Improving Mobile Video Quality Through Predictive Channel Quality Based Buffering

Jan Willem Kleinrouweler Centrum Wiskunde & Informatica Amsterdam, The Netherlands j.w.m.kleinrouweler@cwi.nl

Hans van den Berg TNO, Centrum Wiskunde & Informatica The Hague, The Netherlands j.l.vandenberg@tno.nl Britta Meixner Centrum Wiskunde & Informatica Amsterdam, The Netherlands britta.meixner@cwi.nl

Rob van der Mei Centrum Wiskunde & Informatica, VU University Amsterdam Amsterdam, The Netherlands r.d.van.der.mei@cwi.nl Joost Bosman Centrum Wiskunde & Informatica Amsterdam, The Netherlands j.w.bosman@cwi.nl

Pablo Cesar Centrum Wiskunde & Informatica, Delft University of Technology Amsterdam, The Netherlands p.s.cesar@cwi.nl

Abstract—Frequent variations in throughput make mobile networks a challenging environment for video streaming. Current video players deal with those variations by matching video quality to network throughput. However, this adaptation strategy results in frequent changes of video resolution and bitrate, which diminishes the users' streaming experience. Alternatively, keeping the video quality constant would improve the experience, but puts an additional demand on the network. Downloading high quality content when channel quality is low requires extra resources, because data transfer efficiency is linked to channel quality. In this paper, we present a predictive Channel Quality based Buffering Strategy (CQBS), that grows the video buffer when channel quality is good, and relies on this buffer when channel quality decreases. Our strategy is the outcome of a Markov Decision Process. The underlying Markov chain is conditioned on 377 real-world LTE channel quality traces that we have collected using an Android mobile application. With our strategy, mobile network providers can deliver constant quality video streams, without sacrificing additional network resources.

Index Terms—Video streaming; HTTP Adaptive Streaming, Markov model, Markov Decision Process, Buffering strategy

I. INTRODUCTION

Online video streaming is one of the most popular applications on fixed and mobile networks. Network traffic predictions, such as Sandvine's Internet phenomena report [1], not only show an increase in data volume but also show that on-demand video streaming is the dominant driver for this increase. Currently, around 70% of downlink traffic accounts for video streaming. However, mobile networks are a challenging environment to deliver high-quality video. A shared radio-based medium combined with user movement creates high variability in channel quality from eNodeB to User Equipment (UE). The effective throughput to an UE is strongly dependent on cell load and radio channel conditions.

Dynamic Adaptive Streaming over HTTP (DASH) is the primary video streaming technique [2]. It has shown to effectively deal with throughput variations [3], [4]. DASH players adapt the video quality to match network throughput. On the one hand, lowering the video quality when network conditions decrease reduces annoying playback interruptions. On the other hand, increasing the video quality when throughput increases provides a benefit for the user. However, temporarily streaming at low quality and too many quality switches negatively impact the viewers' Quality of Experience (QoE) [5]. It could even lead to users stopping the stream [6].

The next generation of mobile networks potentially offers better support for video streaming. In 5G networks, so-called network slices can be used to optimize the network for delivery of video streams. Having the 5G architecture in mind, we investigate the potential of a network slice that requests network resources such that a constant video quality can be maintained. Instead of having video players adapting their quality to the mobile network conditions, we propose a strategy that adjusts the number of network resources to let players stream at constant video quality.

Streaming at a constant video quality is possible but requires more network resources. In LTE terms this means increasing usage of Resource Blocks (RBs). The effective data rate per RB strongly depends on the quality of the radio channel between eNodeB to UE. When the channel quality decreases, the eNodeB uses modulation settings with more redundancy, thus effectively reducing the throughput per RB to the UE. Maintaining a constant video quality means using a relatively large number of RBs when channel quality is low.

Traditional adaptation algorithms are efficient because they lower the video quality when the channel quality decreases, thus relieving the network when it's needed. In this paper, we propose a downloading strategy that also relieves the network when channel quality is low, but it maintains constant video quality. The intuition behind our predictive Channel Quality based Buffering Strategy (CQBS) is to grow the video buffer when channel quality is high (bandwidth is thus relatively cheap), and consume this buffer when channel quality decreases (relieve the network when bandwidth becomes expensive). To determine when the buffer should grow or shrink, we compose a Markov chain that represents the changes and behavior of LTE channel quality. The Markov chain is conditioned on an extensive set of 377 5-minute LTE channel quality traces containing measurements in different environments and under different speeds. We define a Markov Decision Process (MDP) based on this model and the current play-out buffer level in the video player to obtain the optimal downloading strategy. With CQBS we can deliver constant video quality for the same price as the varying video quality from traditional adaptation algorithms. We summarize the contribution in this paper as follows:

- A Markov chain that describes the behavior of LTE channel quality. The model is conditioned on an extensive set of LTE channel quality traces that we have collected in the real word using an Android application.
- CQBS: a predictive downloading strategy for video streaming (grow-, maintain-, shrink the video buffer) that reduces network resources while streaming in a constant video quality. We formulate the strategy as an MDP.
- Performance evaluation of CQBS, comparing it to traditional adaptation algorithms. We cross-validate CQBS using our real-world channel quality traces.

