., WiFi LTE tethering). To periodically refresh the MPD file at the client, there are two possible options [25]. First, we can use the dynamic type of the MPD file on the server and specify the `minimumUpdatePeriod` [13]. This is an option used frequently in the case of live streams, in which the available representations may change and the

[12] https://gpac.wp.imt.fr/

client needs to be informed about the changes. Second, we can change the implementation to force the client to refresh the MPD file periodically without altering the MPD type, i.e., refresh static MPD files. We opted for the latter option (i.e., we do not use live streams, but regular Video On Demand (VOD) streams). We therefore modified the MP4Client (c.f., `dash_client.c`) to refresh an MPD file every 5 seconds (c.f., Appendix B.6). In both cases, the client is forced to periodically request a dynamically updated MPD file (with a specified update period). The client continues streaming a video according to representations advertised in the most recently received MPD file.

Chapter 4

Analysis of Video Streaming in Mobile Networks

In this Chapter, we study traffic patterns of DASH video delivery (c.f., Sec. 4.1) with variable radio signal quality caused by user mobility (c.f., Sec. 4.2). To gather the traffic statistics, we use a real deployment of the architecture discussed in Chapter 3. The aforementioned traffic statistics are the key to the development of a novel MEC-assisted context aware video delivery provided later on in Sec. 4.3.

4.1 Analysis of Traffic Profiles

In the following sections, we study traffic profiles (i.e., traffic patters) in mobile networks during the video delivery (c.f., Sec. 4.1.1) and elaborate on the notion, calculation, and evaluation of the margin (i.e., idle time) between the delivery of individual video segments in DASH-AVC (c.f., Secs. 4.1.2, 4.1.3 and 4.1.4).

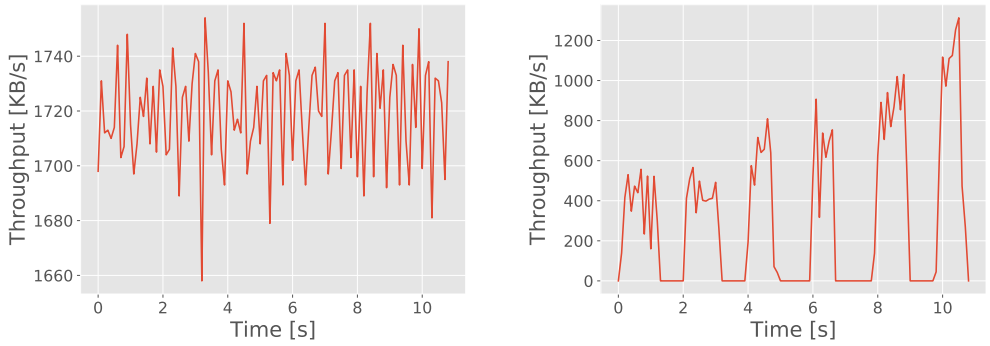
4.1.1 Observations in Traffic Patterns

We assume that the video streaming is the most dominant traffic for a user, i.e., other applications utilize only a small fraction of bandwidth on the mobile device at the same time. Moreover, there should be only one video stream per user. This may seem like a strong restriction, however, watching multiple video streams is rare, and mobile users typically use one application

at a time. Therefore, the background traffic from other applications, such as instant messengers, email, social media or games, can be neglected.

In our initial measurements, using the setup as discussed in Sec. 3.4, we observe a specific periodical pattern (c.f., Fig. 4.1b), which is typical for video delivery with DASH using buffer-based adaptation algorithms [33, 11, 18]. The traffic requested by the client appears approximately as a periodical rectangle function. Fig. 4.1b shows that the channel is periodically occupied and idle with a certain frequency. This is caused by a periodical re-fill of the video buffer at the client side. When the buffer level decreases below a certain threshold, a new segment is requested by the client. The periodicity of traffic peaks (i.e., rectangles) in the experienced goodput is approximately equal to the segment duration (e.g., 2 seconds) as the client consumes the video content in real time. For example, a 2 second segment is consumed within 2 seconds (c.f., Fig. 4.1b). Therefore after these two seconds, a new segment is again needed by the client.

Figs. 4.1a and 4.1b, obtained from the the statistics of the SD-RAN platform (c.f., Sec. 3.4), as well as related traffic plots reported by Augustin et al. [34], clearly show that the DASH video pattern, c.f., Fig. 4.1b, can be distinguished from best-effort downlink/uplink transmission, c.f., Fig 4.1a, which often causes an approximately constant traffic pattern limited by the channel capacity. Fig. 4.1a shows the download of a large file using `wget` over the mobile network.



(a) (b)

FIGURE 4.1: The throughput of best-effort (e.g., wget) (a) and video transmission (e.g., DASH) (b) can be clearly distinguished

We also notice that the throughput amplitude of peaks increases, c.f., Fig. 4.1b, when the client changes to a higher representation/bitrate of the

video. For these segments of higher video quality, the throughput reaches the channel capacity and the rectangle-like peaks start spanning longer time ranges, as it takes a longer amount of time, to download a segment of a larger size. Furthermore we notice that longer transmissions more efficiently use the channel due to the TCP adaptation to the channel capacity through slow start and congestion avoidance phases [35].

Our main idea is to use the distance (i.e., idle-time) between individual peaks on the downstream, to measure the momentary channel capacity for the video transmission. The passive buffer-based algorithm in DASH (c.f., Sec. 2.3.4) gradually loads the link with higher video representations, until the peaks disappear and a constant traffic profile is reached. In such a case, the momentary channel capacity of the radio link is achieved and the link is fully saturated.

4.1.2 Definition of a Margin between Segments

We introduce a notion of the margin t_M, which defines the minimal idle-time on the down-stream between every two consecutive segments requested by the client. Its purpose is to evaluate the link utilization and, thus, give us an idea of how to control the video quality for a given user. In order to provide smooth continuous video streams, our margin should not be too large. Thus, causing the client to request small segments of poor quality that quickly arrive and result in poor channel utilization (i.e., long idle time). It should not be too small either, causing the client to fully saturate the channel with large segments. Roughly speaking, an acceptable interval of the margin between every two consecutive segments should be maintained in order to allow for unexpected variances in the channel capacity, e.g., due to mobility of users, noise, and interference.

4.1.3 Discrete Fourier Transform of Throughput

We use a Fourier transform (FT) [36, 37] of traffic profiles, to derive an appropriate length of the margin. It is the most important indication of the channel utilization that controls the video quality provided to the user.

A Note about the Fourier Transform

A FT is used to transform data sequences from the time domain to the frequency domain. It allows for any integrable function $a : \mathbb{R} \mapsto \mathbb{C}$ to be represented as a combination of sinusoids localized at different frequencies f [37]: $A : \mathbb{R} \mapsto \mathbb{C}$. In our use-case, we have a discrete *signal* about the throughput and can thus use a Discrete Fourier Transform (DFT), which maps a periodic sequence of samples $a = (a_0, a_1, ..., a_j..., a_{N-1})$, where j is an integer $\in (0, N-1)$ and N is the number of samples in the sequence, to another discrete sequence of *frequency* coefficients A_k, where k $\in (0, N-1)$.

$$A_k = \sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} a_j, \tag{4.1}$$

which can be inverted to the original sampling sequence:

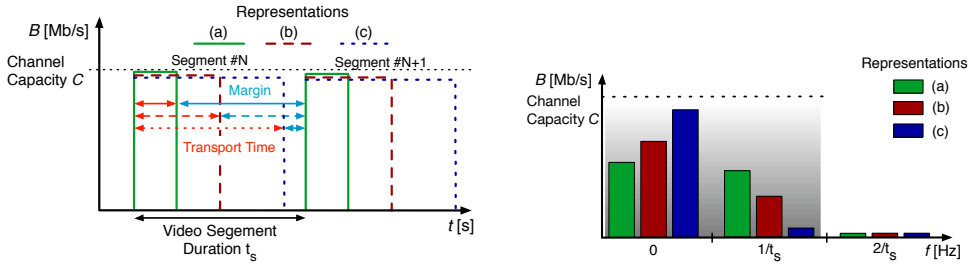
$$a_j = \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi i \frac{jk}{N}} A_k. \tag{4.2}$$

Roughly speaking, A_j displays the amount of sinusoidal behavior at the frequency $\frac{j}{N} f$ of signal a, where f is the sampling frequency.

4.1.4 Evaluation of the Margin

Let us now look at a generic example that explains the notion of the margin in video delivery. We assume that our signal (traffic pattern) is determined by the video segment duration and the periodical request of segments. We therefore work with the time-scale introduced by the video segment duration (t_s). For example, for a video segment of duration equal to $t_s = 2$ seconds, the natural frequency of the periodical behavior in the traffic profile is equal to $f_s = 0.5$ Hz.

Figs. 4.2a and 4.2b show periodical rectangle-shaped video throughput patterns and a corresponding FFT of the traffic profiles for different video qualities, i.e., each color relates to another representation. The rectangle-shaped traffic represented with continues green bars, dashed red bars, and blue dotted bars originates due to representations utilizing a channel in a low, moderate, and high manner respectively.



(a) (b)

FIGURE 4.2: (a) shows a sample throughput of different representations and (b) shows the corresponding FFT of aforementioned traffic profiles.

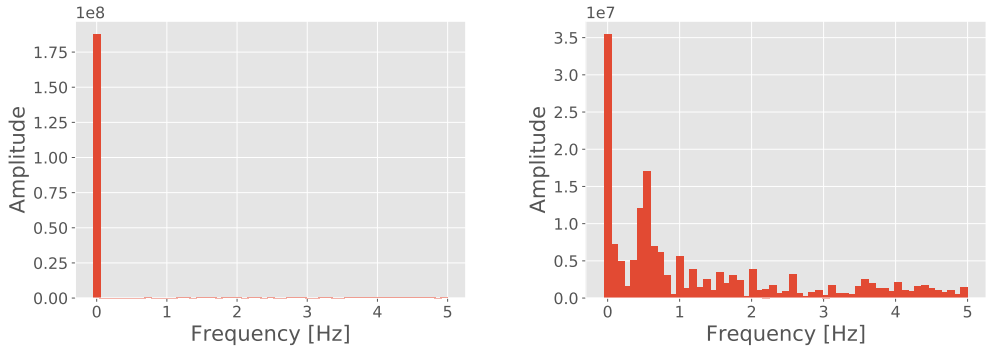
If the traffic of the video delivery is at full capacity and roughly constant, we suspect that the link is overloaded. The margin value is equal to 0 and, therefore, below our minimal acceptable values. Such traffic only contains one peak at 0 Hz in the frequency domain (c.f., Fig. 4.3 the blue representation). In such a case, we could lower the provided video quality to decrease the link utilization.

However, if the traffic periodicity is at a high level (c.f., Fig. 4.3 green representation), the margin is large, so the link may not be fully utilized, and we can still improve the quality of the video presented to the user by choosing a more traffic thirsty representation (e.g., the red representation).

In the frequency domain, the periodical rectangle pattern (c.f., Figs. 4.2a and 4.2b) contains $B[0\,\text{Hz}] = C\frac{t_s - t_M}{t_s}$ traffic at 0 Hz. The video bandwidth for higher harmonics $\frac{n}{t_s}$, where n is an integer, is equal to $B[\pm\frac{n}{t_s}] = \frac{C}{n\pi}\sin(n\pi\frac{t_s - t_M}{t_s})$ [38] pp. 257. One can verify that for small t_M values, i.e., $t_M \to 0^+$, $B[\frac{1}{t_s}]/B[0\,\text{Hz}] = 0$. Similarly for large t_M values, i.e., $t_M \to t_s^-$, $B[\frac{1}{t_s}]/B[0\,\text{Hz}] = 1$. Therefore, the fraction $B[\frac{1}{t_s}]/B[0\,\text{Hz}]$ resides between 0 and 1 for low and high values of the margin between two consecutive video segments on the downstream respectively.

We believe that the link should operate in a balance between the constant and periodical traffic (i.e., short margin), we evaluate the margin through the ratio between the amount of traffic occurring at $\frac{1}{t_s}$ Hz, where t_s denotes the segment duration, and 0 Hz (i.e., the amount of constant traffic).

To obtain the ratio of traffic $B[\frac{1}{t_s}]/B[0\,\text{Hz}]$, we use the Fast Fourier Transform (FFT) [39], which transforms discrete traffic patterns in the time domain into the frequency domain and requires $\mathcal{O}(N \log N)$ operations with respect to the number of samples.



(a) (b)

FIGURE 4.3: Figure (a) shows the FFT of best-effort transmission (b) the FFT for video transmission, c.f. Fig. 4.1 which presents the traffic profile.

Fig. 4.3a shows the FFT obtained from an example download operation of a file through the HTTP protocol, i.e., constant traffic. In our example, the FFT confirms that almost all traffic occurs at 0 Hz indicating a constant function. Comparing that to a DASH video streaming example in Fig. 4.3b (with t_s = 2 seconds), we see a certain amount of traffic at 0 Hz followed by another smaller peak at $f_s = \frac{1}{2}$ Hz = 0.5 Hz. As indicated in the previous paragraph, we use the ratio between the two peaks (amount of traffic at these two frequencies), as the indication of the margin.

According to the Shannon-Nyquist [40] theorem, in order to detect such a signal, the sampling frequency of traffic shall not be lower than $f_{\text{min}} = 2 \cdot f_s$, where f_s denotes the signal frequency, which in our case corresponds to $\frac{1}{t_s}$. As we operate with sampling frequencies of around 10 Hz, there is no risk of oversampling. This is due to the fact that the periodicity of a discrete packet-based transmission will be discovered with sampling rates of around 1000 Hz. For example, if a UE is exchanging packets of size 1500 B = 12000 bits with throughput of 12 Mbps, data packets will be received at a frequency of 1000 Hz. Sampling with frequencies higher than this upper bound could cause the FFT to shift the traffic pattern in the frequency domain from 0 Hz to around 1000 Hz, as we would have a much higher resolution. In such a case, our method using the $B[\frac{1}{t_s}]/B[0\,\text{Hz}]$ ratio would stop working. Very high samplings rates of around 1000 Hz should be therefore avoided due to the risk of oversampling.

4.2 Observation of Radio Quality Indicators

We would like to also incorporate the link quality in our algorithm. There are values reported by the UE used by the eNB to determine the radio quality and select the modulation and coding scheme (MCS) for the downlink and uplink channel. Thus radio quality measurements determine the coding rate, which then affect the maximum achievable throughput for a user. These measurements thus give us an idea on how good the radio signal connection between the eNB and UE is and what video quality shall be provided in the wireless channel.

A brief explanation of the different measured values can be found below. For a more in depth discussion of the LTE technology, consider [41].

- Reference Signal Received Power (RSRP) is defined as the linear average over the power contributions (in [W]) of the resource elements (REs) that carry cell-specific reference signals within the considered measurement frequency bandwidth [42]. In other words, it represents the average power received from a single reference signal. It is typically in a range between -44 dbm and -140 dBm.

- Reference Signal Received Quality (RSRQ) is defined as the ratio $\frac{N \times RSRP}{RSSI}$, where N is the number of resource blocks (RBs) of the LTE carrier RSSI measurement bandwidth. The measurements in the numerator and denominator shall be performed over the same set of resource blocks [42]. In other words, it measures the signal to interference ratio on the LTE reference signals and is in the range between -3 dB and -20 dB

- Receive Signal Strength Indicator (RSSI) is the wide-band received power within the relevant channel bandwidth [42].

- Channel Quality Index (CQI) is computed by the UE and reported on the uplink channel to the eNB. It indicates the modulation and coding rate, which can be handled by the device in the momentary state and is related to the signal to noise / interference ratio. It is mapped to a range between 0 and 15.

- Modulation and Coding Scheme (MCS): Depending on the CQI reported by the UE, the eNB selects the MCS, which determines the modulation and spectral efficiency, essentially determining the maximum

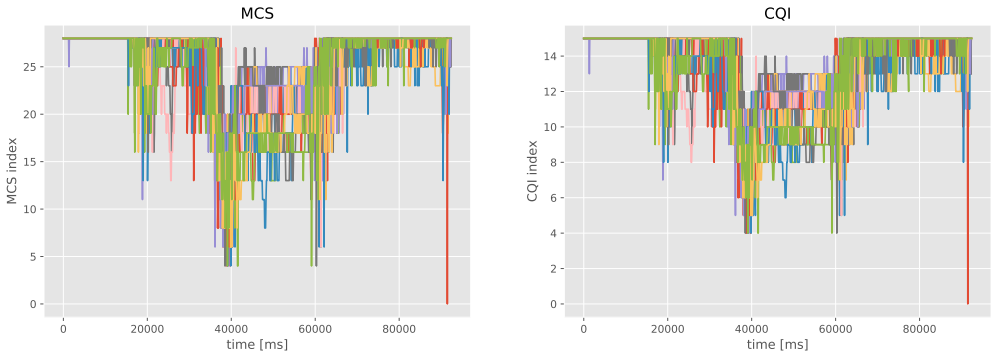
throughput between the UE and eNB.

