SmarterPhones: Anticipatory Download Scheduling for Wireless Video Streaming

Martin Dräxler*, Johannes Blobel*, Philipp Dreimann*, Stefan Valentin† and Holger Karl*
*University of Paderborn, Germany
Email: {martin.draexler, johannes.blobel, philipp.dreimann, holger.karl}@upb.de
†Bell Labs, Alcatel Lucent, Stuttgart, Germany
Email: stefan.valentin@alcatel-lucent.com

***Abstract*—Video streaming is in high demand by mobile users. In cellular networks, however, the unreliable wireless channel leads to two major problems. Poor channel states degrade video quality and interrupt the playback when a user cannot sufficiently fill its local playout buffer: *buffer underruns* occur. In contrast, good channel conditions cause common greedy buffering schemes to buffer too much data. Such *over-buffering* wastes expensive wireless channel capacity.**

Assuming that we can anticipate future data rates, we plan the quality and download time of video segments ahead. This *anticipatory download scheduling* avoids buffer underruns by downloading a large number of segments before a drop in available data rate occurs, without wasting wireless capacity by excessive buffering.

We developed a practical anticipatory scheduling algorithm for segmented video streaming protocols (e.g., HLS or MPEG DASH). Simulation results and testbed measurements show that our solution essentially eliminates playback interruptions without significantly decreasing video quality.

I. Introduction

Delivery of video content over wireless broadband networks is already widely used today and is expected to increase heavily in the upcoming years. Studies by Cisco [1] and Akamai [2] indicate that mobile data traffic will increase by a factor of 25 from 2011 to 2016 with around two-thirds of this traffic being streamed video traffic. The wireless infrastructure cannot keep up with this trend by merely increasing data rate. It is necessary to organize mobile data transmission in a better way, as also indicated by Akamai [2].

We present an approach to combine buffer control and video quality selection based on *anticipation* of wireless data rates. Our approach and the following motivation is based on the HTTP Live Streaming (HLS) protocol [3], but can also be applied to similar video streaming protocols, like MPEG DASH [4], [5].

In HLS a video is not transmitted as a continuous stream of data, but it is divided into *segments* of a certain duration and then transmitted segment by segment. These segments are downloaded via HTTP from the server and are then concatenated by the player application for playback. For example, a video of 120 s using segments of 10 s would be divided into 12 segments. This implies that for uninterrupted playback, segment $i+1$ has to be fully downloaded before segment i has been played to its end in the HLS player application. If a segment is downloaded before it is needed for playback, it is buffered at the HLS player application.

Another key feature of the HLS protocol is video quality selection: each segment can be present on the server in different quality levels. A quality level is determined by the resolution and the encoding bit rate of the video and is then identified in HLS by the resulting file size of the video segment. As our approach optimizes downloading of video segments and the file size has a direct implication on the required data rate for a download. Because of the segmented nature of HLS, a video segment is only played back if it has been fully downloaded. There are no visible artifacts or visual degradation if the video is played back and thus applying standard metrics for perceived video quality like PSNR or MOS provides limited gain. So in this paper video quality always refers to the different quality levels in HLS. As the quality of experience for the users is also impacted by the occurrence of playback interruptions caused by buffer underruns, we also evaluate the accumulated playback interruptions in our evaluation. We refer to this metric as *lateness*. Together the video quality level and the lateness indicate the quality of experience provided by our approach.

To download a segment, the player application has to decide in which quality level to download it. This is done in current HLS-compatible players like VLC or the Apple iOS and Android media players, but the selection only relies on the measurement of the current and past data rates.

In order to integrate anticipatory knowledge of future data rates into our approach we use what we call *data rate anticipation*. The idea behind this is somewhat similar to classical channel prediction used for improved scheduling decisions in mobile access networks, but our approach works on different time scales and accuracy levels. The time scales in which our approach has to work are defined by the length of the video segments, which are usually on the order of tens of seconds, in contrast to channel prediction for a few milliseconds. On the other hand, we are only interested in rough estimates of

This work was partly supported by Bell Labs, Stuttgart within the research collaboration Smarter Phones And smarter Networks (SPAN).
The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 318115.
This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

978-1-4799-5804-7/15/$31.00

achievable data rates and not precise channel quality samples, making this a simpler problem.