The remainder of this paper is as follows. Section II discusses related work. In Section III we describe our Markov model for LTE channel quality, which we use in Section IV to compute the optimal downloading strategy. In Section V evaluate the performance of our CQBS algorithm. Sections VI discusses our findings and concludes this paper.

II. RELATED WORK

Dynamic Adaptive Streaming over HTTP (DASH) is the dominant video streaming technology for the Internet [7]. In DASH, a video file is encoded at multiple bitrates and resolutions. Each representation is then split into small segments of a few seconds. A manifest file describes the characteristics of each representation and contains URLs where to find each segment [2]. A DASH player downloads the video segments one-by-one adapting the video quality to the network conditions. This process is challenging [8], especially in mobile networks that show large variations in throughput. The resulting fluctuations in video quality are distracting for the viewers [5], and may lead to users stopping the stream [6].

To improve the stability of video streams, so called DASH Aware Network Elements (DANEs) are introduced [9]. DANEs are a cross-utilization solution where DASH players exchange information with network elements, such as routers, Wi-Fi access points, and potentially mobile base stations. Most DANEs (or comparable solutions) are targeting wired or Wi-Fi networks. Bouten et al. use proxy servers to guide DASH players [10]. Cofano et al. [11] and Kleinrouweler et al. [12] both presented implementations that employ WebSockets for information exchange and apply traffic shaping to guarantee that enough bandwidth is available for video streaming.

DANEs also show to be useful in mobile networks, where they primarily focus on reducing video freezes. In [13], With et al. present a cross-layer solution where the LTE eNodeB has access to the information in the DASH manifest. It allows the eNodeB to better schedule resources and reduce the number of freezes. Essaili et al. present a pro-active resource scheduler for DASH [14]. Based on the clients' buffer levels, the scheduler determines the best streaming rate. This reduces video freezes and is more fair among clients. Zahran et al., present a solution with a comparable goal and effect [15].

In this paper, we assume the existence of a DANE implementation that can exchange information about channel quality, videos stream, and player state (i.e. play-out buffer level) between mobile base station and client. Instead of only looking at the current state of network and video player, we also predict the evolution of channel quality while determining our buffering strategy. Wang et al. present a channel quality predictor based on machine learning techniques [16]. They target real-time applications but not consider a specific application. We look alter the buffering process of DASH based video streaming based a Markov chain that is conditioned on real-word data.

III. MODELING CHANNEL QUALITY BEHAVIOR

The data transmission efficiency in mobile networks depends on the channel quality. We anticipate on this by downloading more video content into the buffer when efficiency is high, and play video from the buffer when efficiency decreases. To determine when to start growing/consuming the video buffer, we have to understand the evolution of channel quality. We follow the next three steps:

- A) collect an extensive set of real-word channel quality measurements with a mobile application;
- B) fit a Markov chain on the collected traces and determine data transmission efficiency for each state;
- C) predict data transmission efficiency.

A. Collecting channel quality traces

In LTE networks, UEs determine the channel quality using a reference signal that is broadcasted by the eNodeB. This process is known as channel estimation. One of the parameters that the UE obtains during channel estimation is the socalled Reference Signal Received Quality (RSRQ). The RSRQ reporting range is defined by the 3GPP from -20 dB (very poor channel quality) to -3 dB (excellent channel quality) [17].

To the best of our knowledge, a large and consistent dataset that includes RSRQ parameters does not exist. Therefore, we developed a smartphone application for the Android platform to measure LTE downlink channel quality. The app uses the Android 8.0 Telephony API¹ to gather LTE statistics at intervals of two seconds. This interval is the lowest update frequency that the platform provides. RSRQ is subject to frequent small changes as a result of interference and multipath fading. Therefore, UEs internally determine RSRQ (and other parameters) every two milliseconds. Nevertheless, the Telephony API gives a good summary of the last two seconds. Small changes would have little to no effect on downloading

¹https://developer.android.com/reference/android/telephony/packagesummary.html (accessed March 20, 2018)

video segments of a few seconds. The Telephony API reports RSRQ as discrete values within the RSRQ reporting range.

We used two Google/LG Nexus $5X^2$ smartphones for collecting the traces. We measured LTE channel quality in traces of five minutes in several different environments, under different speed profiles and with a varying number of people around us. In total, we have collected real-world 377 traces.