TABLE 4.1: The radio signal quality according to the measurement of RSRP and RSRQ

		RSRQ [dB]	**RSRP [dBm]**
Signal Quality	Excellent	> -5	> -84
	Good	-6 to -10	-85 to -102
	Fair	-11 to -15	-103 to -111
	Poor	< -15	< -112

In our initial measurements, we found that RSRP is the most reliable metric that is used in LTE handovers [43] (c.f., Fig. 4.4). We decided that RSRP matches our needs as a good and valid indication of channel quality that could control the video transmission, i.e., scaling the video representations up and down. In order to to this, we use a simple threshold mechanism degrading video quality if a heavy drop in the measured RSRP was lately detected. However, other mechanisms such as restricting certain video qualities completely depending on the RSRP values, e.g., in poor radio signal conditions could be considered. It could, however, waste resources avoiding streaming of high quality representations. Unlike other contributions [23, 11], we do not use CQIs to assess the channel capacity. The problem of matching CQIs to channel capacity depends on technology, frequency, environment, vendor, radio scheduling, and hardware. Therefore, it is not feasible to derive exact tables matching CQIs and the desired rate of video delivery in a mobile system. Moreover, the use of CQIs does not apply to congested networks, in which the data-rate is limited by the system capacity (e.g., the number of available PRBs was reached).

(a) (b)



(c) (d)

FIGURE 4.4: The results from different test runs comparing the RSRP (a), RSRQ (b), MCS (c) and CQI (d)

4.3 Algorithm for Restricting Video Qualities in DASH

In previous sections of this Chapter, we elaborated on properties of DASH-based video delivery in mobile networks (c.f., Sec. 4.1 and Sec. 4.2). This section now describes an algorithm that processes information obtained from RNIS and controlling the video quality provided towards end-users.

Algorithm 3 Simplified algorithm that is based on radio link utilization and received radio signal power.

procedure MAIN LOOP(T, i) ▷ Duration T of video and interval i of sampling

$t \leftarrow 0$ ▷ Counter variable running from 0 to T/i
$x \leftarrow$ segment duration
delay $\leftarrow$ time period (e.g., 1 sec)
$max_\Gamma \leftarrow$ max tolerated RSRP drop (e.g., 5)
$min_\Delta \leftarrow$ min margin (e.g., 0.04)
$max_\Delta \leftarrow$ max margin (e.g., 0.4)

▷ As long as the video is playing
while $t <= \frac{T}{i}$ **do**
    $S_t =$ getStats() ▷ Get stats from RNIS

    if $S_t.rsrp - S_{t-\frac{delay}{i}}.rsrp > max_\Gamma$ **then**
        Video representation scale down; (c.f., Fig. 4.5a)

                ▷ Run a FFT over the last sampled data with rate i
    fft = runFFT(
        $S_{t-\frac{x}{i}}.throughput, S_{t-\frac{x}{i}+1}.throughput, ...,$
        $S_t.throughput$, i)

    $\text{fft}_0 = \text{fft}.0$
    $\text{fft}_{\frac{1}{x}} = \text{fft}.\frac{1}{x}$

    if $\frac{\text{fft}_{\frac{1}{x}}}{\text{fft}_0} >= max_\Delta$ **then**
        Video representation scale up; (c.f., Fig. 4.5b)
    if $\frac{\text{fft}_{\frac{1}{x}}}{\text{fft}_0} <= min_\Delta$ **then**
        Video representation scale down; (c.f., Fig. 4.5a)

    $t = t + 1$
    sleep(i)

Algorithm 3 is a novel algorithm, which controls the available video qualities for a user over time. To summarize, it limits the available video representations, if the RSRP decreases more than max_{Γ}, a heuristically determined threshold, in a given time interval or the margin between the transmission of two segments is below a predefined threshold min_{Δ} indicating that the radio link is overloaded and the video quality should be decreased. It also increases the video quality, if the margin is above a certain threshold max_{Δ} meaning that the link is idle. If any of the above mentioned conditions is fulfilled, then the users MPD file is modified according to a "scale-down" or "scale-up" action. In the scale-up operation, the lowest available video quality is replaced with the next higher, not yet included video quality (c.f., Fig. 4.5b). Similarly, scale-down refers to the process of replacing the highest video representation with the next lower one available (c.f., Fig. 4.5a).

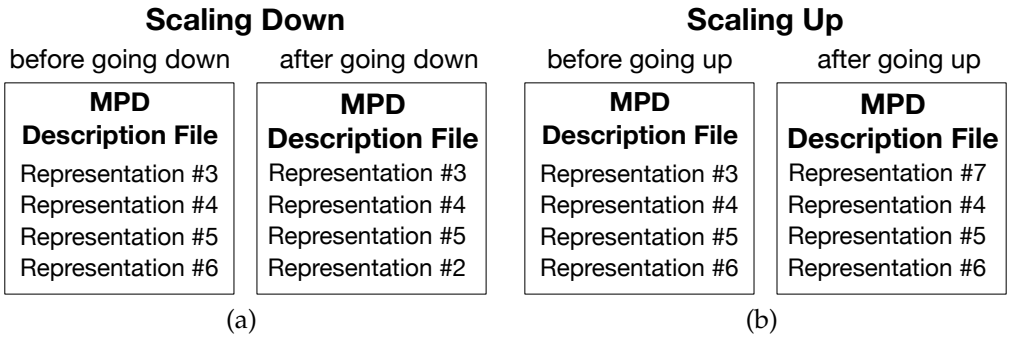


FIGURE 4.5: The concept of going down (a) and up (b) with available video qualities.

Please note, that in Fig. 4.5 the representations in the MPD file refer to video qualities and bitrates required by corresponding representations in an ascending order, i.e., #1 $\prec$ #2 $\prec$ #3 ...and the lowest quality is provided by representation #1.

In our measurements, we identified all the aforementioned thresholds heuristically (c.f., Sec. 5.1).

Potentially, machine learning approaches could be used as well to control the quality of the video definition. These approaches, however, are out of the scope of this work, but could be considered in future research (c.f., Sec. 7.2).

Chapter 5

Experiment Design and Target Values

This Chapter provides an overview of the setup, explains the motivation of our experiments (c.f., Sec. 5.1), and discusses the measured and calculated values (c.f., Sec. 5.2).

5.1 Experiment Design

In our experiments, we work in a real-world radio scenario using a setup as presented in Sec. 3.4. We establish an OpenAirInterface-based [27] LTE eNB providing cellular coverage in LTE FDD Band7 (2.5 GHz / 2.6 GHz) spanning an office space. We evaluate a femto-cell scenario (e.g., an office space, train station, etc.) as a user moves inside a building and the eNB antenna output power is small and equal to 10 dBm[1]. At the moment, due to setup limitations, the eNB and video service are only handling one user, but we expect to apply our work without any changes to multi-user scenarios. We do not expect interactions between our algorithm and the radio scheduler, as the eNB establishes the shared channel for every user independently using an arbitrary scheduling algorithm [44] operating at the granularity of the LTE Transmission Time Interval (TTI), which is 1 ms in the case of LTE FDD, and, thus, shall not introduce periodicity at the level of video segment durations. Furthermore, we believe that our work is equally well applicable in macro-cells, i.e., when the radio coverage is provided by an antenna of higher power (e.g., BTS) over a larger area, while our technique shall not be influenced by the scale of the setup.

[1] A typical WiFi Access Point provides an output transmission power of 15 dBm.

In the experiment a notebook is provided with Internet access through tethering (i.e., the smart phone provides WiFi signal for the notebook). The smart phone itself is connected to the mobile network and receives the LTE signal from the eNB (c.f., Table 5.1). The UE, i.e., smart phone, is then moving during 90 seconds inside an office and a hallway according to a predefined moving pattern (c.f., Fig. 5.1). At the same time the video is streamed on the notebook from the video service, which resides in the ME cloud, and the RNIS is regularly assessing the channel occupancy / quality.

TABLE 5.1: The specification of the devices involved in our experiment.

Device	**CPU**	**RAM**	**Usage**
Farnsworth	i7-4790 CPU @ 3.60GHz	32 GB	running OAI-eNB
Dell R530 servers	48 Intel Xeon @ 2.5 GHz	192 GB	running the OAI-EPC, FlexRAN and the video service
Moto G2[2]	1.2 GHz	1 GB	used to connect to the eNB
HP Pavilion	i7-720QM CPU @ 1.6 GHz	8 GB	tethered to the phone, streaming a video from the video server using MP4Client

The movement of the user is depicted in Fig. 5.1. We stay at point A at 0 to 15 s, move from A to point B in 5 s, stay in point B between 20 and 35 s, and move from B to C in 6 s. Then, we stay in C between 41 and 60 s, before we go back to B again and stay there between 66 and 90 s. At each point we experience different radio signal levels and qualities, so that on the way between A to C, and C to B, we experience decreasing and increasing signal levels (c.f., Table 5.2) respectively.

Finally, we use two different client-side adaptation algorithms (i.e., throughput- and buffer-based).

i) We run an experiment for edge DASH streaming, in which the video server is instantiated on the MEC cloud, provides a static MPD file without any adaptations from the MEC SD-RAN controller. We will call it regular DASH, even though the video server resides at the mobile edge.

ii) We also examine MEC-assisted DASH streaming, in which the video server, again residing at the mobile edge, periodically updates the MPD file according to the algorithm specified in Sec. 4.3. The algorithm uses the information delivered by the MEC SD-RAN controller. For simplicity, we use the

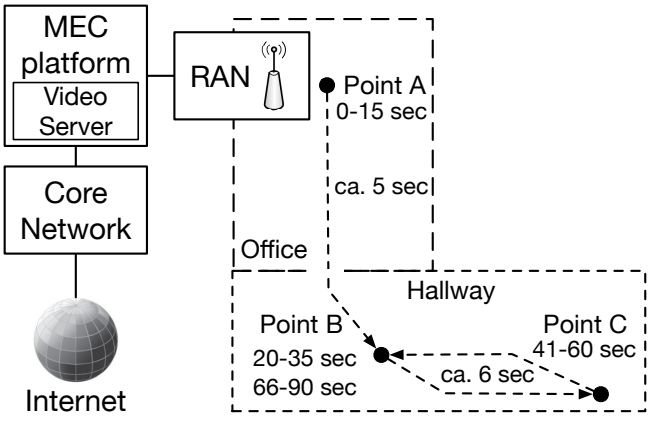


FIGURE 5.1: The movement of the user in the femto-cell, simulating various radio signal conditions.

TABLE 5.2: Description of the different points and experienced signal quality and throughput.

Point	**Description**	**Average RSRP**	**Average RSRQ**	**Max. Throughput**
A	Starting point and closest point to the base station, provides the best signal quality.	-86 dBm	-9 dB	9 Mb/s
B	The intermediate point, as the signal is blocked by some obstacle (e.g. walls) we received a lower signal strength and also a lower signal quality.	-103 dBm	-7 dB	8 Mb/s
C	The furthest point from the antenna, provided the lowest signal quality and strength.	-112 dBm	-9 db	6.5 Mb/s

notion *regular DASH* to refer to regular throughput- and buffer-based DASH streaming, where, however, the video server is already residing in the MEC cloud, and *MEC DASH* to refer to our improvements using the novel MEC-assisted approach presented in this work.

In our algorithm (c.f., Sec. 4.3), we use the following set of parameters: $delay = 1$ s and $max_\Gamma = 5$ for the RSRP degradation and $max_\Delta = 0.4$, $min_\Delta = 0.04$ for the margin settings. We use the DASH video mentioned in Sec. 3.8 with segment duration of 2 seconds each. Please consult Chapter 4 for a more detailed explanation of radio quality variations and the required margin parameters in DASH-video delivery.

Due to a high variance of DASH streaming, we compare the average values of our target values on a statistical basis (c.f., Sec. 5.2) for a series of ten measurements for each streaming technique. Please note, that our measurements and comparison is very challenging, as we compare two methods of video delivery residing very close to the user (i.e., network edge).

5.2 Target Values for the Evaluation

Our target is to improve the video quality provided to the user. In static networks of constant bandwidth and latency, there is no problem of providing a smooth, continuous stream of high quality video segments. However, in mobile situations, we expect the varying network conditions, highly depending on the radio signal quality received by the user. As the target values for our measurements, we identify the video quality presented to the user (i.e., representation with respect to Table 3.1) as well as the buffer fill level (i.e., the pre-fetched video content available at the client for a couple of subsequent seconds), which influences on how much the stream is vulnerable to a stall. These two values together present an objective measurement of the video quality provided, which highly impacts on the user perceived Quality of Experience (QoE).

In the evaluation of our work, we consider the Adaptability (A) and Adaptation Frequency (AF) as defined by Karagkioules et al. [33] (c.f., Equations 5.1 and 5.2).

$$A = \frac{1}{K} \sum_{i=1}^{K} \frac{R_i}{\min(R_N, \overline{C_i})} \tag{5.1}$$

In Eq. 5.1, $\overline{C_i}$ denotes the average throughput available to the client, R_i stands for the bit rate of the i-th segment and R_N is the bit rate of the lowest representation, i.e., lowest video quality. In optimal video delivery using an optimal bit rate, A should be 1. Generally speaking, A values above 1 represent too aggressive algorithms demanding too much throughput over-saturating the link, while values below 1 represent too passive algorithms, which suboptimally utilize the channel capacity.

$$AF = \frac{\sum_{i=1}^{K-1} (1 - \delta_{R_i R_{i+1}})}{K} \tag{5.2}$$

In Eq. 5.2, δ is the *Kronecker delta*, which returns 1 if $R_i = R_{i+1}$ and 0 otherwise; K is the total number of streamed segments. This value shows how often the algorithm changes the video definition providing a variable video quality towards end-users, which may negatively impact on the QoE. AF resides close to 0 for optimal solutions. Values close to 1 are received for very poor solutions.

Chapter 6

Results

In this Chapter, we discuss the results of our measurements. Our experiments are performed on a setup presented in Sec. 5.1. We discuss and compare the state-of-the art adaptation algorithms against our MEC-assisted adaptation techniques. First, we present results of each adaptation algorithm running 10 times on a statistical basis. Second, we compare performance of different algorithms using statistical analysis.

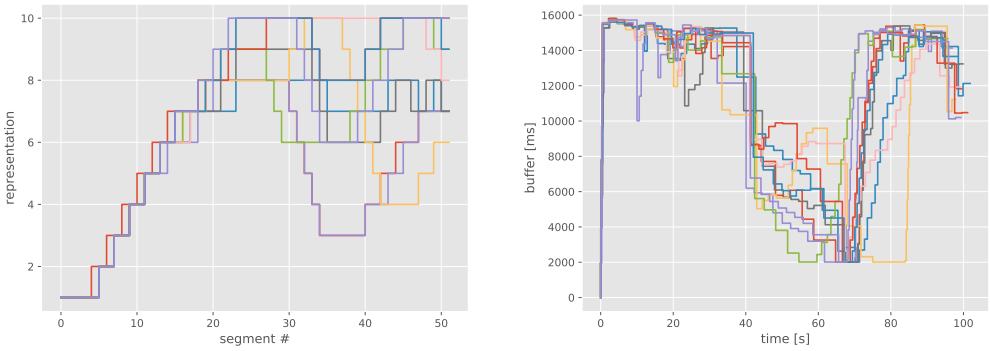
6.1 Experiments with Buffer-Based DASH

In the subsequent sections, we present the results obtained by the buffer-based adaptation algorithm as described in Sec. 2.3.4. We expect that the buffer-based adaptation provides a reasonable buffer fill level combined with a passive (i.e., not aggressive) request for segments of extended video definitions.

6.1.1 Regular Buffer-Based DASH

In this section we discuss the results from buffer-based DASH. Although, the video server resides at the network edge, we call this regular buffer-based DASH, as we do not take the mobile edge capabilities into account. Fig. 6.1a shows the quality of received segments against the segment number and Fig. 6.1b illustrates the buffer fill level against time. Notice that video segment length is equal to 2 seconds. Therefore, the time in an experiment roughly translates to segment number according to t = segment number $\cdot$ 2 s. First, we notice that the progression in video quality provided is similar in

all experiments until segment #20. Please notice that due to a passive buffer-based adaptation, we only experience a step-by-step increase of the video quality. After, we experience a huge variance in the video quality as well as buffer fill level in individual measurements (c.f., Fig. 6.1). The necessary bandwidth is still large enough when moving between points *A* and *B*. We, therefore, keep the buffer fill at a steady level up to 35 seconds (c.f., Fig. 6.1b), but the video quality constantly increases up to segment #25 (c.f., Fig. 6.1a). However, when the video quality becomes good enough, i.e., saturating the link, we witness a steep decline of the buffer fill level, when moving towards location *C* with bad reception (c.f., Fig. 6.1b between 40 and 60 seconds). As the buffer level decreases, we become more prone to a video stall, and the client begins to reduce the video quality (c.f., Fig. 6.1a from segment #30 on) to quickly refill the buffer. Finally, between 60 and 70 seconds, we move back towards point *B* and the signal quality increases. The buffer level is then quickly refilled and kept at a stable level.