The idea of incorporating knowledge of future data rates into wireless video streaming already exists in research [6]–[12], which has shown that future data rates can be anticipated in practice. A recent study [13] on different prediction mechanisms for different time scales also shows such anticipation of future available data rates to be feasible with an acceptable margin of error. Thus, we do not focus on an actual implementation of *data rate anticipation*, but investigate how it can be gainfully utilized.

With the idea of anticipation we extend the default behavior in the HLS protocol by explicit buffer control and quality selection based on the anticipated data rates. The motivation for this extension is straightforward: As long as enough data rate is available in the future, the HLS video player should not download and buffer too many segments. Buffering too many segments in this case has no benefit for the user's QoE, but may have the downside of using wireless resources that could otherwise be used to benefit other users. We call this problem *over-buffering*. If there is a future decrease in available data rate, the HLS player has to download and buffer more segments in advance. If the HLS player does not download enough segments in advance the playback will stall; we call this problem *buffer underrun*.

In parallel to this decision on *when* to download segments is the decision in *which quality* to download segments. If the data rate is insufficient to download segments in a high quality, but a lower quality is available, the HLS player should switch to the lower quality to prevent a buffer underrun.

We call this combination of *when* to download each segment in *which quality* a *download schedule*. Such a download schedule is only executed on the application layer. For the physical layer schedule we assume that a normal, fair scheduler has already assigned radio resources to the users. This makes our scheduling independent from the physical layer scheduling of different wireless technologies. Additionally this allows us to perform our scheduling for each user individually, because the physical resources are already shared and we do not have to consider any resource sharing. Hence, the anticipated wireless data rates are actually achievable and effects like number of users per cell are already incorporated by the anticipation scheme.

In our previous work [14] we have presented an mixed integer quadratically constrained optimization problem (MIQCP) to create a download schedule. In Section III we introduce a new heuristic algorithm. In Section IV, we explain how our scheduling approach can be integrated into an existing system and describe how we developed a testbed implementation. We use this testbed implementation together with a simulation in Section V to evaluate our approach and to present the results. We conclude our work in Section VI.

In our previous work [15] we have already introduced the overall idea of anticipatory scheduling for segmented wireless video streaming, but did not elaborate on the heuristic algorithm and the testbed implementation.

II. Related Work

Incorporating channel anticipation into video streaming in mobile networks has been investigated in general [6], [16] and specifically for public transport scenarios [8], [9], but all without considering video quality selection, which is significantly less complex than our combined approach. Recent studies has also investigated the quality adaptation mechanism of HTTP video streaming [17], [18], but only in a *reactive* way without the combination with data rate anticipation. Anticipatory approaches based on existing data rate traces have also been presented [19], [20], but focus solely on the quality selection and do not incorporate the full download scheduling with the option to pre-buffer a variable number of segments.

The measurements in [21] and [22] illustrate the performance of HLS in mobile networks, but do not include any anticipation mechanism or cross-layer approach.

III. Heuristic Algorithm

We assume a discrete time model. Time is represented as a sequence of time slots t_i of constant length. For simplification, we further assume that the length of each time slot is equal to the playback duration of one video segment. Thus time slots and segments are unitless and can be used together in a constraint. Additionally, each video segment has to be downloaded within exactly one time slot, i.e. the download of a video segment must not be spread across multiple time slots. This implies that for an uninterrupted playback of a video, the i-th video segment has to be downloaded within time slot t_i or earlier. Downloads in a given time slot are limited by the data rate for each user in this slot. Each user is connected to at most one base station per t_i. We assume that the allocation of data rates to the users is done by an underlying, non-modifiable radio resource scheduler, limiting our scheduling approach for the download of video segments to a higher layer. The file size for each video segment, i.e the required amount of data to download, is determined by the selected video quality level. The data rate limits and the video quality levels are given in the same units.

Consistent with our existing optimization problem, the heuristic algorithm, called FILL, is an offline scheduler, which means the anticipated data rates for all time slots are known in advance and the result of the heuristic is a complete schedule for all users over a given number of time slots.