B. Modeling channel quality and data rate behavior

Through analysis of the channel quality traces we obtained how RSRQ evolves over time. Based on this information, we construct a Markov chain to fit this process. The state space, S, is defined as eighteen states representing the RSRQ reporting range in steps of one decibel. Transitions in the Markov chain express changes in RSRQ. We observed in our traces that RSRQ can change between any two levels. Therefore, we use a fully connected Markov chain. The transition probability matrix is denoted by P. We opt for a transition interval of 1/3 second. The two-second intervals from our channel quality traces are too coarse to express the download process of DASH video segments at different buffering speeds. A too small transition interval would cause long execution times when solving the MPD.

For each state in the Markov chain, the data transmission efficiency that corresponds with the RSRQ is computed. We follow a process that is similar to how the modulations settings (modulation settings determine the efficiency) in LTE network are established. In the first step, we map RSRQ to Signal to Noise and Interference Ratio (SINR). A theoretical mapping between RSRQ and SINR exists, and is approximated using the following function [17], [18]:

$$SINR = \frac{1}{\frac{1}{12 - RSRQ} - \rho},\tag{1}$$

where ρ is the load of the serving cell. We use $\rho = 1/6$, which indicates a lightly loaded cell. Given the SINR, we use a lookup table to obtain the Channel Quality Indicator (CQI) and corresponding data rate. The data rate is the fraction of bits in a Resource Block (RB) that remains after demodulation (i.e. the actual data). Table I lists the mapping from RSRQ to the data rate. The function d(x) applies this mapping and provides the effective data for state x.

C. Predicting transmission efficiency

Given the RSRQ Markov chain and the mapping to effective data rate, we can make predictions on how the data will evolve. We will use those predictions to estimate how many resources will be required to download the next video segment at a certain speed (i.e. grow, maintain, or shrink the video buffer). Depending on download speed, the downloading a video segment may to take longer than one interval in our model. We compute the multi-step transition probabilities to determine how RSRQ and data rate change while downloading this video segment. Given transition probability matrix P, we

Table I MAPPING RSRQ TO EFFECTIVE DATA RATE

RSRQ (dB)	SINR (dB)	CQI	Eff. data rate
-20	-9.12	0	0.000
-19	-8.10	0	0.000
-18	-7.07	0	0.000
-17	-6.03	1	0.026
-16	-4.98	1	0.026
-15	-3.92	2	0.039
-14	-2.85	2	0.039
-13	-1.75	3	0.063
-12	-0.06	3	0.063
-11	0.54	4	0.101
-10	1.76	4	0.101
-9	3.05	5	0.147
-8	4.45	6	0.198
-7	6.00	6	0.198
-6	7.82	7	0.248
-5	10.13	9	0.404
-4	13.70	10	0.459
-3	INF	15	0.930

obtain the *n*-step transition probabilities by multiplying *P*. We denote the probability that our RSRQ model transitions from state x to y in n steps as $P_{x,y}^n$.

Driven by the changes in RSRQ that can occur during one segment download, the effective data rate may also vary. To get the expected network resources needed for one video segment we need to estimate the data rate during that download. Therefore, we average the data rates for each possible path between two states, weighing each path by its probability of taking it. We compute the expected data rate when transitioning from state x to y in n steps as follows:

$$D_{x,y}^{n} = \frac{1}{n} \sum_{i=1}^{n} \sum_{z \in \mathcal{S}} \frac{\lceil P_{z,y}^{n-i} \rceil P_{x,z}^{i}}{\sum_{w \in \mathcal{S}} \lceil P_{w,y}^{n-i} \rceil P_{x,w}^{i}} d(z).$$
(2)

For every of the *n* steps, we compute the expected data rate in that step. The expected data rate for a step is the average data rate of all states, weighted by the probabilities that a route from *x* to *y* traverses a state in the given step. However, not all paths provide a viable route $x \rightarrow y$, especially when the number of steps becomes small. Therefore, we only consider transitions that after a transition have enough steps left to reach destination state *y*. For each sub-state *z* in Equation 2, we only include *z* when the probability of transitioning $z \rightarrow y$ in n - isteps is non-zero. Since not all states are included, we adjust the weights accordingly.

IV. CHANNEL QUALITY BASED BUFFER STRATEGY

The number of network resources that are required for downloading one segment can be estimated with $D_{x,y}^n$ and the current channel conditions. How many intervals are needed to download one segment depends on how fast it should be downloaded. Assuming a segment with a duration of two seconds, downloading it in two seconds would maintain the buffer level. Downloading faster increases the buffer, while downloading slower than two seconds (or not at all) would

²http://www.lg.com/in/lg-nexus-5x/specification.jsp (accessed January 23, 2018)

shrink the buffer. We define six different buffering actions (denoted as *Bxxx*) for our CQBS strategy:

- **Growing the buffer**: Downloading video segments two (*B200*) or three (*B300*) times faster than playing segments out.
- Maintaining the buffer: Download of one video segment takes the same time as playing one segment (*B100*). This action will be chosen over *B200/300* when the buffer reaches its maximum level.
- Shrinking the buffer: Relieving the network when bandwidth becomes expensive. We download video segments either at $\frac{1}{3}$ (*B033*) or $\frac{1}{2}$ (*B050*) of the playback speed.
- Not downloading: Data transmission are sometimes not possible because channel quality it too low. During those periods, no video segments will be downloaded and no resources are required (*B000*).