(a) (b)

FIGURE 6.1: (a) shows the requested segment quality against the segment number and (b) illustrates the buffer fill level over time for each experiment in buffer-based DASH.

Figs. 6.2a-6.2d show the measured channel quality and throughput over time. Again, each color represents one measurement. We can see that all measurements of RSRP values stick together (c.f., Fig. 6.2a), while other signal quality indicators, such as RSRQ or CQI vary a lot (c.f., Figs. 6.2b and 6.2c). This corresponds to the findings of our first measurements, where we observed the behavior of the signal quality indicators for a a given moving pattern (c.f. 4.4). Furthermore, in Fig. 6.2d, the individual periodical throughput peaks (c.f., Sec. 4.1) are clearly visible.

Table 6.1 provides a summary of the experiment showing the most important metrics captured with their minimum, average, and maximum values for all

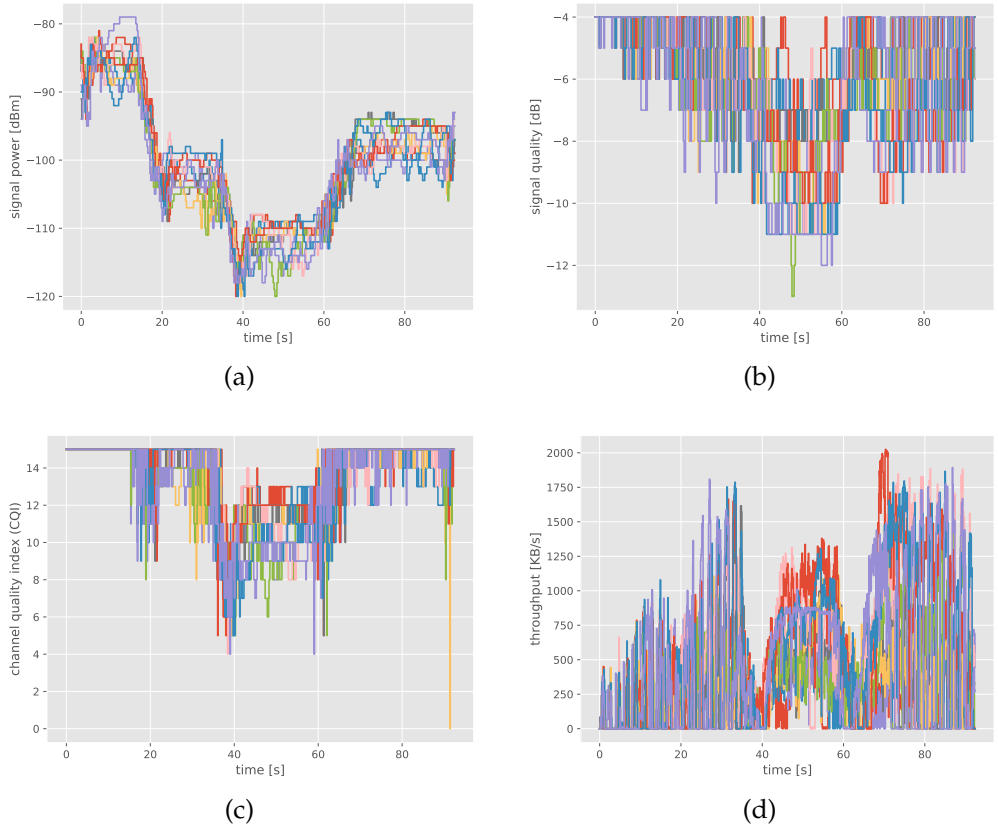


FIGURE 6.2: (a) shows the RSRP, (b) the RSRQ, (c) the CQI value, and (d) shows the average throughput over time respectively.

of 10 buffer-based DASH experiments. The upper half of the table provides values obtained from the GPAC MP4Client video player, while the lower half contains values obtained from the SD-RAN FlexRAN platform.

TABLE 6.1: The results obtained by regular, buffer-based DASH.

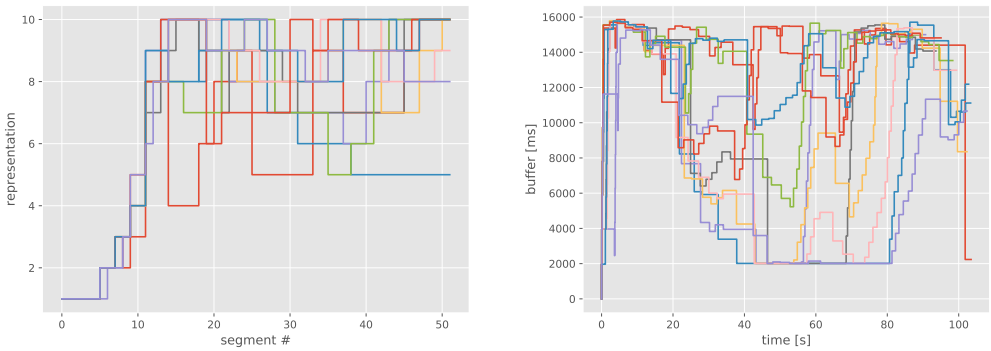
	minimum	**average**	**maximum**
Representation [ID]	1	6	10
Buffer size [ms]	1966	10332	15825
Adaptability	0.3158	0.4071	0.6545
Adaption Frequency	0.2	0.2753	0.3272
RSRP [dBm]	-120	-101	-79
RSRQ [dB]	-13	-7	-4
CQI	4	13	15
MCS	4	25	28
Throughput [B/s]	0	579980	2025937

6.1.2 MEC-Assisted DASH

In the subsequent sections, we study the buffer-based adaptation algorithm using our MEC-assisted algorithm (c.f., Sec. 4.3) with and without the consideration of the radio signal quality, but we always consider the margin (c.f., Algorithm 3).

MEC-Assisted DASH with Margin and Signal Consideration

We repeat the same experiment 10 times (c.f., Sec. 6.1.1) , but this time, we study our novel MEC-assisted approach using the margin and signal quality as an indication that helps us to control available representations. Again, we plot the representation and buffer fill level against segment number and time respectively (c.f., Fig. 6.3). We observe that the variance between individual measurements increases. However, sometimes, we experience lower video qualities (c.f., Fig. 6.3a, e.g., red line seg. #15). The video quality generally tends to improve faster (c.f., Fig. 6.3a) than in the case of regular DASH (c.f., Fig. 6.1a). In Fig. 6.3b, we also notice an almost empty buffer throughout an extended period of time compared to the previous measurements in regular buffer-based DASH (c.f., Fig. 6.1b). Furthermore, the buffer fill level shows a high variance compared to regular buffer-based DASH (c.f., Figs. 6.3b and 6.1b). It is not possible to compare individual experiments directly, so we average them to allow for a fair comparison (c.f., Sec. 6.1.3).



(a) (b)

FIGURE 6.3: (a) shows the received qualities of each segment and (b) the buffer fill level over time.

Considering signal qualities and throughput measured, c.f., Fig 6.4, we again notice a similar pattern in link-related values (c.f., Sec. 6.1.1), i.e., de-/increasing power-related profiles depending on the position of the user and

also the DASH specific traffic pattern (c.f., Sec. 4.1, which is enforced by the algorithm keeping an appropriate buffer level.

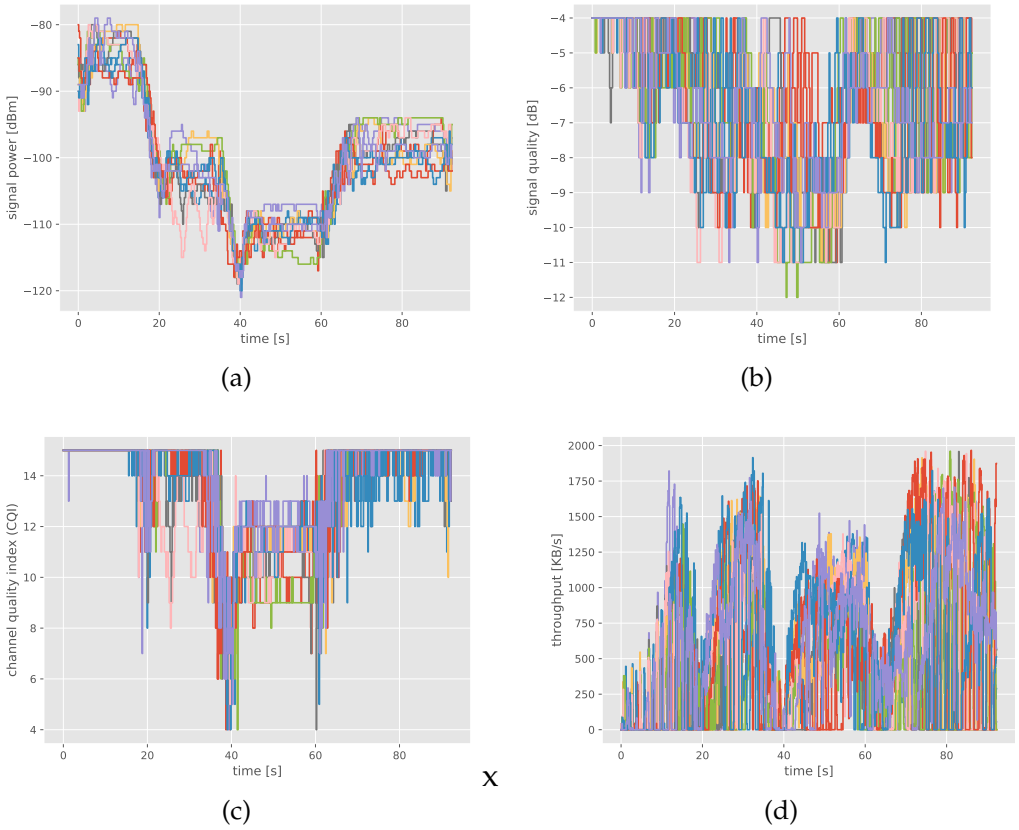


FIGURE 6.4: (a) shows the RSRP, (b) the RSRQ, (c) the CQI value, and (d) the average throughput over time respectively.

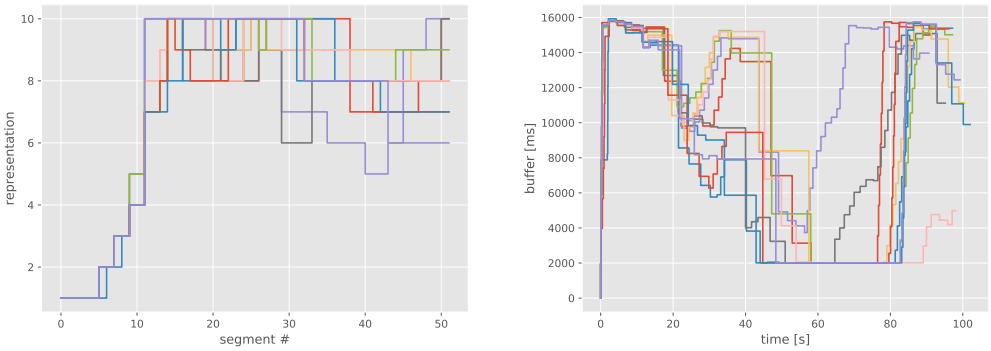
TABLE 6.2: The results obtained by MEC-assisted, buffer-based DASH.

	minimum	**average**	**maximum**
Representation [ID]	1	7	10
Buffer size [ms]	1966	9539	15857
Adaptability	0.4	0.4603	0.5472
Adaption Frequency	0.1923	0.2213	0.2641
RSRP [dBm]	-121	-101	-79
RSRQ [dB]	-12	-7	-4
CQI	4	13	15
MCS	4	25	28
Throughput [B/s]	0	734374	1966767

MEC-Assisted DASH with Margin Consideration Only

As before (c.f., Sec. 6.1.2), we run the same experiment 10 times with a buffer-based client adaptation. This time, we use the MEC-assisted algorithm (c.f., Algorithm 3) with the margin consideration, but we do not rely on the signal quality condition (c.f., Algorithm 3 line 14-15) to adapt available video qualities towards the client. We keep the same parameter values in all experiments. However, this time max_{Γ} and delay, used to indicate the maximum derivation of the signal quality, are ignored, as they are not considered by the studied algorithm.

Fig. 6.5a shows that the video stream experiences less variances in the quality of video segments compared to MEC-assisted DASH with margin and signal consideration (c.f., Fig. 6.3a from segment #10 to #28). However, the buffer fill level in the MEC-assisted DASH with margin consideration only, suffers from network overload (c.f., Fig. 6.5b) as the algorithm provides video representations above the network capacity and, thus, leads to low buffer fill levels and possible stalls (c.f., Fig. 6.5b between 40 and 80 seconds).



(a) (b)

FIGURE 6.5: (a) displays segment quality against segment number and (b) the buffer fill level over time in MEC-assisted DASH with margin consideration only.

Again, measurements of the signal quality are provided in Fig. 6.6. Most of individual experiments show a big variance. However, the received RSRP values tend to stick closely together. Furthermore, Fig. 6.6d shows a fully saturated link. Thus, the video service provides too high video qualities during the experiment (except the beginning).

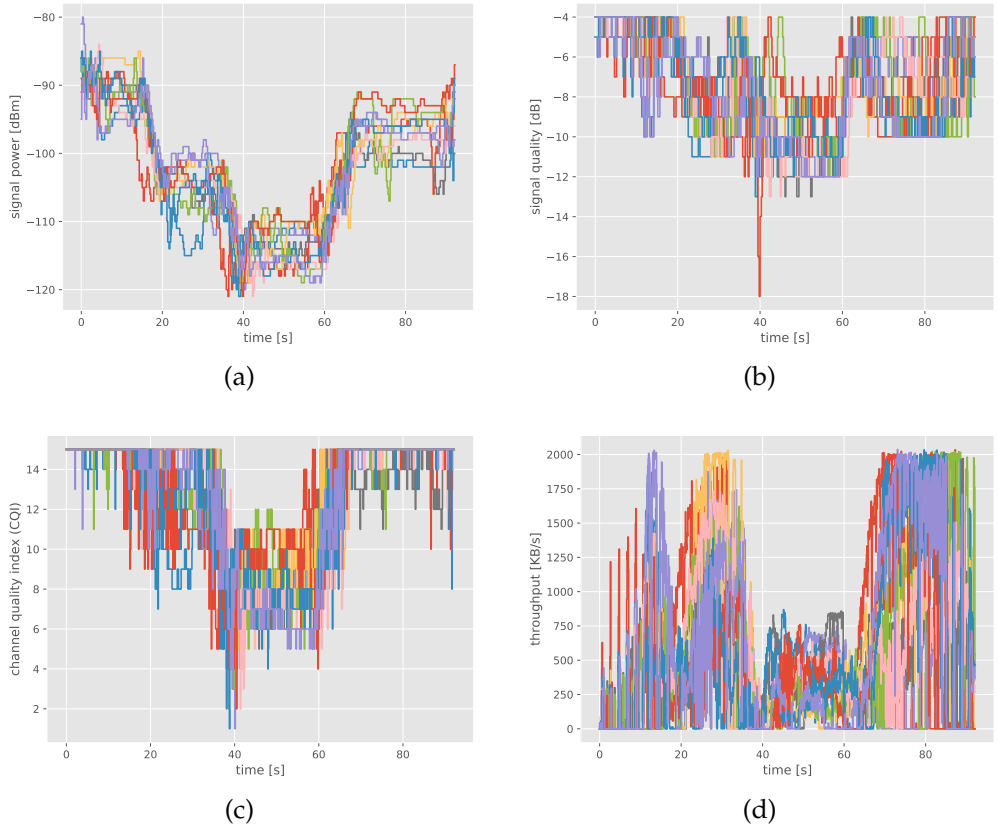


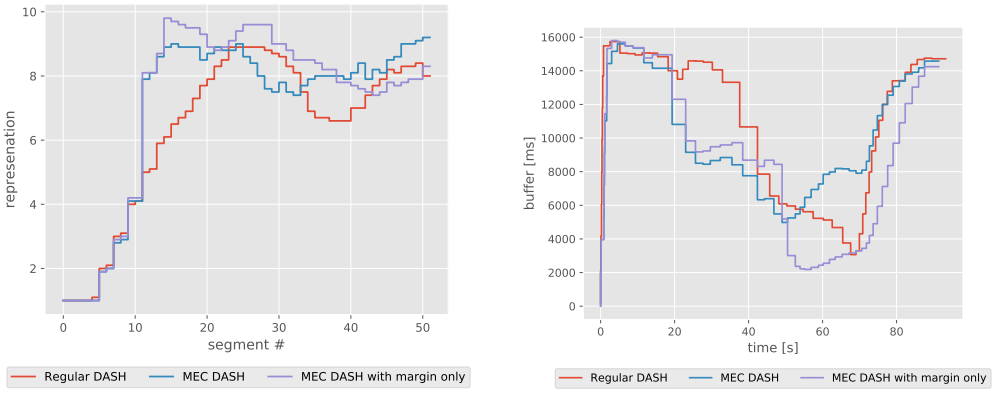
FIGURE 6.6: (a) displays the RSRP, (b) the RSRQ, (c) the CQI value, and (d) the average throughput over time respectively.