The FILL algorithm takes the available data rate for each user in each time slot as its parameter and it also iterates over all time slots to fill the buffer with video segments (independently for all users). Algorithm 1 shows this main structure. The function ANTICIPATEUSERRATES(u) returns anticipated data rates for a user for all time slots based on the underlying radio resource scheduler and channel anticipation.

The basic operation of SCHEDULESEGMENT is illustrated in Figure 1. For each time slot there are two different operations possible, depending on the available data rate in the time slot.

If there is enough data rate to download a new segment in the currently examined time slot (Algorithm 2, lines 3 and 4),

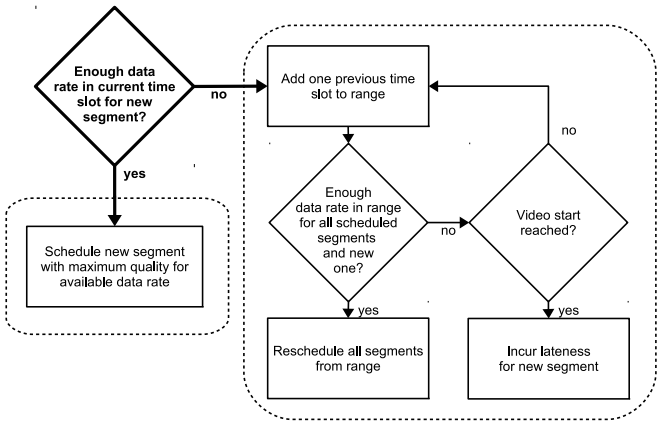


Figure 1. Flowchart for Fill Scheduler

Algorithm 1 FILLSCHEDULER(U, T, Q)

1: // users U, times T, qualities Q
2: **for all** $u \in U$ **do** // schedule all users
3: $C \leftarrow$ ANTICIPATEUSERRATES(u) // from channel anticipation
4: $s \leftarrow 0$ // initialize counter for scheduled segments
5: **for all** $t \in [0..|T|]$ **do** // schedule all time slots/segments
6: $s \leftarrow s +$ SCHEDULESEGMENT(u, t, s, Q, C)
7: **end for**
8: **end for**

the FILL algorithm will just schedule this video segment at maximum possible quality. This behavior ensures a minimum number of segments in the buffer as long as there is no need for buffering more segments for future time slots with insufficient data rate.

If, during the iteration, the anticipated data rate in some time slot t does not suffice to download a new video segment (Algorithm 2, lines 6 to 22), even at the lowest video quality level, the FILL algorithm has to change the schedule for one or more *previous* time slots to download and buffer a video segment before time slot t with insufficient data rate. This

Algorithm 2 SCHEDULESEGMENT(u, t, s, Q, C)

1: $q \leftarrow$ GETBESTQUALITY$(Q, C[t])$
2: **if** $q \neq$ **false then** // enough capacity in current time slot for new segment?
3: SCHEDULE(u, s, t, q) // schedule new segment with maximum quality for available data rate
4: **return** 1
5: **else** // even lowest quality not feasible in time slot t
6: **for all** $g \in [t..0]$ **do**
7: // enough capacity in range $[g..t]$ for all scheduled segments and new one?
8: **if** GETBESTQUALITYRANGE$(Q, t - g + 1, \sum_{i=g}^{t} C[i]) \neq$ **false then** // going back to g provides enough data rate
9: $q \leftarrow$ GETBESTQUALITYRANGE$(Q, t - g + 1, C[g...t])$
10: $p \leftarrow 0$
11: **for all** $r \in [g..t]$ **do** // reschedule all segments from range
12: $n \leftarrow$ GETSEGMENTSFORQUALITY$(q, C[r])$
13: **for all** $v \in [(g + p)..(g + p + n)]$ **do**
14: SCHEDULE(u, v, r, q)
15: **end for**
16: $p \leftarrow p + n$
17: **end for**
18: **return** 1
19: **end if**
20: **end for** // video start reached
21: **return** 0 // incur lateness for new segment
22: **end if**

part of the algorithm, as outlined in Algorithm 2, requires the definition of the following helper functions:

- GETBESTQUALITY(Q, c)
  Returns the best downloadable quality (out of Q) for a segment with anticipated available data rate c, or FALSE if there is not enough data rate even for the lowest quality
- GETBESTQUALITYRANGE(Q, n, c)
  Returns the best possible quality (out of Q) in which n segments can be downloaded with anticipated available data rate c, or FALSE if there is not enough data rate to download even in the lowest quality
- GETSEGMENTSFORQUALITY(q, c)
  Returns the number of downloadable segments with quality q and available data rate c
- SCHEDULE(u, s, t, q)
  Schedule the download of segment s for user u at time t with quality q

These functions can be easily implemented and their precise implementation is omitted in this paper to improve the readability of the algorithm.

From time slot t where a download of a full segment was not possible, the algorithm goes back time slot by time slot. In these previous time slots, it downgrades the video quality of the segments, freeing up capacity to enable the download of the segment that has to be playout out in time slot t. It can push up the scheduled download times of earlier segments in order to fit more segments into time slots. Once a range of time slots is found where all segments including the one to be played out in time slot t fit in (at reduced quality), the computation of the schedule up to time slot t is complete. This schedule is then the basis to plan the download for the segment for time slot $t + 1$ in the next iteration.

IV. SYSTEM INTEGRATION

In this section we discuss how the previously introduced algorithms can be integrated into a real system, using existing tools and extending existing protocols with backwards-compatible extensions as needed. We first explain the system architecture in Section IV-A and then in Section IV-B implementation details and adjustments to the HLS protocol necessary to use the scheduling algorithms. The concrete testbed implementation which we used to verify our simulation results is described afterwards in Section IV-C

A. Architecture

For our implementation use a modified video player on the UEs, so we can fully control the buffer and we can analyze the performance of the system. Our implementation supports arbitrary content providers (in our tests we used our own video source to eliminate external influences).

To implement our schedulers we assume an overall architecture as depicted in Figure 2. This architecture does not require any changes to current cellular radio interfaces and networks (RANs) and can be implemented in a cellular network as well as in a wireless LAN scenario, since the scheduler is implemented in higher layers. It also does not require any

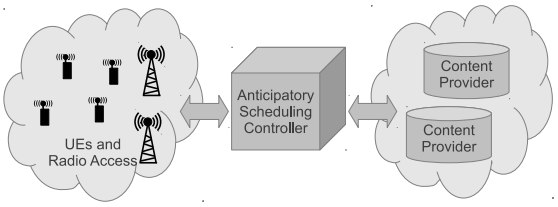


Figure 2. System Architecture

changes to the content provider since all scheduling decisions and the schedule is enforced in the *Anticipatory Scheduling Controller*.

The *Anticipatory Scheduling Controller*, as the central entity in this architecture, intercepts the requests from the UEs to the content providers. It can then perform the buffer control and quality selection with the following three steps:

1) Intercept the video request from the UE and analyze it (video data rates, available variants)
2) Calculate schedule based on video data and anticipatory information on future data rates
3) Control the buffering behavior of the UE according to the schedule

To do so, the *Anticipatory Scheduling Controller* could be configured as an HTTP proxy (as HLS video requests are transported via HTTP). This could be enforced in cellular networks or be done voluntarily by the users. Both operators and users have incentives to do so (less load on the network, better QoE for the users).