Depending on the buffering action, downloading one twosecond segment will take between two and eighteen intervals of 1/3 second. The channel quality of the client and the current buffer fill level determine the best buffer action at each time. We formulate this problem, which is the core of CQBS, as an MDP. Let (S', A, P', R, γ) be our MDP. The state space S', describes the state of a single video player instance. A state is defined as a two-tuple (c, b) combining the current channel quality (indicated by RSRQ) with the current buffer level. Buffer fill levels are modeled as the number of video units of 1/3 second, aligned with the transitions intervals in our Markov chain. As such, one video segment of two seconds consists of six video units. The buffer is limited to 60 seconds or 180 video units.

The set of actions, A, covers the six buffering actions, B000 to B300. Based on the buffering action, and the download time of one segment given that action, certain transitions within S' are possible. For example, downloading a two-second segment at speed B200 takes three steps. While downloading six video units, the video player plays out three units. The buffer grows with three units. For buffering action B200, transitions are possible to all states in S', such that $b \rightarrow b + 3$. The probabilities for these transitions are obtained from the 3-step transition probability matrix of the RSRQ model, $P_{x,y}^3$. Following this principle, the transition matrix P' is build for each combination of RSRQ and buffer level.

The rewards for each transition, R, are in general linked to the expected cost in the network. For each action and transition, we compute the expected load on the network as the percentage of resource blocks allocated for the video player. We calculate the expected load as follows:

$$ld(x, y, n) = \frac{B \cdot 3T_{\text{segment}}}{D_{x, y}^{n}} \cdot \frac{1}{\frac{1}{3}Rbs \cdot n}$$
(3)

where B is the video bitrate, $T_{segment}$ the duration of a video segment in seconds, Rbs the number of resource blocks that is available per second, and n the number of steps that the segment download takes. Whether the load is used in the reward depends on the following three step process:

- 1) Check if the buffering action is possible. Only actions that do not cause an overload on the system (ld(x, y, n) > 1) are accepted. Also the *B000* action is restricted to only being used when channel quality is too low for data transmission. A high negative reward (-10000) to ensure invalid actions.
- Check if the buffering action would lead to an empty buffer. In case the buffer becomes empty, a penalty of -50 times the duration of the freeze (in ticks) is assigned as a reward for the transitiotns
- 3) When 1) and 2) do not apply the inverse load (1 ld(x, y, n)) is used as reward.

We solve the MDP using the value iteration algorithm with discount factor $\gamma = 0.999$ from the Python-based mdptoolbox³. The resulting policy is the optimal buffering strategy. For each channel quality and buffer level the strategy provides the buffering action (B000 to B300) to take. A visualization of our strategy is shown in Figure 1. The lighter the color is in the visualization the faster should be downloaded. The strategy reflects the intuition behind CQBS. When the network resources becomes expensive, the buffering actions that shrink the buffer are selected. Only when the buffer level is low, the buffer has to grow, or has to be maintained as a minimum. As such, the strategy avoids the penalty for an empty buffer. When network traffic becomes cheaper, the strategy lets the buffer grow. The maximum fill level of the buffer grows with the channel quality. This means that the strategy stops filling the buffer at some point. Only when the channel quality further increases the strategy continues to grow the buffer.



Figure 1. Buffering strategy for constant video quality (1458 Kbit/s), best buffering action for each channel quality and buffer level.

Figure 2 shows an example streaming session with the channel quality and the buffer level overlaid in the same plot. Initially, CQBS grows the buffer when channel quality is good. Around, t = 60, the signal quality decreases and more video from the buffer is used. The buffer level is restored when signal quality restores. Around t = 350, the signal quality becomes excellent. This triggers CQBS to fill the buffer to a higher level than it did before.

³http://pymdptoolbox.readthedocs.io/en/latest/api/mdptoolbox.html (accessed March 12, 2018)



Figure 2. Example operation of CQBS: interaction between channel quality and buffer level.

V. PERFORMANCE EVALUATION

In this section, we evaluate CQBS's performance in terms of video quality and required network resources. We compare CQBS to two traditional adaptation algorithms and to a naive strategy that maintains a constant quality. We start with a description of the simulation setup. Then we evaluate the performance of CQBS with traces that are based on random walks in the RSRQ Markov chain. Last, we cross-validate CQBS agains our real-world channel quality traces.