6.1.3 Comparison of Regular DASH and MEC-Assisted DASH using Buffer-Based Adaptation Technique

As the variance of individual measurements is very high in terms of video quality and buffer fill level, we use the statistical average over 10 measurements to make a fair comparison between buffer-based DASH with and without the assistance of MEC. Fig. 6.7a shows the average representation against the segment number, while Fig. 6.7b illustrates the buffer fill level over time.

We can clearly see the improvement provided by MEC-assisted DASH delivering the extended video quality in the first segments (c.f., Fig. 6.7a between segment #10-#20), which results in a faster adaptation to the network capacity in the beginning than for regular, passive DASH. However, MEC-assisted DASH then experiences a lower buffer level, which equals to 50% of the buffer level experienced by regular DASH, c.f., Fig. 6.7b between 25-40 seconds, which happens when the UE stays in *B* or moves from *B* to *C* and when

a medium/poor signal quality occurs. As a result, MEC-assisted DASH with signal consideration pro-actively decreases the video quality and, thus, keeps the buffer level stable or even increases it from 40 seconds onwards (c.f., Fig. 6.7b). While both, regular DASH and MEC-assisted DASH with margin consideration only (i.e., without taking RSRP into account) fail to anticipate the decreasing radio signal conditions and, thus, continue to deliver segments of high quality for a longer period of time, MEC-assisted DASH with margin and signal consideration can (on average) keep the buffer at a higher level (c.f. Fig 6.7b between 50 and 70 seconds). Using MEC-assisted DASH, we not only keep the buffer at a reasonable fill-level during the mobility phases (c.f., Fig. 6.7b between 20-60 seconds) but keep the video quality at a more stable level (c.f., Fig. 6.7a segment #25-#40), resulting in a less aggressive adaptation (c.f., Sec. 6.3). Thus, over the entire experiment, we can provide a better video quality and QoE to the user in comparison to the regular DASH.



(a) (b)

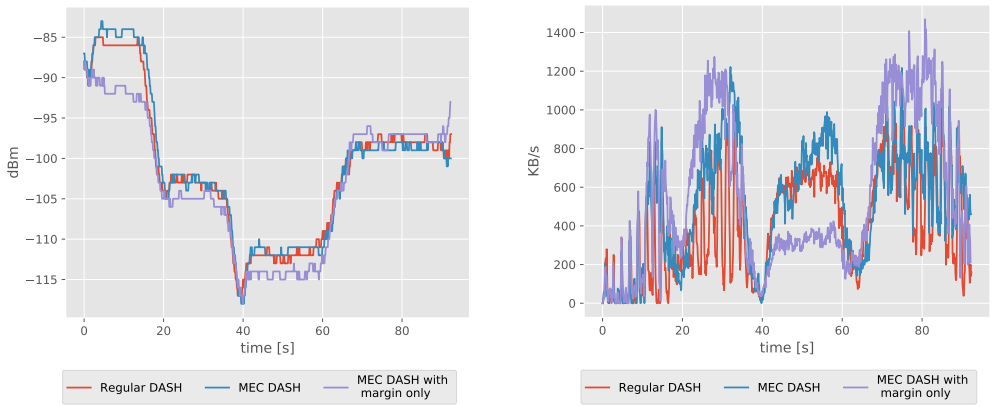
FIGURE 6.7: The average segment quality (a) and buffer fill level (b) from the different adaptation techniques using buffer-based streaming.

Fig. 6.8 displays similar average RSRP values, but different traffic patterns depending on the adaptation algorithm considered. Please notice that the experiment titled "MEC DASH margin only", i.e. without taking into account RSRP values, c.f., Fig 6.14a, turquoise line, was carried out at a different moment than other experiments, so the signal profile differs. We were unable to tune the setup to receive the same signal profile as before. This can be explained by the position of the eNB antenna or other environmental factors. For example, the antenna could slightly change the position in comparison to previous experiments (c.f., Fig. 5.1) and, thus, we experience a less powerful signal at same points. However, our "MEC DASH margin only" experiment

does not consider signal qualities and, therefore, the impact of the decreased received power shall be minimal.

Furthermore, in Fig. 6.14b, the peaks in the traffic pattern are less visible, as the delivery of the segments in the different experiments may be a little shifted, due to different download speeds and segment qualities and, thus, result in a wider peak.

We witness differences among the considered adaptation algorithms. Regular DASH and MEC-assisted DASH provide similar traffic profiles except of transmission beginnings (i.e., the reception of a few first segments in the video streaming). "MEC DASH with margin only" always provides a too high video quality and, thus, saturates the link (c.f., Fig. 6.14b). We suspect that the over-provisioning of video quality could be prevented by using different parameters in our algorithm (c.f., Algorithm 3). However, no tests were performed in this particular research direction.



(a) (b)

FIGURE 6.8: The averaged values for the signal quality (a) and throughput (b) measurements obtained by each individual adaptation algorithm using buffer-based streaming.

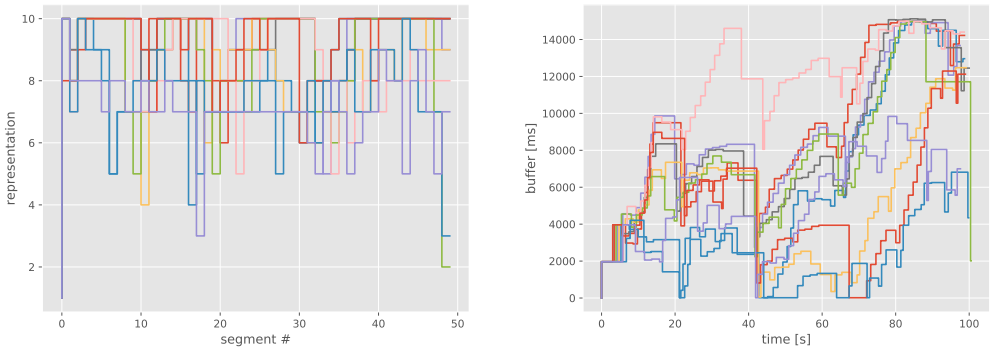
6.2 Experiments with Throughput-Based DASH

Again, we perform the same experiment (c.f., Sec. 6.1.1 and 6.1.2) for throughput-based adaptation. We studied regular state-of-the-art DASH and

our novel MEC DASH approach. As Karagkioules et al. [17] suggest, we expect a much more volatile video quality for throughput-based DASH. Furthermore, the video quality may be oscillating around the optimal quality and the buffer fill level shall be low values. This adaptation algorithm could have a negative effect on the perceived QoE, but we also compare it against our MEC-assisted adaptation.

6.2.1 Regular Throughput-Based DASH

Fig 6.9a shows the video quality against segment number and Fig. 6.9b displays the buffer fill level against time. We immediately notice that the video quality is much more volatile, changing rapidly, and maintaining constant values for only a few segments. Furthermore, we notice that the buffer is almost never filled completely, and we, therefore, experience stalls, c.f., Fig. 6.9b, e.g., blue line between 40 and 70 seconds. We also notice that the drop in the buffer level is clearly correlated with mobility in our experiments, i.e., the buffer decreases mostly when a UE moves towards points of worse reception and increases when moving towards regions of better radio signal quality (c.f., Fig. 6.9b at 20, 40 and 65 seconds).



(a) (b)

FIGURE 6.9: (a) shows the requested segment quality against the segment number and (b) displays the buffer fill level over time for each experiments in throughput-based DASH.

Figs. 6.10a-6.10d show the radio signal quality and throughput as reported by FlexRAN. We notice that the signal quality is very similar for all measurements; the plots may be a little bit different, i.e., shifted, as we do not move exactly at the same time or stay at exact same positions. As in previous experiments (c.f., Sec. 6.1.1 and 6.1.2), Figs. 6.10a-6.10d provide a good insight

on the received signal quality. Fig. 6.10a shows the average throughput in a 0.1 seconds interval. We can clearly see that the traffic increases quickly at the the beginning and looks similar to our MEC-DASH approach with the buffer-based adaptation (c.f., Figs. 6.4d and 6.10d). Furthermore, the peaks of the individual segment requests are no longer visible and the margin between them almost disappear, which indicate that we momentarily overload the channel.

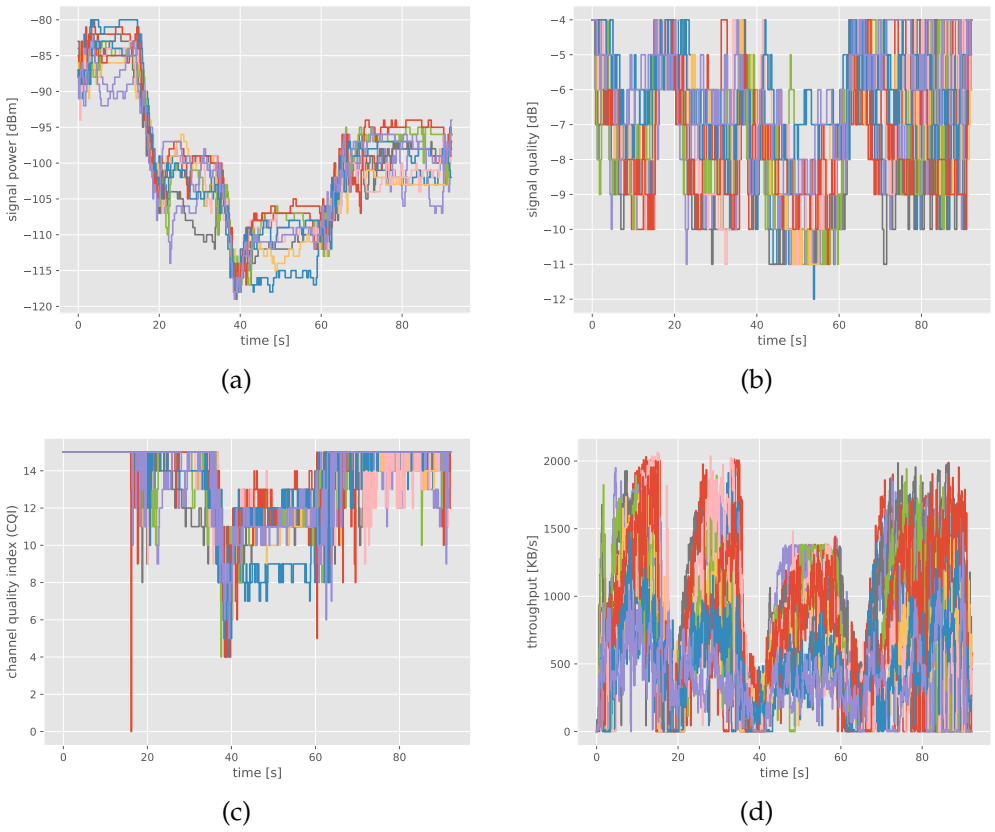


FIGURE 6.10: (a) displays the RSRP, (b) the RSRQ, (c) the CQI value and (d) the average throughput over time respectively.

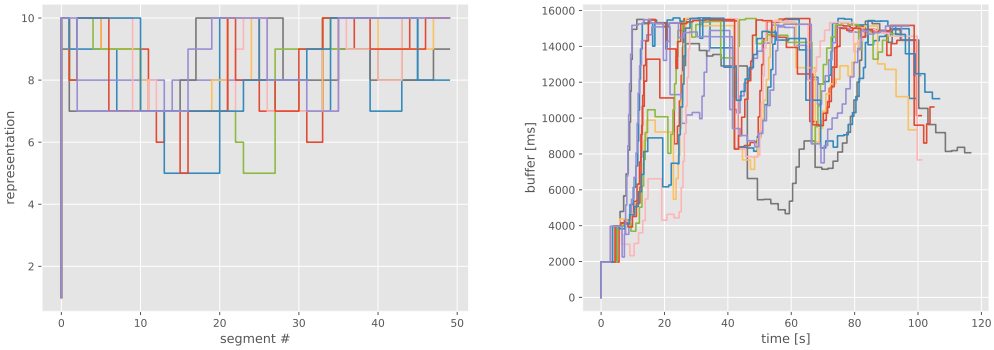
TABLE 6.3: The results obtained by regular, throughput-based DASH.

	minimum	**average**	**maximum**
Representation [ID]	1	8	10
Buffer size [ms]	0	6710	15122
Adaptability	0.36	0.6591	0.88
Adaption Frequency	0.3272	0.4539	0.58
RSRP [dBm]	-119	-101	-80
RSRQ [dB]	-12	-8	-4
CQI	4	13	15
MCS	4	25	28
Throughput [B/s]	0	833391	2065702

6.2.2 MEC-Assisted DASH with Margin and Signal Consideration

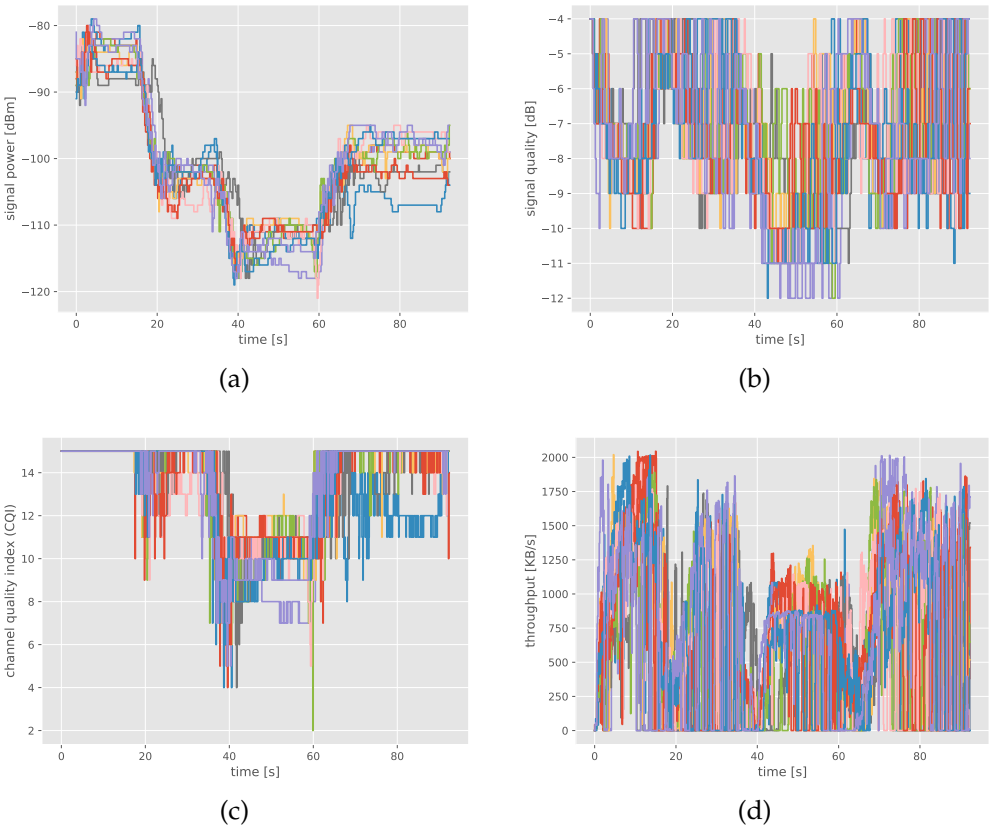
Fig. 6.11a shows the video quality of the individual segments and Fig. 6.11b displays the buffer fill level over time. Undoubtedly, we find that the video quality is much less volatile and more stable (c.f., Figs. 6.9a and 6.11a). Furthermore, on average, we experience a good video quality, and as we do not over-provision video quality, the buffer level is kept at a significantly higher level. Overall, the video quality and quality of experience is satisfying for the user, as in none of the experiments, we get close to stalling the video; the video streams continuously in a quality above representation #5 (c.f., Fig. 6.11a). It is, however, noticeable that we always experience a drop of about 50% in the buffer level, as we are moving away from the eNB, c.f., Fig. 6.11b upon a move between points *B* and *C* at around 40 seconds as well as between points *C* and *B* shortly after 60 seconds. However, due to the enforcement of the margin and the consideration of the signal quality in the MEC-assisted adaptation, we can increase the overall video quality.

In Figs. 6.12a-6.12c, we display the changes of the signal quality over time. Again (c.f., Sec. 6.1.1 and 6.1.2), we witness that the performance of the RSRP value (c.f., Fig. 6.12a) provides the best estimation of the signal quality. We notice that in Fig. 6.12d, the individual traffic peaks are much more visible. This indicates that, indeed, our margin is kept at the reasonable level and the video service does not overload the link with a too high video quality.



(a) (b)

FIGURE 6.11: (a) displays the representation of the individual segments and (b) shows the buffer fill level over time.



(a) (b)

(c) (d)

FIGURE 6.12: (a) shows the development of the RSRP, (b) of the RSRQ and (c) the CQI value over time respectively. (d) shows the average throughput over a 0.1 seconds interval.

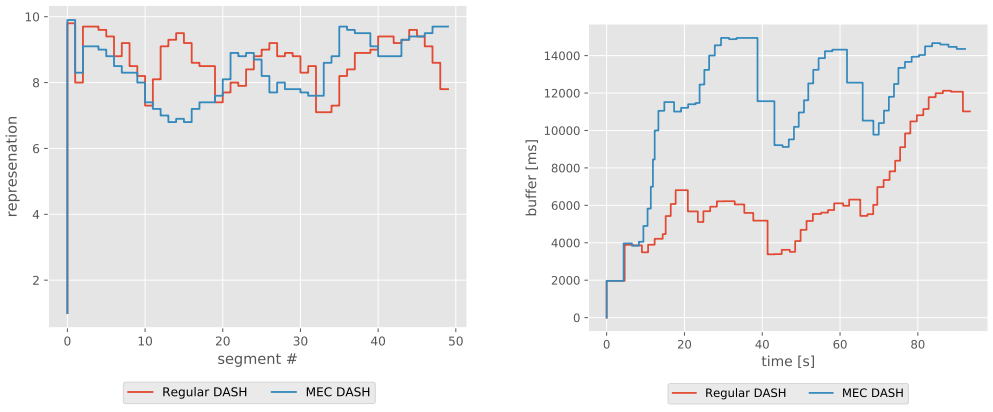
TABLE 6.4: The results obtained by MEC-assisted, throughput-based DASH.

	minimum	**average**	**maximum**
Representation [ID]	1	8	10
Buffer size [ms]	1966	11080	15587
Adaptability	0.3725	0.5001	0.5961
Adaption Frequency	0.1296	0.2025	0.2692
RSRP [dBm]	-121	-101	-79
RSRQ [dB]	-12	-8	-4
CQI	2	13	15
MCS	1	24	28
Throughput [B/s]	0	871253	2045224

6.2.3 Comparison of Regular DASH and MEC-Assisted DASH using Throughput-Based Adaptation Technique

Again (c.f., Sec. 6.1.1 and 6.1.2), we use the statistical average over 10 individual measurements to compare throughput-based DASH against throughput-based MEC-assisted DASH. In Fig. 6.13a, we display the average representation quality against segment number and in Fig. 6.13b we show the buffer fill level against time. Fig. 6.13b shows that upon mobility (from 15-60 seconds), MEC-assisted DASH provides a buffer level, which is about twice as large as in regular throughput-based DASH. At the same time, we limit the number of representation changes and keep the video quality at an appropriate level (between representations #7 and #9). It becomes clear that the MEC-assisted DASH approach provides a better context aware streaming algorithm and, thus, improves the overall quality provided to the user.

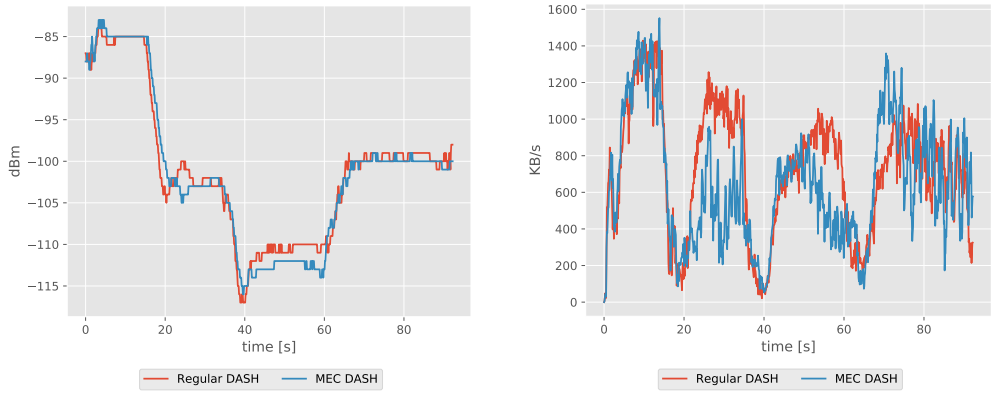
Comparing the averaged signal power value, c.f., Fig. 6.14a, we see that the experiments were performed in an almost identical setup and that there is a small variance in the signal power received by the user. Please also consider Figs. 6.14b and 6.13a. Until segment #10, both algorithms display similar performance, which is reflected in Fig. 6.14b, as comparable average throughput until t=20 seconds. Then, the MEC DASH algorithm provides slightly less throughput from 20 up to 60 seconds, which corresponds to segments between #10 and #30 in Fig. 6.13a.



(a) (b)

FIGURE 6.13: A statistical comparison of our MEC-assisted implementation and the regular rate-based implementation.

We also notice that the throughput-based adaptation immediately provides the user with very high video quality without the establishment of an appropriate buffer level. This is a huge difference in comparison to the buffer-based adaptation (c.f., Sec. 6.1).



(a) (b)

FIGURE 6.14: The averaged values of the signal quality (a) and throughput (b) measurements of each individual adaptation algorithm using rate-based streaming.

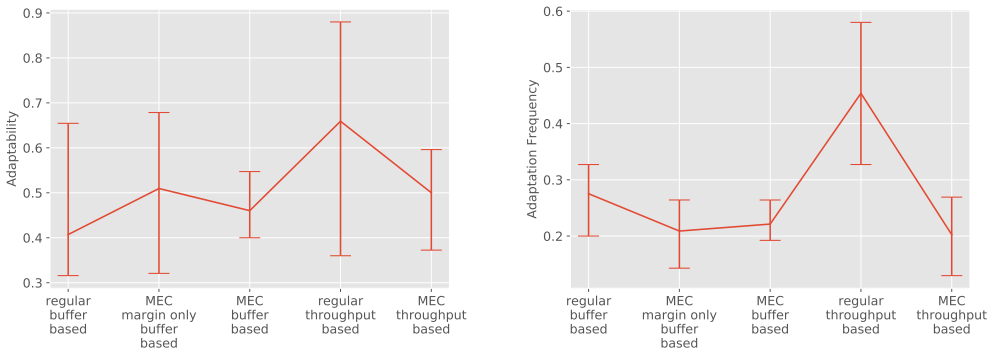
6.3 Discussion

In previous sections (c.f., Secs. 6.1.1– 6.2.2), we showed the results from our measurements and compared regular buffer-/throughput-based adaptation algorithms against our novel MEC-assisted video delivery scheme with respect to the video quality and buffer fill level provided.

Let us now have a closer look at the Adaptability (A) and Adaptation Frequency (AF) of different algorithms (c.f., Sec. 5.2). Figs. 6.15a and 6.15b show the average Adaptability and the average Adaptation Frequency of each measured adaptation algorithm with the min/max deviation.

Again, we notice a very large variance between the individual measurements, i.e., the min/max deviation is very large (c.f., Fig. 6.15). As we limit the available video qualities towards the end-user, the MEC-assisted DASH tends to provide a lower *AF* (c.f., Fig. 6.15b), i.e., fewer changes in quality are done, and at the same time, we also limit the deviation between the individual measurements. This means that using MEC in both buffer- and throughput-based DASH, we can provide a user with a video stream of a

more consistent quality. This is obviously an improvement of the QoE experienced by users. Regarding A, c.f., Fig. 6.15a, we need to distinguish between buffer- and throughput-based adaptation. Compared to regular buffer-based adaptation, in throughput-based adaptation, we can benefit from a faster adaptation to high quality and, thus, load the link more appropriately, which, on average, provides a slightly higher adaptability. As we see, in throughput-based MEC-assisted adaptation, we generally tend to be more cautious, i.e., we do not overload the link, and keep the margin. Thus, we do not tend to use the entire link capacity, which results in a lower A value (c.f., Fig. 6.15a).



(a) (b)

FIGURE 6.15: The five algorithms considered compared in terms of Adaptation Frequency (AF) and Adaptability (A).

Although the variation in individual measurements is high, our algorithms improve the statistical quality of the video stream as demonstrated by different key performance indicators (e.g., segment quality, buffer fill level, adaptation frequency). Our improvements are highly visible in the case of video streaming with throughput-based adaptation, while the video stream displays much better performance in experiments, in which the MEC-assisted adaptation is deployed.

Chapter 7

Conclusions

7.1 Summary

As indicated by Karagkioules et al. [33], the development of a robust algorithm that provides high video qualities under varying conditions experienced in mobile networks is still a major research challenge.

In our work on video delivery in mobile networks, we combine radio network information to improve certain characteristics of mobile video streaming in regular state-of-the-art DASH AVC. In a proof of concept implementation, we developed a MEC platform hosting a video service relying on a server-side adaptation technique, which matches the momentary radio link quality with a desirable video quality. We conducted several real-world experiments in a femto-cell scenario testing the algorithm developed.

In our experiments we observed different behaviors depending on the video adaptation mechanism used at the client. The results showed that on average MEC-assisted DASH improved the overall delivered video quality, no matter which client-side adaptation (e.g., buffer-, throughput-based) was used. We observed a better average video quality and at the same time experienced fewer changes in quality (lower adaptation frequency) and an extended buffer fill-level. Our results, therefore, suggest that a RAN aware video service can, indeed, improve the video quality provided towards the user in mobile networks.

7.2 Future Work

In future research, we would like to evaluate the optimal parameters of our algorithm, i.e., to determine the minimum and maximum thresholds to provide lower and higher video quality respectively and the optimal periodicity of downloading the dynamic MPD file, which may be segment-duration dependent. Moreover, we plan to evaluate different adaptation strategies at the video server restricting the available video qualities.

Furthermore, we plan to include Artificial Intelligence (AI) to detect the traffic pattern/segment duration in traffic carrying multiple video-streams. We also want to use the information on mobility prediction to anticipate the user location and signal quality.

Appendix A

Software Configuration

A.1 Commands to build and run the FlexRAN controller and eNB

To build and run the FlexRAN controller module, we used the following commands:

Build: `./build_flexran_rtc.sh`

Run: `./run_flexran_rtc.sh`

To build and run the eNB, with the co located FlexRAN agent, the commands are as follows:

Build: `./build_oai -a -c -C -eNB -w USRP`

Run: `./lte-softmodem -O enb.band7.tm1.25PRB.usrpb210.conf -S -ulsch-max-errors 100000`

Where `enb.band7.tm1.25PRB.usrpb210.conf` references to the configuration file of the OAI-eNB, c.f., Appendix A.3.

A.2 Configuration Files of the OAI Core Network

LISTING A.1: Configuration file of HSS

```
1 ##################################################################
```

```
2  # Licensed to the OpenAirInterface (OAI) Software Alliance under
       one or more
3  # contributor license agreements.  See the NOTICE file distributed
       with
4  # this work for additional information regarding copyright
       ownership.
5  # The OpenAirInterface Software Alliance licenses this file to You
       under
6  # the Apache License, Version 2.0  (the "License"); you may not
       use this file
7  # except in compliance with the License.
8  # You may obtain a copy of the License at
9  #
10 #      http://www.apache.org/licenses/LICENSE-2.0
11 #
12 # Unless required by applicable law or agreed to in writing,
       software
13 # distributed under the License is distributed on an "AS IS" BASIS,
14 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
       implied.
15 # See the License for the specific language governing permissions
       and
16 # limitations under the License.
17 #---------------------------------------------------------------------
18 # For more information about the OpenAirInterface (OAI) Software
       Alliance:
19 #      contact@openairinterface.org
20 ######################################################################
21 HSS :
22 {
23 ## MySQL mandatory options
24 MYSQL_server = "127.0.0.1";     # HSS S6a bind address
25 MYSQL_user   = "@MYSQL_user@"; # Database server login
26 MYSQL_pass   = "@MYSQL_pass@"; # Database server password
27 MYSQL_db     = "oai_db";        # Your database name
28
29 ## HSS options
30 OPERATOR_key = "1006020f0a478bf6b699f15c062e42b3"; # OP key
       matching your database
31 #OPERATOR_key = "11111111111111111111111111111111"; # OP key
       matching your database
```

```
32
33 RANDOM = "true";                                    # True random
    or only pseudo random (for subscriber vector generation)
34
35 ## Freediameter options
36 FD_conf = "/usr/local/etc/oai/freeDiameter/hss_fd.conf";
37 };
```

LISTING A.2: Configuration file of S/PGW

```
1 ##########################################################################
2 # Licensed to the OpenAirInterface (OAI) Software Alliance under
    one or more
3 # contributor license agreements.  See the NOTICE file distributed
    with
4 # this work for additional information regarding copyright
    ownership.
5 # The OpenAirInterface Software Alliance licenses this file to You
    under
6 # the Apache License, Version 2.0  (the "License"); you may not
    use this file
7 # except in compliance with the License.
8 # You may obtain a copy of the License at
9 #
10 #      http://www.apache.org/licenses/LICENSE-2.0
11 #
12 # Unless required by applicable law or agreed to in writing,
    software
13 # distributed under the License is distributed on an "AS IS" BASIS,
14 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied.
15 # See the License for the specific language governing permissions
    and
16 # limitations under the License.
17 #-------------------------------------------------------------------
18 # For more information about the OpenAirInterface (OAI) Software
    Alliance:
19 #      contact@openairinterface.org
20 ##########################################################################
21 S-GW :
22 {
```

```
23  NETWORK_INTERFACES :
24  {
25      # S-GW binded interface for S11 communication (GTPV2-C),
    if none selected the ITTI message interface is used
26      SGW_INTERFACE_NAME_FOR_S11              = "ens3";
                # STRING, interface name, YOUR NETWORK CONFIG
    HERE
27      SGW_IPV4_ADDRESS_FOR_S11                =
    "130.92.70.164/24";               # STRING, CIDR, YOUR NETWORK
    CONFIG HERE
28
29      # S-GW binded interface for S1-U communication (GTPV1-U)
    can be ethernet interface, virtual ethernet interface, we don't
    advise wireless interfaces
30      SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP    = "ens3";
                # STRING, interface name, YOUR NETWORK CONFIG
    HERE, USE "lo" if S-GW run on eNB host
31      SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP      =
    "130.92.70.164/24";           # STRING, CIDR, YOUR NETWORK
    CONFIG HERE
32      SGW_IPV4_PORT_FOR_S1U_S12_S4_UP         = 2152;
                # INTEGER, port number, PREFER NOT CHANGE UNLESS
    YOU KNOW WHAT YOU ARE DOING
33
34      # S-GW binded interface for S5 or S8 communication, not
    implemented, so leave it to none
35      SGW_INTERFACE_NAME_FOR_S5_S8_UP         = "none";
                # STRING, interface name, DO NOT CHANGE (NOT
    IMPLEMENTED YET)
36      SGW_IPV4_ADDRESS_FOR_S5_S8_UP           = "0.0.0.0/24";
                # STRING, CIDR, DO NOT CHANGE (NOT IMPLEMENTED YET)
37  };
38
39  INTERTASK_INTERFACE :
40  {
41      # max queue size per task
42      ITTI_QUEUE_SIZE             = 2000000;
                # INTEGER
43  };
44
45  LOGGING :
```

```
46      {
47          # OUTPUT choice in { "CONSOLE", "SYSLOG", ‘path to file‘",
    "‘IPv4@‘:‘TCP port num‘"}
48          # ‘path to file‘ must start with ’.’ or ’/’
49          # if TCP stream choice, then you can easily dump the
    traffic on the remote or local host: nc -l ‘TCP port num‘ >
    received.txt
50          OUTPUT             = "CONSOLE";
                    # see 3 lines above
51          #OUTPUT             = "SYSLOG";
                    # see 4 lines above
52          #OUTPUT             = "/tmp/spgw.log";
                    # see 5 lines above
53          #OUTPUT             = "127.0.0.1:5656";
                    # see 6 lines above
54
55          # THREAD_SAFE choice in { "yes", "no" } means use of
    thread safe intermediate buffer then a single thread pick each
    message log one
56          # by one to flush it to the chosen output
57          THREAD_SAFE        = "no";
58
59          # COLOR choice in { "yes", "no" } means use of ANSI
    styling codes or no
60          COLOR              = "yes";
61
62          # Log level choice in { "EMERGENCY", "ALERT", "CRITICAL",
    "ERROR", "WARNING", "NOTICE", "INFO", "DEBUG", "TRACE"}
63          UDP_LOG_LEVEL      = "TRACE";
64          GTPV1U_LOG_LEVEL   = "TRACE";
65          GTPV2C_LOG_LEVEL   = "TRACE";
66          SPGW_APP_LOG_LEVEL = "TRACE";
67          S11_LOG_LEVEL      = "TRACE";
68      };
69  };
70
71  P-GW =
72  {
73      NETWORK_INTERFACES :
74      {
```

```
75      # P-GW binded interface for S5 or S8 communication, not
    implemented, so leave it to none
76      PGW_INTERFACE_NAME_FOR_S5_S8          = "none";
                # STRING, interface name, DO NOT CHANGE (NOT
    IMPLEMENTED YET)
77
78      # P-GW binded interface for SGI (egress/ingress internet
    traffic)
79      PGW_INTERFACE_NAME_FOR_SGI            = "ens3";
                # STRING, YOUR NETWORK CONFIG HERE
80      PGW_MASQUERADE_SGI                    = "yes";
                 # STRING, {"yes", "no"}. YOUR NETWORK CONFIG
    HERE, will do NAT for you if you put "yes".
81      UE_TCP_MSS_CLAMPING                   = "no";
                # STRING, {"yes", "no"}.
82  };
83
84  # Pool of UE assigned IP addresses
85  # Do not make IP pools overlap
86  # first IPv4 address X.Y.Z.1 is reserved for GTP network
    device on SPGW
87  # Normally no more than 16 pools allowed, but since recent GTP
    kernel module use, only one pool allowed (TODO).
88  IP_ADDRESS_POOL :
89  {
90      IPV4_LIST = (
91                    "172.16.0.0/12"
                # STRING, CIDR, YOUR NETWORK CONFIG HERE.
92                  );
93  };
94
95  # DNS address communicated to UEs
96  DEFAULT_DNS_IPV4_ADDRESS     = "130.92.9.52";
                  # YOUR NETWORK CONFIG HERE
97  DEFAULT_DNS_SEC_IPV4_ADDRESS = "130.92.9.53";
                  # YOUR NETWORK CONFIG HERE
98
99  # Non standard feature, normally should be set to "no", but
    you may need to set to yes for UE that do not explicitly
    request a PDN address through NAS signalling
```

```
100     FORCE_PUSH_PROTOCOL_CONFIGURATION_OPTIONS = "yes";
                    # STRING, {"yes", "no"}.
101     UE_MTU                                  = 1428
                    # INTEGER
102 };
```