A. Simulation setup

In a real LTE network, a moving client would roam between cells with different characteristics and loads. However, as a first step into this type of buffering strategies, we will look into the performance of CQBS from the perspective of a single client (i.e. how many resources would be required to execute the strategy) in a heterogeneous environment. We assume that the video streaming client gets the resources assigned that it needs. We use discrete event simulations to simulate an LTE network and a video streaming application. For simplicity, the simulation covers a single video node that is associated to one LTE base station. The LTE base station is configured with 15 MHz bandwidth, which has 75 RBs available every millisecond. In this section, we denote the network load as the fraction of those RBs that are used by our video streaming application. The simulation of channel quality between the LTE base station and mobile client are obtained by random walks in the RSRQ Markov chain, or from our real-world channel quality traces. Even though we use a single LTE base station, the RSRQ behavioris consistent with a roaming client.

The video player that runs on the client node simulates the download of a DASH video stream. A video clip is taken from the movie Sintel⁴ and encoded in ten representations⁵ (i.e. combinations of bitrate and resolution). The video is segmented for DASH with a segment size of two seconds. The video player inform the LTE base station about it its current

buffer level and channel quality. The player signals the base station every time before downloading the new video segment. The base station executes our CQBS strategy by combining the buffer level with the current channel quality, selecting the best buffer action and allocating appropriate resources.

As part of this evaluation, we compare four different adaptation and buffering strategies:

- 1) **Conventional:** The conventional algorithms uses the download speed of previous video segments to estimate future bandwidth. The video quality will be the highest bitrate that is below this estimation.
- BOLA: An implementation of the Buffer Occupancybased Lyapunov Algorithm from Spiteri et al. [4]. We use a version that reduces video quality oscillations, comparable to the reference implementation in the DASH.js⁶ player.
- 3) Static: A naive implementation that produces static video quality. Until the buffer is full, resources for twice the video bitrate are allocated to quickly growing the buffer and prevent freezes. When the buffer is full, resources matching the video bitrate are allocated. When the video bitrate exceeds the networks capacity (e.g. when signal quality is very low), the maximum amount of resources are allocated.
- 4) CQBS: Maintains a constant video quality but adjusts its buffer based on the current channel quality. Resources between one third- and three times the video bitrate are allocated, depending on the selected buffering action. When the video buffer drops below four seconds (i.e. two video segments), and the channel quality does not permit streaming in the target quality, the video player temporarily lowers its video quality, as part of a failover mechanism to ensure uninterrupted playback.

For the conventional and BOLA adaptation algorithms we limited the number of resources so that these algorithms would on average yield the desired video quality. As such, we can compare these traditional algorithms to a naive static video quality strategy and our CQBS strategy. The maximum allocated resources for each of the evaluated quality levels are listed in Table II. The allocation settings are specified as the maximum fraction of RBs that can be used by our video streaming client. Based on the allocation settings, we can already observe that BOLA yields the same average video quality while the maximum allocation is at least 20% lower.

B. Performance evaluation

In the first part of the evaluation we use model-based simulations. Per setting, we generated 25.000 channel quality traces by randomly walking the RSRQ Markov chain from Section III. As such, we assume that our strategy perfectly knows channel quality evolution. Given those ideal settings, our CQBS will perform at its maximum, providing a good indication of the potential of this solution. In each instance we

⁴https://durian.blender.org (last accessed: February 20, 2017)

⁵296Kbit/s@240p, 395Kbit/s@240p, 493Kbit/s@360p, 732Kbit/s@360p, 971Kbit/s@480p, 1.458Kbit/s@480p, 1.934Kbit/s@720p, 2.878Kbit/s@720p, 3.779Kbit/s@1080p, 5.544Kbit/s@1080p

⁶https://github.com/Dash-Industry-Forum/dash.js/wiki (accessed: March 5, 2018)

Table II RESOURCE RESERVATION SETTINGS

Target quality	3	4	5	6	7	8	9	10
Target bitrate	493	732	971	1458	1934	2878	3779	5544
Conventional adaptation allocation	0.063	0.095	0.140	0.200	0.280	0.400	0.650	1.00
BOLA allocation	0.045	0.072	0.110	0.156	0.216	0.305	0.435	1.00

stream a ten minute video, while measuring the video quality and load on the network. A video quality level, ranging from 1 to 10, is used instead of the video bitrate. The encoding settings of our video are chosen such that every step gives a comparable increase in quality. However, the bitrate that is needed for each step increases with the video quality. Averaging over video bitrate would give skewed results.