LISTING A.3: Configuration file of MME

```
1  ################################################################################
2  # Licensed to the OpenAirInterface (OAI) Software Alliance under
       one or more
3  # contributor license agreements.  See the NOTICE file distributed
       with
4  # this work for additional information regarding copyright
       ownership.
5  # The OpenAirInterface Software Alliance licenses this file to You
       under
6  # the Apache License, Version 2.0  (the "License"); you may not
       use this file
7  # except in compliance with the License.
8  # You may obtain a copy of the License at
9  #
10 #      http://www.apache.org/licenses/LICENSE-2.0
11 #
12 # Unless required by applicable law or agreed to in writing,
       software
13 # distributed under the License is distributed on an "AS IS" BASIS,
14 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
       implied.
15 # See the License for the specific language governing permissions
       and
16 # limitations under the License.
17 #-------------------------------------------------------------------
18 # For more information about the OpenAirInterface (OAI) Software
       Alliance:
19 #      contact@openairinterface.org
20 ################################################################################
21
22 MME :
23 {
```

```
24  REALM                                   = "openair4G.eur";
                # YOUR REALM HERE
25  PID_DIRECTORY                           = "/var/run";
26  # Define the limits of the system in terms of served eNB and
    served UE.
27  # When the limits will be reached, overload procedure will
    take place.
28  MAXENB                                  = 10;
                 # power of 2
29  MAXUE                                   = 10;
                # power of 2
30  RELATIVE_CAPACITY                       = 10;
31
32  EMERGENCY_ATTACH_SUPPORTED                  = "no";
33  UNAUTHENTICATED_IMSI_SUPPORTED              = "no";
34
35  # EPS network feature support
36  EPS_NETWORK_FEATURE_SUPPORT_IMS_VOICE_OVER_PS_SESSION_IN_S1
      = "no";    # DO NOT CHANGE
37
    EPS_NETWORK_FEATURE_SUPPORT_EMERGENCY_BEARER_SERVICES_IN_S1_MODE
    = "no";    # DO NOT CHANGE
38  EPS_NETWORK_FEATURE_SUPPORT_LOCATION_SERVICES_VIA_EPC
      = "no";    # DO NOT CHANGE
39  EPS_NETWORK_FEATURE_SUPPORT_EXTENDED_SERVICE_REQUEST
      = "no";    # DO NOT CHANGE
40
41  # Display statistics about whole system (expressed in seconds)
42  MME_STATISTIC_TIMER                     = 10;
43
44  IP_CAPABILITY = "IPV4V6";
                # UNUSED, TODO
45
46
47  INTERTASK_INTERFACE :
48  {
49      # max queue size per task
50      ITTI_QUEUE_SIZE             = 2000000;
51  };
52
53  S6A :
```

```
54  {
55      S6A_CONF                    =
    "/usr/local/etc/oai/freeDiameter/mme_fd.conf"; # YOUR MME
    freeDiameter config file path
56      HSS_HOSTNAME                = "oai-hss-0";
                    # THE HSS HOSTNAME
57  };
58
59  # ------- SCTP definitions
60  SCTP :
61  {
62      # Number of streams to use in input/output
63      SCTP_INSTREAMS  = 8;
64      SCTP_OUTSTREAMS = 8;
65  };
66
67  # ------- S1AP definitions
68  S1AP :
69  {
70      # outcome drop timer value (seconds)
71      S1AP_OUTCOME_TIMER = 10;
72  };
73
74  # ------- MME served GUMMEIs
75  # MME code DEFAULT  size = 8 bits
76  # MME GROUP ID size = 16 bits
77  GUMMEI_LIST = (
78       {MCC="208"; MNC="95"; MME_GID="4" ; MME_CODE="1"; }
              # YOUR GUMMEI CONFIG HERE
79  );
80
81  # ------- MME served TAIs
82  # TA (mcc.mnc:tracking area code) DEFAULT = 208.34:1
83  # max values = 999.999:65535
84  # maximum of 16 TAIs, comma separated
85  # !!! Actually use only one PLMN
86  TAI_LIST = (
87       {MCC="208"; MNC="95";  TAC="1"; }
          # YOUR TAI CONFIG HERE
88  );
89
```

```
90
91  NAS :
92  {
93      # 3GPP TS 33.401 section 7.2.4.3 Procedures for NAS
    algorithm selection
94      # decreasing preference goes from left to right
95      ORDERED_SUPPORTED_INTEGRITY_ALGORITHM_LIST = [ "EIA2" ,
    "EIA1" , "EIA0" ];
96      ORDERED_SUPPORTED_CIPHERING_ALGORITHM_LIST = [ "EEA0" ,
    "EEA1" , "EEA2" ];
97
98      # EMM TIMERS
99      # T3402 start:
100     # At attach failure and the attempt counter is equal to 5.
101     # At tracking area updating failure and the attempt
    counter is equal to 5.
102     # T3402 stop:
103     # ATTACH REQUEST sent, TRACKING AREA REQUEST sent.
104     # On expiry:
105     # Initiation of the attach procedure, if still required or
    TAU procedure
106     # attached for emergency bearer services.
107     T3402                                   =  1
                # in minutes (default is 12 minutes)
108     # T3412 start:
109     # In EMM-REGISTERED, when EMM-CONNECTED mode is left.
110     # T3412 stop:
111     # When entering state EMM-DEREGISTERED or when entering
    EMM-CONNECTED mode.
112     # On expiry:
113     # Initiation of the periodic TAU procedure if the UE is
    not attached for
114     # emergency bearer services. Implicit detach from network
    if the UE is
115     # attached for emergency bearer services.
116     T3412                                   =  54
                # in minutes (default is 54 minutes, network
    dependent)
117     # T3422 start: DETACH REQUEST sent
118     # T3422 stop: DETACH ACCEPT received
```

```
119         # ON THE 1st, 2nd, 3rd, 4th EXPIRY: Retransmission of
    DETACH REQUEST
120         T3422                                   =  6
                    # in seconds (default is 6s)
121
122         # T3450 start:
123         # ATTACH ACCEPT sent, TRACKING AREA UPDATE ACCEPT sent
    with GUTI, TRACKING AREA UPDATE ACCEPT sent with TMSI,
124         # GUTI REALLOCATION COMMAND sent
125         # T3450 stop:
126         # ATTACH COMPLETE received, TRACKING AREA UPDATE COMPLETE
    received, GUTI REALLOCATION COMPLETE received
127         # ON THE 1st, 2nd, 3rd, 4th EXPIRY: Retransmission of the
    same message type
128         T3450                                   =  6
                    # in seconds (default is 6s)
129
130         # T3460 start: AUTHENTICATION REQUEST sent, SECURITY MODE
    COMMAND sent
131         # T3460 stop:
132         # AUTHENTICATION RESPONSE received, AUTHENTICATION FAILURE
    received,
133         # SECURITY MODE COMPLETE received, SECURITY MODE REJECT
    received
134         # ON THE 1st, 2nd, 3rd, 4th EXPIRY: Retransmission of the
    same message type
135         T3460                                   =  6
                    # in seconds (default is 6s)
136
137         # T3470 start: IDENTITY REQUEST sent
138         # T3470 stop: IDENTITY RESPONSE received
139         # ON THE 1st, 2nd, 3rd, 4th EXPIRY: Retransmission of
    IDENTITY REQUEST
140         T3470                                   =  6
                    # in seconds (default is 6s)
141
142         # ESM TIMERS
143         T3485                                   =  8
                    # UNUSED in seconds (default is 8s)
144         T3486                                   =  8
                    # UNUSED in seconds (default is 8s)
```

```
145         T3489                                   =  4
                    # UNUSED in seconds (default is 4s)
146         T3495                                   =  8
                    # UNUSED in seconds (default is 8s)
147     };
148
149     NETWORK_INTERFACES :
150     {
151         # MME binded interface for S1-C or S1-MME  communication
    (S1AP), can be ethernet interface, virtual ethernet interface,
    we don't advise wireless interfaces
152         MME_INTERFACE_NAME_FOR_S1_MME         = "ens3";
                    # YOUR NETWORK CONFIG HERE
153         MME_IPV4_ADDRESS_FOR_S1_MME           =
    "192.168.0.116/24";             # YOUR NETWORK CONFIG HERE
154
155         # MME binded interface for S11 communication (GTPV2-C)
156         MME_INTERFACE_NAME_FOR_S11_MME        = "ens3";
                      # YOUR NETWORK CONFIG HERE
157         MME_IPV4_ADDRESS_FOR_S11_MME          =
    "192.168.0.116/24";                 # YOUR NETWORK CONFIG HERE
158         MME_PORT_FOR_S11_MME                  = 2123;
                    # YOUR NETWORK CONFIG HERE
159     };
160
161     LOGGING :
162     {
163         # OUTPUT choice in { "CONSOLE"}
164         # `path to file` must start with '.' or '/'
165         # if TCP stream choice, then you can easily dump the
    traffic on the remote or local host: nc -l `TCP port num` >
    received.txt
166         OUTPUT            = "CONSOLE";
167         #OUTPUT            = "CONSOLE";
168         #OUTPUT            = "CONSOLE";
169         #OUTPUT            = "CONSOLE";
170
171         # THREAD_SAFE choice in { "yes", "no" } means use of
    thread safe intermediate buffer then a single thread pick each
    message log one
172         # by one to flush it to the chosen output
```

```
173         THREAD_SAFE       = "yes";
174
175         # COLOR choice in { "yes", "no" } means use of ANSI
    styling codes or no
176         COLOR             = "yes";
177
178         # Log level choice in { "EMERGENCY", "ALERT", "CRITICAL",
    "ERROR", "WARNING", "NOTICE", "INFO", "DEBUG", "TRACE"}
179         SCTP_LOG_LEVEL    = "TRACE";
180         S11_LOG_LEVEL     = "TRACE";
181         GTPV2C_LOG_LEVEL  = "TRACE";
182         UDP_LOG_LEVEL     = "TRACE";
183         S1AP_LOG_LEVEL    = "TRACE";
184         NAS_LOG_LEVEL     = "TRACE";
185         MME_APP_LOG_LEVEL = "TRACE";
186         S6A_LOG_LEVEL     = "TRACE";
187         UTIL_LOG_LEVEL    = "TRACE";
188         MSC_LOG_LEVEL     = "ERROR";
189         ITTI_LOG_LEVEL    = "ERROR";
190         MME_SCENARIO_PLAYER_LOG_LEVEL = "TRACE";
191
192         # ASN1 VERBOSITY: none, info, annoying
193         # for S1AP protocol
194         ASN1_VERBOSITY    = "none";
195     };
196     TESTING :
197     {
198         # file should be copied here from source tree by following
    command: run_mme --install-mme-files ...
199         SCENARIO_FILE =
    "/usr/local/share/oai/test/mme/no_regression.xml";
200     };
201 };
202
203 S-GW :
204 {
205     # S-GW binded interface for S11 communication (GTPV2-C), if
    none selected the ITTI message interface is used
206     SGW_IPV4_ADDRESS_FOR_S11                = "130.92.70.164/24";
                        # YOUR NETWORK CONFIG HERE
207
```

```
208 };
```

A.3 Configuration Files of the eNB

LISTING A.4: Configuration of the eNB, enb.band7.tm1.25PRB.usrpb210.conf

```
1  Active_eNBs = ( "eNB_Eurecom_LTEBox");
2  # Asn1_verbosity, choice in: none, info, annoying
3  Asn1_verbosity = "none";
4
5  eNBs =
6  (
7   {
8      ////////// Identification parameters:
9      eNB_ID    =  0xe00;
10
11     cell_type =  "CELL_MACRO_ENB";
12
13     eNB_name  =  "eNB_Eurecom_LTEBox";
14
15     // Tracking area code, 0x0000 and 0xfffe are reserved values
16     tracking_area_code  =  "1";
17
18     mobile_country_code =  "208";
19
20     mobile_network_code =  "95";
21
22        ////////// Physical parameters:
23
24     component_carriers = (
25       {
26         node_function                                   =
     "eNodeB_3GPP";
27    node_timing                                         =
     "synch_to_ext_device";
28    node_synch_ref                                      = 0;
29         frame_type                      = "FDD";
30         tdd_config                      = 3;
```

```
31        tdd_config_s                            = 0;
32        prefix_type                             = "NORMAL";
33        eutra_band                              = 7;
34        downlink_frequency                      = 2660000000L;
35        uplink_frequency_offset                 = -120000000;
36        Nid_cell                    = 0;
37        N_RB_DL                                 = 25;
38        Nid_cell_mbsfn                          = 0;
39        nb_antenna_ports                      = 1;
40        nb_antennas_tx                          = 1;
41        nb_antennas_rx                          = 1;
42        tx_gain                                                 = 90;
43        rx_gain                                                 = 125;
44        prach_root                              = 0;
45        prach_config_index                      = 0;
46        prach_high_speed                        = "DISABLE";
47        prach_zero_correlation                  = 1;
48        prach_freq_offset                       = 2;
49        pucch_delta_shift                       = 1;
50        pucch_nRB_CQI                           = 1;
51        pucch_nCS_AN                            = 0;
52        pucch_n1_AN                             = 32;
53        pdsch_referenceSignalPower                 = -24;
54        pdsch_p_b                                  = 0;
55        pusch_n_SB                                 = 1;
56        pusch_enable64QAM                          = "DISABLE";
57        pusch_hoppingMode                                       =
    "interSubFrame";
58        pusch_hoppingOffset                                     = 0;
59        pusch_groupHoppingEnabled                  = "ENABLE";
60        pusch_groupAssignment                      = 0;
61        pusch_sequenceHoppingEnabled                     = "DISABLE";
62        pusch_nDMRS1                                            = 1;
63        phich_duration                                          =
    "NORMAL";
64        phich_resource                                          =
    "ONESIXTH";
65        srs_enable                                              =
    "DISABLE";
66        /*  srs_BandwidthConfig                                     =;
67        srs_SubframeConfig                                      =;
```

```
68          srs_ackNackST                                   =;
69          srs_MaxUpPts                                    =;*/
70
71          pusch_p0_Nominal                                = -96;
72          pusch_alpha                                     = "AL1";
73          pucch_p0_Nominal                                = -104;
74          msg3_delta_Preamble                             = 6;
75          pucch_deltaF_Format1                            =
    "deltaF2";
76          pucch_deltaF_Format1b                           =
    "deltaF3";
77          pucch_deltaF_Format2                            =
    "deltaF0";
78          pucch_deltaF_Format2a                           =
    "deltaF0";
79          pucch_deltaF_Format2b                = "deltaF0";
80
81          rach_numberOfRA_Preambles                       = 64;
82          rach_preamblesGroupAConfig                      =
    "DISABLE";
83          /*
84          rach_sizeOfRA_PreamblesGroupA                   = ;
85          rach_messageSizeGroupA                          = ;
86          rach_messagePowerOffsetGroupB                   = ;
87          */
88          rach_powerRampingStep                           = 4;
89          rach_preambleInitialReceivedTargetPower         = -104;
90          rach_preambleTransMax                           = 10;
91          rach_raResponseWindowSize                       = 10;
92          rach_macContentionResolutionTimer               = 48;
93          rach_maxHARQ_Msg3Tx                             = 4;
94
95          pcch_default_PagingCycle                        = 128;
96          pcch_nB                                         =
    "oneT";
97          bcch_modificationPeriodCoeff              = 2;
98          ue_TimersAndConstants_t300             = 1000;
99          ue_TimersAndConstants_t301             = 1000;
100         ue_TimersAndConstants_t310             = 1000;
101         ue_TimersAndConstants_t311             = 10000;
102         ue_TimersAndConstants_n310             = 20;
```

```
103         ue_TimersAndConstants_n311              = 1;
104
105     ue_TransmissionMode                     = 1;
106        }
107     );
108
109     srb1_parameters :
110     {
111         # timer_poll_retransmit = (ms) [5, 10, 15, 20,... 250,
        300, 350, ... 500]
112         timer_poll_retransmit    = 80;
113
114         # timer_reordering = (ms) [0,5, ... 100, 110, 120, ...
        ,200]
115         timer_reordering         = 35;
116
117         # timer_reordering = (ms) [0,5, ... 250, 300, 350, ...
        ,500]
118         timer_status_prohibit    = 0;
119
120         # poll_pdu = [4, 8, 16, 32 , 64, 128, 256,
        infinity(>10000)]
121         poll_pdu                 =  4;
122
123         # poll_byte = (kB) [25, 50, 75, 100, 125, 250, 375, 500,
        750, 1000, 1250, 1500, 2000, 3000, infinity(>10000)]
124         poll_byte                =  99999;
125
126         # max_retx_threshold = [1, 2, 3, 4 , 6, 8, 16, 32]
127         max_retx_threshold       =  4;
128     }
129
130     # ------- SCTP definitions
131     SCTP :
132     {
133         # Number of streams to use in input/output
134         SCTP_INSTREAMS  = 2;
135         SCTP_OUTSTREAMS = 2;
136     };
137
138     ////////// MME parameters:
```

```
139     mme_ip_address      = (
140                                { ipv4       = "130.92.70.163";
141                                  ipv6       = "192:168:30::17";
142                                  active     = "yes";
143                                  preference = "ipv4"; }
144                              );
145     NETWORK_INTERFACES :
146     {
147         ENB_INTERFACE_NAME_FOR_S1_MME            = "em1";
148         ENB_IPV4_ADDRESS_FOR_S1_MME              =
    "130.92.182.38/24";
149
150         ENB_INTERFACE_NAME_FOR_S1U               = "em1";
151         ENB_IPV4_ADDRESS_FOR_S1U                 =
    "130.92.182.38/24";
152         ENB_PORT_FOR_S1U                         = 2152; # Spec
    2152
153     };
154
155     NETWORK_CONTROLLER :
156     {
157         FLEXRAN_AGENT_INTERFACE_NAME             = "em1";
158         FLEXRAN_AGENT_IPV4_ADDRESS               = "130.92.70.169";
159         FLEXRAN_AGENT_PORT                       = 2210;
160         FLEXRAN_AGENT_CACHE                      =
    "/mnt/oai_agent_cache";
161     };
162
163
164     log_config :
165     {
166       global_log_level                      ="info";
167       global_log_verbosity                  ="medium";
168       hw_log_level                          ="info";
169       hw_log_verbosity                      ="medium";
170       phy_log_level                         ="info";
171       phy_log_verbosity                     ="medium";
172       mac_log_level                         ="info";
173       mac_log_verbosity                     ="high";
174       rlc_log_level                         ="info";
175       rlc_log_verbosity                     ="medium";
```

```
176         pdcp_log_level                      ="info";
177         pdcp_log_verbosity                  ="medium";
178         rrc_log_level                       ="info";
179         rrc_log_verbosity                   ="medium";
180       };
181     }
182 );
```

Appendix B

Software Modifications and Implementation of the Video Service

B.1 Modification to the OAI-S/PGW

Since the S/PGW is running running on OpenStack and OpenStack is using floating IP addresses, to which the S/PGW cannot bind. So we also had to address a change in the S/PGW, so that, it can bind to the local addresses.

LISTING B.1: Modified to bind to IP address of local computer instead of floating IPv4 address in /src/udp/udp_primitives_server.c of OAI CN

```
1    addr.sin_port = htons (port);
2  - addr.sin_addr.s_addr = inet_addr (address);
3  + // addr.sin_addr.s_addr = inet_addr (address);
4  +
5  + addr.sin_addr.s_addr = inet_addr ("0.0.0.0");
```

B.2 Extensions at the OAI-eNB and FlexRAN

LISTING B.2: The new message implemented in stats_common.proto at the eNB and FlexRAN

```
1 +message flex_dl_video_stats {
2 +  optional int32 rsrp_value = 1;
```

```
3 +  optional int32 rsrq_value = 2;
4 +  optional uint32 cqi_value = 3;
5 +  optional uint64 num_total_bytes = 4;
6 +  optional uint64 num_acks = 5;
7 +  optional uint64 num_nacks = 6;
8 +  optional uint32 mcs_value = 7;
9 +  optional uint64 timestamp = 8;
10 }
```

LISTING B.3: The new message needs to be included in the request message in /stats_messages.proto at the eNB and FlexRAN

```
1    optional flex_rrc_measurements rrc_measurements = 10;
2 +  optional flex_dl_video_stats video_stats = 11;
3
4 ...
5
6        FLUST_UL_CQI = 64;
7 +      FLUST_DL_VIDEO_STATS = 128;
```

B.3 Modifications to FlexRAN

LISTING B.4: Request the new stats from the eNB in FlexRAN, src/app/stats_manager.cc

```
1        ue_flags |= protocol::FLUST_UL_CQI;
2 -      // ue_flags |= protocol::FLUST_RRC_MEASUREMENTS;
3 +      ue_flags |= protocol::FLUST_RRC_MEASUREMENTS;
4 +      ue_flags |= protocol::FLUST_DL_VIDEO_STATS;
```

LISTING B.5: Add new route to FlexRAN only providing our statistics, src/north_api/stats_manager_calls.cc

```
1        response.send(Pistache::Http::Code::Ok, resp);
2 +  } else if (stats_type.compare("video_stats") == 0) {
3 +      resp = stats_app->mac_config_to_string();
4 +      resp = resp.substr(resp.find("video_stats"), resp.find("Harq
     status")-resp.find("video_stats"));
5 +      response.send(Pistache::Http::Code::Ok, resp);
```

B.4 Modifications to the OAI-eNB

LISTING B.6: Fill the new message with appropriate information openair2/ENB_APP/CONTROL_MODULES/MAC/flexran_agent_mac.c

```
+#include <time.h>
+#include "RRC/LITE/MESSAGES/asn1_msg.h"
+#include "../RRC/flexran_agent_rrc.h"
...

                ue_report[i]->rrc_measurements =
    rrc_measurement_report;
                }

+           if (report_config->ue_report_type[i].ue_report_flags &
    PROTOCOL__FLEX_UE_STATS_TYPE__FLUST_DL_VIDEO_STATS){
 +       Protocol__FlexDlVideoStats * video_stats;
 +       video_stats = malloc(sizeof(Protocol__FlexDlVideoStats));
 +       protocol__flex_dl_video_stats__init(video_stats);
 +       if(rrc_stats[0] != 0){
 +          video_stats->rsrp_value = rrc_stats[0];
 +          video_stats->has_rsrp_value = 1;
 +       }
 +       if(rrc_stats[1] != 0){
 +          video_stats->rsrq_value = rrc_stats[1];
 +          video_stats->has_rsrq_value = 1;
 +       }
 +       video_stats->cqi_value = flexran_get_ue_wcqi (enb_id, i);
 +       video_stats->has_cqi_value = 1;
 +       video_stats->num_total_bytes =
    flexran_get_ue_num_total_bytes(enb_id, i);
 +       video_stats->has_num_total_bytes = 1;
 +       video_stats->num_acks = flexran_get_num_acks();
 +       video_stats->has_num_acks = 1;
 +       video_stats->num_nacks = flexran_get_num_nacks();
 +       video_stats->has_num_nacks = 1;
 +       video_stats->mcs_value =
    flexran_get_ue_dlsch_mcs1(enb_id, i);
 +       video_stats->has_mcs_value = 1;
 +       video_stats->timestamp = (int)time(NULL);
```

```
+       video_stats->has_timestamp = 1;
+
+       ue_report[i]->video_stats = video_stats;
+
     }
```

LISTING B.7: Extensions to the RAN API openair2/ENB_APP/flexran_agent_ran_api.c

```

 #include "flexran_agent_ran_api.h"
+#include "flexran_agent_extern.h"
+uint64_t num_acks = 0;
+uint64_t num_nacks = 0;

...
   return ((UE_list_t *)
    enb_ue[mod_id])->UE_template[UE_PCCID(mod_id,ue_id)][ue_id].phr_info;
 }

+uint8_t flexran_get_ue_dlsch_mcs1(mid_t mod_id, mid_t ue_id){
 + LTE_eNB_UE_stats     *eNB_UE_stats     = NULL;
 +  eNB_UE_stats = mac_xface->get_eNB_UE_stats(mod_id, 0,
    UE_RNTI(mod_id, ue_id));
 +  return eNB_UE_stats->dlsch_mcs1;
 +}
 +
 +uint8_t flexran_get_ue_dlsch_mcs2(mid_t mod_id, mid_t ue_id){
 +        LTE_eNB_UE_stats     *eNB_UE_stats     = NULL;
 +  eNB_UE_stats = mac_xface->get_eNB_UE_stats(mod_id, 0,
    UE_RNTI(mod_id, ue_id));
 +  return eNB_UE_stats->dlsch_mcs2;

+uint64_t flexran_get_ue_num_total_bytes(mid_t mod_id, mid_t
    ue_id){
 +  for(int k = 0; k < 8; k++){
 +                if(harq_pid_round[ue_id][k] == 0){
 +                        num_acks += 1;
 +                }else{
 +                        num_nacks += 1;
 +                }
```

```
29  +  }
30  +  return ((UE_list_t *) enb_ue[mod_id])->
31        eNB_UE_stats[UE_PCCID(mod_id,ue_id)][ue_id].total_pdu_bytes;
32 +}
33
34 +uint64_t flexran_get_num_acks(){
35  + return num_acks;
36 +}
37 +uint64_t flexran_get_num_nacks(){
38  + return num_nacks;
39 +}
```

LISTING B.8: Extensions to the /ENB_APP/CONTROL_MODULES/RRC/flexran_agent_rrc.c, store the measured value in shared global variable

```
1 #include "log.h"
2
3 +extern float rrc_stats[2];
4
5  ...
6
7 rrc_measurements->pcell_rsrq =
      (measResults->measResultPCell.rsrqResult)/2 - 20;
8       rrc_measurements->has_pcell_rsrq = 1 ;
9  +    rrc_stats[0] = rrc_measurements->pcell_rsrp;
10 +    rrc_stats[1] = rrc_measurements->pcell_rsrq;
```

B.5 Video Service Implementation

The video service presents a Python script, that is running during the video delivery on the video server and adapts to the current network status.

LISTING B.9: Flexran.py, a script reading and storing the stats obtained from FlexRAN

```
1 import numpy as np
2 import urllib2
3
4 class Flexran(object):
```

```
5
6   def __init__(self, source=None):
7       self.source = source
8       self.cqi = []
9       self.rsrp = []
10      self.rsrq = []
11      self.mcs = []
12      self.num_bytes = []
13      self.num_acks = []
14      self.num_nacks = []
15      self.timestamps = []
16      self.throughputs = []
17
18  @property
19  def cqi(self):
20      return self.__cqi
21
22  @cqi.setter
23  def cqi(self, cqi):
24      self.__cqi = cqi
25
26  @property
27  def rsrp(self):
28      return self.__rsrp
29
30  @rsrp.setter
31  def rsrp(self, rsrp):
32      self.__rsrp = rsrp
33
34  @property
35  def rsrq(self):
36      return self.__rsrq
37
38  @rsrq.setter
39  def rsrq(self, rsrq):
40      self.__rsrq = rsrq
41
42  @property
43  def mcs(self):
44      return self.__mcs
45
```

```
46    @mcs.setter
47    def mcs(self, mcs):
48        self.__mcs = mcs
49
50    @property
51    def num_bytes(self):
52        return self.__num_bytes
53
54    @num_bytes.setter
55    def num_bytes(self, num_bytes):
56        self.__num_bytes = num_bytes
57
58    @property
59    def num_acks(self):
60        return self.__num_acks
61
62    @num_acks.setter
63    def num_acks(self, num_acks):
64        self.__num_acks = num_acks
65
66    @property
67    def num_nacks(self):
68        return self.__num_nacks
69
70    @num_nacks.setter
71    def num_nacks(self, num_nacks):
72        self.__num_nacks = num_nacks
73
74    @property
75    def timestamps(self):
76        return self.__timestamps
77
78    @timestamps.setter
79    def timestamps(self, timestamps):
80        self.__timestamps = timestamps
81
82    @property
83    def throughputs(self):
84        return self.__throughputs
85
86    @throughputs.setter
```

```
87    def throughputs(self, throughputs):
88       self.__throughputs = throughputs
89
90    @property
91    def source(self):
92       return self.__source
93
94    @source.setter
95    def source(self, source):
96       self.__source = source
97
98    def update(self):
99       try:
100         # Connect and read the data from FlexRAN
101         data = urllib2.urlopen(self.source).read()
102         stats = Flexran.parse(data)
103
104         self.cqi.append(stats[0])
105         self.rsrp.append(stats[1])
106         self.rsrq.append(stats[2])
107         self.mcs.append(stats[3])
108         self.num_bytes.append(stats[4])
109         self.num_acks.append(stats[5])
110         self.num_nacks.append(stats[6])
111         self.timestamps.append(stats[7])
112
113
114         if(len(self.timestamps) >= 2):
115            # Calculate differences
116            num_bytes_sent = self.num_bytes[-1] -
      self.num_bytes[-2]
117            num_acks_ = self.num_acks[-1] - self.num_acks[-2]
118            num_nacks_ = self.num_nacks[-1] - self.num_nacks[-2]
119            time = self.timestamps[-1] - self.timestamps[-2]
120
121            # Convert ms to s
122            if(self.timestamps[-1] > 10**10):
123               time = time/1000.0
124
125            # If nothing is sent we would have
126            # division by 0, so set it to 1
```

```
127             if(num_acks_ == 0):
128                 num_acks_ = 1
129 
130             # Set first throughput to 0, as we
131             # can only calculate the throughput
132             # if we have two values
133             if(len(self.timestamps) == 2):
134                 self.throughputs.append(0)
135 
136             # Calculate the throughput
137             self.throughputs.append(1. * num_bytes_sent *
    num_acks_ / (num_acks_+num_nacks_) / time)
138 
139         except urllib2.HTTPError as e:
140             raise Exception('Some flexran error, check the flexran
    service: ' + str(type(e)) + ' and message: ' + e.message)
141 
142         except Exception as e:
143             raise Exception('Some other issue: ' + str(type(e)) + '
    with message ' + e.message)
144 
145     # Parse the JSON data
146     @staticmethod
147     def parse(string):
148         args = string.split("\n")
149         args = [arg for arg in args if ":" in arg]
150         cqi = rsrp = rsrq = mcs = num_bytes = num_acks = num_nacks =
    timestamp = None
151 
152         for arg in args:
153             if "cqi" in arg:
154                 cqi = int(arg.split(":")[1].strip())
155             elif "rsrp" in arg:
156                 rsrp = int(arg.split(":")[1].strip())
157             elif "rsrq" in arg:
158                 rsrq = int(arg.split(":")[1].strip())
159             elif "mcs" in arg:
160                 mcs = int(arg.split(":")[1].strip())
161             elif "total_bytes" in arg:
162                 num_bytes = int(arg.split(":")[1].strip())
163             elif "num_acks" in arg:
```

```
164             num_acks = int(arg.split(":")[1].strip())
165         elif "num_nacks" in arg:
166             num_nacks = int(arg.split(":")[1].strip())
167         elif "timestamp" in arg:
168             timestamp = int(arg.split(":")[1].strip())
169 
170     return cqi, rsrp, rsrq, mcs, num_bytes, num_acks, num_nacks,
    timestamp
```