First, we look into the differences in video quality between the traditional adaptation algorithms and the constant video strategies. The target video quality is level five. For each instance we computed the mean video quality. The distribution of mean video qualities is shown in Figure 3. For the static and CQBS strategies, this figure reveals that the overall video quality matches the target quality in almost all instances. The exceptional run suffered from either low channel quality at the startup or long periods of low channel quality during the run. As result, the fallback mechanism temporarily lowered the video quality. For the conventional and BOLA algorithms the mean video quality lies within one and a half video quality levels around the target.



Figure 3. Distribution of overall video quality, target video quality level five.

The video quality during playback is also not constant for the conventional and BOLA algorithms. These adaptation algorithms adapt the video quality to match the network conditions. Because the network conditions change (as a result of the channel condition chaning), the video quality changes as well. The distribution of video quality during the runs is shown in Figure 4. The figure shows that the video qualities are spread out over all levels, as can be seen by comparing the dashed lines to the solid lines in Figure 4.

When using the conventional and BOLA algorithm, the video is streamed in the target quality only a relatively small fraction of the time. The broad spread of video qualities during



Figure 4. Distribution of video quality during the runs, target video quality level five.



Figure 5. Number of quality switches as a function of video quality.

each run indicates that the video quality changes from time to time. In Figure 5, we observe that both the conventional algorithm and BOLA show many video quality switches. The conventional algorithm performs worst, resulting in more than 12 quality switches per minute for most of the target quality levels. BOLA perform better, but the switching frequency is still high. A big difference can be observed from target quality level 10. In this case, all algorithms had the full spectrum available for streaming. In the case of BOLA, it meant that it could maintain large buffers. Due to the nature of BOLA, an almost full buffer will result in video segments of the highest quality. Even when the signal quality decreased, the buffer levels were still sufficient for BOLA to request video quality level 10. As the result, BOLA worked similar (except during the startup phase) to the naive static strategy. This resulted in little quality switches, but to an increased network load as we will discuss next in this section.

The high number of quality switches when using the traditional algorithms are noticeable for the user and can be perceived as annoying. In comparison, the naive static quality strategy and CQBS maintain an almost perfectly constant video quality. Although providing a constant video quality may be more beneficial for the end-user, it could increase cost for the network operator. Figure 6 shows the distribution of network load (percentage of RBs that were used for video streaming) per instance. For most of the instances, the network load for the conventional algorithm and BOLA is comparable. Nevertheless, BOLA shows some instances that have relatively high load. Comparing the medians of the traditional algorithms with the naive static quality strategy, we observe and increase in network load of almost three percent points. This 25% increase in network load takes up resources that cannot be used by other clients in the same cell. The increase makes a difference over ten minutes, from up to 2 Mbit/s bandwidth for clients with good signal quality, to about 330 Kbit/s for an average client. This bandwidth cannot be used anymore for several web-browsers, a high quality music stream, or a low quality video stream.



Figure 6. Distribution of overall network load per instance

Our CQBS strategy is designed to reduce the load on the network. It consumes video from the buffer when network bandwidth is expensive. When channel quality becomes better and bandwidth comparatively cheap, it re-fills the video buffer. Figure 6 shows the impact of this strategy on the network load. On average, the use of network resources for video streaming is on the same level as the traditional adaptation algorithms. In 23% to 64% of the cases, CQBS requires less resources than the conventional and BOLA algorithms.

Over the whole range of target video quality levels, we observe similar behavior. Figure 7 shows the median network load for the four different implementations. CQBS significantly reduces the network load compared to non-optimized static quality implementation. On average, the network load is 19% lower when using CQBS. When comparing CQBS to the conventional algorithm and BOLA, we see overall a similar network load for the three strategies.

For quality level nine, the conventional algorithm yields

a high network load. The conventional algorithm bases the quality of future video segments on the network throughput in the near past. Given the high variation in LTE networks, past performance is not a good indicator for the future. In case, the conventional algorithm overestimated the quality and had to recover the buffer afterwards. The reason for the higher load for BOLA at quality level 10, is given earlier this section.



Figure 7. Median network load as a function of video quality.

C. Validation against real-world traces

In the second part of the evaluation, we validate our CQBS algorithm agains the real-world LTE traces that we have collected. We use the cross-validation technique, where our strategy was trained on $\frac{4}{5}$ of the traces and then tested against the remaining $\frac{1}{5}$ of the LTE traces. The results in this section are combination of five rounds, every round changing the fraction of traces that are used in the simulation. This leads up to 377 instances per setting. The DASH player streams a 8:20 minutes clip from the movie Sintel using the same encoding settings as before. The LTE channel qualitt traces have a duration of five minutes. Therefore, we repeat a trace to cover the full length of the video.

In general, the performance of CQBS and the other algorithms is similar when using the real-world traces instead of the model-based traces. Therefore, we will focus on the differences that we have observed, starting with the stability of the streams' qualities. Figure 8 shows the distribution of mean video quality for each instance. Given target quality level seven, the constant video strategies perform well, having almost all runs at the target video quality. For the conventional algorithm and BOLA the spread of average video quality is larger. With video resolutions ranging between SD (360p) and Full-HD (1080p) the differences between runs are large, indicating that reserving a fixed portion of network resources cannot provide guarantees on the video quality.

With regards to the number of quality switches, we do not observe a difference for the conventional algorithm and BOLA (the video was slightly shorter, but the number of switches is consistently lower). In contrast, our CQBS strategy produces a higher switching frequency, especially for the high target qualities. In CQBS, quality switches only occur when channel



Figure 8. Distribution of overall video quality, target video quality level seven.



Figure 9. Mean number of quality switches as a function of video quality.

quality is bad and the buffer level is too low to guarantee uninterrupted streaming. This indicates that buffers levels get lower more often because the duration and severity of low channel quality is underestimated by our model. Our model is thus not entirely accurate at low channel qualities. However, even though there is a slight increase in quality switches, CQBS still greatly outperforms the traditional algorithms.

In Figure 10 the median networks loads for the different target quality levels are shown. Compared to the model-based traces, the network loads for the real-world LTE traces show small differences. There is a slightly smaller difference in network load between the naive static strategy and CQBS. The naive static strategy seems to perform slightly more efficient given the real-world traces. Nevertheless, our CQBS strategy is able to decrease the network load by on average 15%. In terms of network load, CQBS performs slightly better than the conventional algorithm, and on average as good as BOLA.

Although BOLA performs overall best when it comes to network load, we did observe a high number of freezes when using BOLA. In Figure 11, it can be seen that BOLA has a structural problem and that freezes occur in most of the runs. In a large number of runs, the number of freezes can even be considered as very high. A high number of freezes is disastrous for the user experience, and will lead to abandonment of the video stream [6]. The other algorithms, including CQBS, do no have playback interruptions in almost all the runs. Only in



Figure 10. Median network load as a function of video quality.



Figure 11. Distribution of the number of video freezes per instance.

exceptional cases freezes might occur. This shows that even though CQBS actively reduces the buffer level, it does not compromise the continuity of the stream.

Overall, we can conclude that CQBS can maintain a constant video quality, per video stream and during the video stream. It outperforms the traditional adaptation algorithms that show fluctuations in video quality as a result of variations in LTE channel quality. Compared to high number of freezes in BOLA, CQBS is able to stream without interruptions in almost all runs. As such, CQBS provides the best experience to the user. When looking at network load, CQBS performs on par with the traditional algorithms, but outperforms the non-optimized static quality strategy.

VI. CONCLUSION & FUTURE WORK

Online video streaming is one of the biggest contributors to data consumption on mobile networks. Because of the high data volumes and the mobile networks' high throughput variations, it is a difficult practice to deliver high quality video to the end-user. Video streaming players adapt the video quality to the networks' conditions. However, the common variations in LTE channel quality can cause large fluctuations in video quality during playback, which may be perceived by users as annoying. In this paper we present CQBS, a strategy that eliminates quality switches by keeping the video quality constant, with the overall goal to improve the users' streaming experiences. By smartly adapting the video buffer to the channel quality, we were able to reduce the load on the mobile network, that would otherwise be inflicted upon when keeping the video quality constant. As such, network operators can deliver the user a better Quality of Experience without compromising their network resources.

CQBS is based on a relatively simple Markov chain that describes channel quality behavior. That method that we use to derive the optimal buffering strategy is quite elegant. At its base, the state space of our MDP only consists of two dimensions: RSRQ level and buffer level. The results show that this relatively simple model already yields good performance, and thus models the environment reasonably well. Furthermore, RSRQ and buffer can both be measured by the client device, which is important when implementing CQBS in practice.

In a practical deployment, the channel quality, video quality and buffer level have to be communicated to the mobile base station. Exchanging channel quality indicators (CQI) is already part of the LTE protocol and will continue to exists in future 5G networks. Exchanging information about video quality and buffer level is not default in DASH-based streaming. To realize our strategy we rely on DASH Aware Network Elements. These network elements coordinate DASH players and manages network resources. In our case, the DANE will be located in the mobile base station where it executes our CQBS strategy. Communication between DANE and DASH player should follow the standardized interactions as described in the Server and Network Assisted DASH standard [9], [19].

In future work, we will look into realistic heterogeneous LTE deployments and investigate the impact of heterogeneity on the performance of CQBS. Furthermore, we will profile the network load required by CQBS in more detail. In some of our simulations, CQBS used significantly less resources than the other algorithms. In other runs, CQBS required a higher network load. During a streaming session, the network load also showed some peaks. When a mobile cell is lightly loaded, there is enough resources available to withstand such a peak. In a crowded cell, it might be more beneficial to temporarily lower the video quality. Determining what is best action to take in a crowded cell is a challenging exercise that we will work on as part of future work.