LISTING B.10: main.py, a script running on the video server, updating the MPD file

```
1 from argparse import ArgumentParser
2 from lxml import etree
3 from Flexran import Flexran
4 import time, sys, traceback, copy
5 import scipy.fftpack
6 import numpy as np
7 
8 def main():
9 
10    # Parsing the following arguments
11    parser = ArgumentParser()
12    parser.add_argument("-f", "--flexran", dest="flexran",
      help="specifiy the webpage for flexran stats (url)",
      metavar="FLEXRAN", required="TRUE")
13    parser.add_argument("-m", "--mpd-file", dest="mpd_file",
      help="specifiy the MPD source file (path)", metavar="MPD_IN",
      required="TRUE")
14    parser.add_argument("-s", "--save-file", dest="save_file",
      help="specifiy the MPD save file (name of the manifest)",
      metavar="MPD_OUT", required="TRUE")
15    parser.add_argument("-i", "--interval", dest="interval",
      help="specifiy the interval at which we update the MPD file (in
      sec)", type=float)
16 
17    args = parser.parse_args()
18    flexran_page = args.flexran
19    mpd_input = args.mpd_file
20    mpd_output = args.save_file
21    interval = args.interval
```

```
22    available_representations = None
23
24    # Instantiate a FlexRAN agent, that subscribes to
25    # the source / RNIS information page of the UE
26    flexran_agent = Flexran(source=flexran_page)
27
28    # If no interval is given use 1 second
29    if(interval == None):
30       interval = 1
31
32    # fetch the first stats
33    flexran_agent.update()
34    counter = 1
35
36    # experiment duration 90[s]
37    duration = 90
38
39    # enter the main loop
40    try:
41       # as long as we the experiment is running
42       while True and counter < duration/interval:
43          time.sleep(interval)
44          flexran_agent.update()
45          counter += 1
46          adapatation_algorithm(mpd_input, mpd_output,
      flexran_agent, interval)
47
48       sys.exit(0)
49
50    # exit by Ctrl + C
51    except KeyboardInterrupt:
52       print("Video service was terminated \n")
53       sys.exit(0)
54
55
56    # exit by exception
57    except Exception:
58       traceback.print_exc(file=sys.stdout)
59       sys.exit(1)
60
```

```
61 def adapatation_algorithm(mpd_input, mpd_output, flexran,
       interval):
62
63     # Get relevant information about UE <-> eNB channel
64     throughput = flexran.throughputs
65     rsrp = flexran.rsrp
66     rsrq = flexran.rsrq
67
68     # t_s defines the segment size of the DASH video
69     t_s = 2
70
71     # Every time a segment is sent over the network
72     # we do adapt the available representations using FFT
73     if(len(throughput)%(t_s/interval) == 0):
74
75         # Use FFT over the last recorded stats
76         yf = np.fft.rfft(throughput[-1*int(t_s/interval):])
77         yf = np.absolute(yf)
78         xf = np.fft.rfftfreq(int(t_s/interval),interval)
79
80         # Use the relation between the peak at 0 Hz and 1/t_s Hz
81         # where t_s denotes the segment size of the DASH video
82         if(yf[1]/yf[0] >= 0.9):
83             print("Security margin too big")
84             remove_lowest_representation(mpd_input, mpd_output)
85             remove_lowest_representation(mpd_input, mpd_output)
86         elif(yf[1]/yf[0] >= 0.4):
87             print("Security margin too big")
88             remove_lowest_representation(mpd_input, mpd_output)
89         elif(yf[1]/yf[0] <= 0.04):
90             print("Security margin too small")
91             remove_highest_representation(mpd_input, mpd_output)
92
93     # Restrict available representations if the RSRP value dropped
       drastically
94     # in the last second
95     if(len(rsrp) >= 1/interval):
96         if(rsrp[-1]-rsrp[-1*int(1/interval)]  <= -5):
97             print("Anticipated drop in radio link quality")
98             remove_highest_representation(mpd_input, mpd_output)
99
```

```
100
101 # Removes the current highest representation in the MPD Output file
102 # and replaces it with the next lower representation available from
103 # the MPD Input file.
104 def remove_highest_representation(mpd_input, mpd_output):
105
106     tree = etree.parse(mpd_output)
107     representations = tree.xpath("//*[local-name() =
        'Representation']")
108
109     # Get the IDs of the currently available representations
110     avail_idxs = list(map(int, map(lambda rep: rep.get("id"),
        representations)))
111     # Current max. representation id
112     current_max_id = max(avail_idxs)
113     # Index of ALL the current max. representation ids
114     current_max_idx = np.where(np.array(avail_idxs) ==
        max(avail_idxs))
115
116     # Parse the original MPD file
117     tree2 = etree.parse(mpd_input)
118     all_representations = tree2.xpath("//*[local-name() =
        'Representation']")
119     # Get all available representations from the original MPD file
120     avail_idxs = list(map(int, map(lambda rep: rep.get("id"),
        all_representations)))
121
122     # Check that current max. representation is not already the
        lowest one
123     if(current_max_id != 1):
124
125         # Get the index of the next lower representation
126         idx = avail_idxs.index(current_max_id - 1)
127
128         # Replace all occurences of the highest representations
129         # with the next lower representation
130         for max_idx in current_max_idx[0]:
131             max_representation = representations[np.int(max_idx)]
132             parent = max_representation.getparent()
133             parent.remove(max_representation)
```

```
134             lower_representation =
    copy.deepcopy(all_representations[idx])
135             #max_representation.attrib['id'] = id
136             parent.append(lower_representation)
137
138     # Nothing happens if we only present the lowest
139     # representation.
140     elif(current_max_id == 1):
141         return
142
143     # Update the dynamically updated MPD Output file
144     f = open(mpd_output, 'w')
145     f.write(etree.tostring(tree, xml_declaration=True,
    pretty_print=True))
146     f.close()
147
148     return
149
150 # Removes the current lowest representation in the MPD Output file
151 # and replaces it with the next higher representation available
    from
152 # the MPD Input file.
153 def remove_lowest_representation(mpd_input, mpd_output):
154
155     tree = etree.parse(mpd_output)
156     representations = tree.xpath("//*[local-name() =
    'Representation']")
157
158     # Get the IDs of the currently available representations
159     avail_idxs = list(map(int, map(lambda rep: rep.get("id"),
    representations)))
160     # Current min. representation id
161     current_min_id = min(avail_idxs)
162     # Index of ALL the current min. representation ids
163     current_min_idx = np.where(np.array(avail_idxs) ==
    min(avail_idxs))
164
165     tree2 = etree.parse(mpd_input)
166     all_representations = tree2.xpath("//*[local-name() =
    'Representation']")
```

```
167     avail_idxs = list(map(int, map(lambda rep: rep.get("id"),
        all_representations)))
168
169     # Make sure we are not already providing the highest available
        representation
170     max_id = max(avail_idxs)
171     if(current_min_id != max_id):
172        # Get the idx of the next higher representation
173        idx = avail_idxs.index(current_min_id + 1)
174
175          # Replace all occurences of the lowest representations
176        # with the next higher representation
177        for min_idx in current_min_idx[0]:
178           min_representation = representations[np.int(min_idx)]
179           parent = min_representation.getparent()
180           parent.remove(min_representation)
181           lower_representation =
        copy.deepcopy(all_representations[idx])
182           #max_representation.attrib['id'] = id
183           parent.append(lower_representation)
184
185     # Nothing happens if we only present the highest
186     # representation.
187     elif(current_min_id == max_id):
188        return
189
190     # Update the dynamically updated MPD Output file
191     f = open(mpd_output, 'w')
192     f.write(etree.tostring(tree, xml_declaration=True,
        pretty_print=True))
193     f.close()
194
195     return
196
197 if __name__ == "__main__":
198     main()
```

B.6 Modifications to the MP4Client

LISTING B.11: Modifications to the client, to dynamically refresh the MPD file, in /src/media_tools/dash_client.c

```
1
2           while (!dash->mpd_stop_request) {
3               u32 timer = gf_sys_clock() - dash->last_update_time;
4 +             dash->mpd->minimum_update_period = 5000;
```

Bibliography

[1] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2016–2021," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf, 2007, [Online; accessed 21-January-2018].

[2] F. Giust, G. Verin, K. Antevski, J. Chou, Y. Fang, W. Featherstone, F. Fontes, D. Frydman, A. Li, A. Manzalini, D. Purkayastha, D. Sabella, C. Wehner, K.-W. Wen, and Z. Zhou, "MEC Deployments in 4G and Evolution Towards 5G," ETSI White Paper No. 24, 2018. [Online]. Available: http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp24_MEC_deployment_in_4G_5G_FINAL.pdf

[3] R. Nossenson, "Long-term evolution network architecture," in *2009 IEEE International Conference on Microwaves, Communications, Antennas and Electronics Systems*, Nov 2009, pp. 1–4. [Online]. Available: http://doi.org/10.1109/COMCAS.2009.5385947

[4] E. Dahlman, S. Parkvall, and J. Skold, *4G: LTE/LTE-Advanced for Mobile Broadband*, 1st ed. Academic Press, 2011.

[5] M. Patel, Y. Hu, P. Hédé, J. Joubert, C. Thornton, B. Naughton, J. R. Ramos, C. Chan, V. Young, S. J. Tan, D. Lynch, N. Sprecher, T. Musiol, C. Manzanares, U. Rauschenbach, S. Abeta, L. Chen, K. Shimizu, A. Neal, P. Cosimini, A. Pollard, and G. Klas, "Mobile-Edge Computing – Introductory Technical White Paper," ETSI White Paper, 2014. [Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf

[6] E. Schiller, N. Nikaein, K. Eirini, M. Gasparyan, and T. Braun, "CDS-MEC: NFV/SDN-based application management for MEC in 5G Systems," *Elsevier*, 2018.

[7] C.-Y. Chang, K. Alexandris, N. Nikaein, K. Katsalis, and T. Spyropoulos, "Mec architectural implications for lte/lte-a networks," in *Proceedings of the Workshop on Mobility in the Evolving Internet Architecture*, ser. MobiArch '16. New York, NY, USA: ACM, 2016, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/2980137.2980139

[8] B. Nguyen, N. Choi, M. Thottan, and J. V. der Merwe, "Simeca: Sdn-based iot mobile edge cloud architecture," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 503–509. [Online]. Available: http://doi.org/10.23919/INM.2017.7987319

[9] A. Huang, N. Nikaein, T. Stenbock, A. Ksentini, and C. Bonnet, "Low latency mec framework for sdn-based lte/lte-a networks," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6. [Online]. Available: http://doi.org/10.1109/ICC.2017.7996359

[10] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "Softran: Software defined radio access network," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491207

[11] X. Foukas, N. Nikaein, M. M. Kassem, and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks," in *CONEXT 2016, 12th International on Conference on emerging Networking EXperiments and Technologies, Irvine, California, USA*, Irvine, UNITED STATES, 12 2016. [Online]. Available: http://dx.doi.org/10.1145/2999572.2999599

[12] S. Lederer, "Why Youtube & Netflix use MPEG-DASH in HTML5," https://bitmovin.com/status-mpeg-dash-today-youtube-netflix-use-html5-beyond/, February 2015, [Online; accessed 21-January-2018].

[13] 3rd Generation Partnership Project (3GPP), "Transparent end-to-end Packet-switched Streaming Service (PSS); Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH)," 3GPP, Tech. Rep., 2015.

[14] C. Müller, D. Renzi, S. Lederer, S. Battista, and C. Timmerer, "Using scalable video coding for dynamic adaptive streaming over http in mobile

environments," in *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, Aug 2012, pp. 2208–2212.

[15] E. Thomas, M. van Deventer, T. Stockhammer, A. Begen, M.-L. Champel, and O. Oyman, "Applications and deployments of server and network assisted dash (sand)," *IET Conference Proceedings*, pp. 22 (8 .)–22 (8 .)(1), January 2016. [Online]. Available: http://digital-library.theiet.org/content/conferences/10.1049/ibc.2016.0022

[16] D. I. F. (DASH-IF), "DASH-IF Position Paper: Server and Network Assisted DASH (SAND)," 2016. [Online]. Available: https://dashif.org/wp-content/uploads/2017/01/SAND-Whitepaper-Dec13-final.pdf

[17] T. Karagkioules, D. Tsilimantos, C. Concolato, and S. Valentin, "A comparative case study of HTTP adaptive streaming algorithms in mobile networks," *CoRR*, vol. abs/1705.01762, 2017. [Online]. Available: http://arxiv.org/abs/1705.01762

[18] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan, "Mp-dash: Adaptive video streaming over preference-aware multipath," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. New York, NY, USA: ACM, 2016, pp. 129–143. [Online]. Available: http://doi.acm.org/10.1145/2999572.2999606

[19] C. Xu, T. Liu, J. Guan, H. Zhang, and G. M. Muntean, "Cmt-qa: Quality-aware adaptive concurrent multipath data transfer in heterogeneous wireless networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 11, pp. 2193–2205, Nov 2013. [Online]. Available: http://doi.org/10.1109/TMC.2012.189

[20] C. Cetinkaya, Y. Ozveren, and M. Sayit, "An sdn-assisted system design for improving performance of svc-dash," in *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sept 2015, pp. 819–826. [Online]. Available: http://doi.org/10.15439/2015F333

[21] Y. Li, P. A. Frangoudis, Y. Hadjadj-Aoul, and P. Bertin, "A mobile edge computing-assisted video delivery architecture for wireless heterogeneous networks," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, July 2017, pp. 534–539. [Online]. Available: http://doi.org/10.1109/ISCC.2017.8024583

[22] C. f. Lai, R. h. Hwang, H. c. Chao, M. M. Hassan, and A. Alamri, "A buffer-aware http live streaming approach for sdn-enabled 5g wireless networks," *IEEE Network*, vol. 29, no. 1, pp. 49–55, Jan 2015. [Online]. Available: http://doi.org/10.1109/MNET.2015.7018203

[23] J. O. Fajardo, I. Taboada, and F. Liberal, "Improving content delivery efficiency through multi-layer mobile edge adaptation," *IEEE Network*, vol. 29, no. 6, pp. 40–46, Nov 2015. [Online]. Available: http://doi.org/10.1109/MNET.2015.7340423

[24] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[25] Y. Li, P. A. Frangoudis, Y. Hadjadj-Aoul, and P. Bertin, "A mobile edge computing-based architecture for improved adaptive http video delivery," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, Oct 2016, pp. 1–6. [Online]. Available: http://doi.org/10.1109/CSCN.2016.7784892

[26] P. Bahl and V. N. Padmanabhan, "Radar: an in-building rf-based user location and tracking system," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 2, 2000, pp. 775–784 vol.2. [Online]. Available: http://doi.org/10.1109/INFCOM.2000.832252

[27] N. Nikaein, R. Knopp, F. Kaltenberger, L. Gauthier, C. Bonnet, D. Nussbaum, and R. Ghaddab, "Demo: Openairinterface: An open lte network in a pc," in *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '14. New York, NY, USA: ACM, 2014, pp. 305–308. [Online]. Available: http://doi.acm.org/10.1145/2639108.2641745

[28] N. Nikaein, R. Knopp, L. Gauthier, E. Schiller, T. Braun, D. Pichon, C. Bonnet, F. Kaltenberger, and D. Nussbaum, "Demo: Closer to cloud-ran: Ran as a service," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15.

New York, NY, USA: ACM, 2015, pp. 193–195. [Online]. Available: http://doi.acm.org/10.1145/2789168.2789178

[29] B. Pfaff, J. Pettit, T. Koponen, E. J. . Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. String er, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Des ign and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 117–130. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789779

[30] P. Frenger, S. Parkvall, and E. Dahlman, "Performance comparison of harq with chase combining and incremental redundancy for hsdpa," in *IEEE 54th Vehicular Technology Conference. VTC Fall 2001. Proceedings (Cat. No.01CH37211)*, vol. 3, 2001, pp. 1829–1833 vol.3. [Online]. Available: http://doi.org/10.1109/VTC.2001.956516

[31] Goddard Media Studio, "NASA Enters World of Ultra-High-Def (4K) Video," https://svs.gsfc.nasa.gov/12034, November 2015, [Online; accessed 28-January-2018].

[32] S. Lederer, "MPEG-DASH Content Generation with MP4Box and x264," https://bitmovin.com/mp4box-dash-content-generation-x264/, November 2014, [Online; accessed 28-January-2018].

[33] T. Karagkioules, C. Concolato, D. Tsilimantos, and S. Valentin, "A comparative case study of http adaptive streaming algorithms in mobile networks," in *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV'17. New York, NY, USA: ACM, 2017, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/3083165.3083170

[34] B. Augustin and A. Mellouk, "On traffic patterns of http applications," in *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, Dec 2011, pp. 1–6. [Online]. Available: http://doi.org/10.1109/GLOCOM.2011.6134438

[35] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," RFC, Tech. Rep., 2009.

[36] R. N. Bracewell and R. N. Bracewell, *The Fourier transform and its applications*. McGraw-Hill New York, 1986, vol. 31999.

[37] H. Shatkay, "The Fourier Transform - A Primer," November 1995. [Online]. Available: ftp://ftp.cs.brown.edu/pub/techreports/95/cs95-37.pdf

[38] S. W. Smith *et al.*, "The scientist and engineer's guide to digital signal processing," 1997.

[39] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[40] C. E. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, Jan 1949. [Online]. Available: http://doi.org/10.1109/JRPROC.1949.232969

[41] B. Furht and S. Ahson, *Long Term Evolution: 3GPP LTE Radio and Cellular Technology*, ser. Internet and Communications. CRC Press, 2016. [Online]. Available: https://books.google.ch/books?id=eUISAI5F7z4C

[42] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer; Measurements," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.214, 03 2017, version 14.2.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2428

[43] J. Agrawal, P. Mor, J. M. keller, R. Patel, and P. Dubey, "Introduction to the basic lte handover procedures," in *2015 International Conference on Communication Networks (ICCN)*, Nov 2015, pp. 197–201. [Online]. Available: http://doi.org/10.1109/ICCN.2015.39

[44] S. F. Sulthana and R. Nakkeeran, "Study of downlink scheduling algorithms in lte networks," *JNW*, vol. 9, pp. 3381–3391, 2014.