REFERENCES

- [1] Sandvine, Inc, "Global internet phenomena report," Tech. Rep., 2016.
- [2] "ISO/IEC 23009-1 Information technology Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats," Tech. Rep., 2014.
- [3] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over

HTTP," in Proceedings of the Second Annual ACM Conference on Multimedia Systems. New York, NY, USA: ACM, 2011, pp. 157–168.

- [4] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Nearoptimal bitrate adaptation for online videos," in *IEEE INFOCOM* 2016 - The 35th Annual IEEE International Conference on Computer Communications, Apr. 2016, pp. 1–9.
- [5] T. Hoßfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia, "Identifying QoE optimal adaptation of HTTP adaptive streaming based on subjective studies" *Computer Networks*, vol. 81, pp. 320–332, 2015.
- on subjective studies," *Computer Networks*, vol. 81, pp. 320–332, 2015.
 [6] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang, "Understanding the Impact of Video Quality on User Engagement," *Commun. ACM*, vol. 56, no. 3, pp. 91–99, Mar. 2013.
- [7] S. Lederer. (2015, Feb.) Why YouTube & Netflix use MPEG-DASH in HTML5. [Online]. Available: https://bitmovin.com/status-mpeg-dashtoday-youtube-netflix-use-html5-beyond/
- [8] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: picking a video streaming rate is hard," in *IMC '12: Proceedings of the 2012 ACM conference on Internet measurement conference.* New York, New York, USA: "ACM Request Permissions, Nov. 2012, pp. 225–238.
- [9] E. Thomas, M. O. van Deventer, T. Stockhammer, A. C. Begen, and J. Famaey, "Enhancing MPEG DASH performance via server and network assistance," in *The Best of IET and IBC*. Institution of Engineering and Technology, 2015, pp. 48–53.
- [10] N. Bouten, J. Famaey, S. Latré, R. Huysegems, B. D. Vleeschauwer, W. V. Leekwijck, and F. D. Turck, "QoE optimization through innetwork quality adaptation for HTTP Adaptive Streaming," in 2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualization management (svm), Oct. 2012, pp. 336–342.
- [11] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo, "Design and Experimental Evaluation of Network-assisted Strategies for HTTP Adaptive Streaming," in *Proceedings of the 7th International Conference on Multimedia Systems*. New York, NY, USA: ACM, 2016, pp. 3:1–3:12.
- [12] J. W. Kleinrouweler, S. Cabrero, and P. Cesar, "Delivering Stable Highquality Video: An SDN Architecture with DASH Assisting Network Elements," in *Proceedings of the 7th International Conference on Multimedia Systems*. New York, NY, USA: ACM, 2016, pp. 4:1–4:10.
- [13] T. Wirth, Y. S a nchez, B. Holfeld, and T. Schierl, "Advanced Downlink LTE Radio Resource Management for HTTP-streaming," in *Proceedings* of the 20th ACM International Conference on Multimedia. New York, NY, USA: ACM, 2012, pp. 1037–1040.
- [14] A. E. Essaili, D. Schroeder, E. Steinbach, D. Staehle, and M. Shehada, "QoE-Based Traffic and Resource Management for Adaptive HTTP Video Delivery in LTE," *IEEE Transactions on Circuits and Systems* for Video Technology, vol. 25, no. 6, pp. 988–1001, Jun. 2015.
- [15] A. H. Zahran, J. J. Quinlan, K. K. Ramakrishnan, and C. J. Sreenan, "SAP: Stall-Aware Pacing for Improved DASH Video Experience in Cellular Networks," in *Proceedings of the 8th ACM on Multimedia Systems Conference*. New York, NY, USA: ACM, 2017, pp. 13–26.
- [16] X. Wang, E. Grinshpun, D. Faucher, and S. Sharma, "On Medium and Long Term Channel Conditions Prediction for Mobile Devices," in 2017 IEEE Wireless Communications and Networking Conference (WCNC), Mar. 2017, pp. 1–6.
- [17] H. Holma, A. Toskala, and J. Reunanen, *LTE Small Cell Optimization:* 3GPP Evolution to Release 13. John Wiley & Sons, 2016.
- [18] C. Ide, R. Falkenberg, D. Kaulbars, and C. Wietfeld, "Empirical Analysis of the Impact of LTE Downlink Channel Indicators on the Uplink Connectivity," in 2016 IEEE 83rd Vehicular Technology Conference (VTC Spring), May 2016, pp. 1–5.
- [19] "ISO/IEC 23009-5: Information technology Dynamic adaptive streaming over HTTP (DASH) - Part 5: Server and network assisted DASH (SAND)," Tech. Rep., 2017.