B. Protocol Extension

We concentrated on HLS (HTTP Live Streaming) [3] as the streaming protocol for our implementation. It is available in the stock media players on Android and Apple iOS and is also available as an open-source implementation in the VLC player. To stream a video using HLS, regardless of our extension, the video has to be encoded properly. This encoding is a CPU-intensive, one-time task. The video input is cut into independently playable segments with the same playback duration. URLs to these segments are then added to a playlist. An example of such a normal HLS playlist is shown in Figure 3.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:10
#EXTINF:10,
http://hostname/high/001.ts
#EXTINF:10,
http://hostname/high/002.ts
#EXTINF:10,
http://hostname/high/003.ts
#EXTINF:10,
http://hostname/high/004.ts
#EXTINF:10,
http://hostname/high/005.ts
#EXTINF:10,
http://hostname/high/006.ts
#EXT-X-ENDLIST
```

Figure 3. Single quality HLS example with high quality segments, each 10 seconds long (playlist is split into two columns)

HLS streams can provide multiple qualities of the same video. Each quality can be encoded using a different codec, bitrate, or resolution. HLS players can switch between different qualities because all segments have equal length and are independently playable. A separate playlist is created for each quality and additionally a master playlist with links to all quality playlists is used. An example of a master playlist with three qualities is shown in Figure 4. The master playlist contains parameters for each quality to enable HLS players to select the most appropriate one. We use the `BANDWIDTH` parameter, given as a data rate in bit/s, for each quality for this paper. The created playlists and segments can then be placed on an HTTP server. An HLS player only needs the URL to the HLS master playlist. From there, all qualities and their segments are accessible.

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=1000000
http://hostname/low/hls.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1500000
http://hostname/med/hls.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=3000000
http://hostname/high/hls.m3u8
```

Figure 4. Multi-quality master playlist with three qualities

To control the buffering behavior of HLS players, we need a method to pass messages to them. HLS players have no interface to receive control data besides playlists and segments via their own HTTP-GET requests. We intercept the requests for playlists and modify the replies in the anticipatory scheduling controller.

The anticipatory scheduling controller is aware of the schedule but also needs a means of inserting buffering instructions in the playlists. Thus, we introduce two new tags to HLS playlists: `BUFFERSIZE` and `REFRESH`. Both are defined as natural numbers including 0. These new tags are backwards compatible because the HLS standard instructs players to ignore tags which they do not recognize [3].

`BUFFERSIZE` sets the size of the HLS player buffer to the given value. Up to this amount of segments, the player will just greedily try to download more segments. If there are more segments in the buffer than instructed, the buffer content is played, and no downloaded segments are discarded. As soon as there are fewer segments in the buffer than the given limit, the HLS player downloads additional segments to fill the buffer.

The `REFRESH` parameter instructs the HLS player to refresh the playlists every `REFRESH` seconds. This will then update the `BUFFERSIZE` and `REFRESH` parameters. We suggest to set `REFRESH` to the playback length of a segment, thus after playing one segment the HLS player will update its buffering parameters.

The two parameters together solve the over-buffering and buffer underrun problem by precisely adapting the HLS player buffer size according to the schedule. This indirectly influences *when* an HLS player can download a segment.

Another property of an HLS stream that the scheduling algorithm needs to decide is *which quality* to download. In the case of multi-variant HLS streams, the player would try to download the segments in the quality it prefers by doing its own local measurements. But the schedules also include the HLS video quality for each segment, selected from the available HLS qualities.

Every time the HLS player requests an HLS master playlist

the anticipatory scheduling controller downloads the playlists of the scheduled qualities and creates a single variant playlist out of the multi-quality playlist. As shown in Figure 5, segments from different qualities are being selected and placed in a new single-quality playlist. Only the joined (single-quality) playlist is then returned to the HLS player. The decision which quality to download is hereby made by the anticipatory scheduling controller and not by the player anymore. The joined playlist contains the REFRESH and BUFFERSIZE parameters. Each time a player refreshes an HLS playlist, it can receive a different value for the BUFFERSIZE parameter.

```
#EXTM3U                                  | #EXTINF:10,
#EXT-X-VERSION:3                         | http://hostname/med/003.ts
#EXT-X-TARGETDURATION:10                 | #EXTINF:10,
#EXT-X-BUFFERSIZE: 2                     | http://hostname/med/004.ts
#EXT-X-REFRESH:10                        | #EXTINF:10,
#EXTINF:10,                              | http://hostname/low/005.ts
http://hostname/med/001.ts               | #EXTINF:10,
#EXTINF:10,                              | http://hostname/med/006.ts
http://hostname/med/002.ts               | #EXT-X-ENDLIST
```

Figure 5. Joined playlist with REFRESH and BUFFERSIZE extensions (BUFFERSIZE set to 2 and REFRESH set to 10, playlist is split into two columns)

Through both mechanisms, the buffer size (*when* to download) and preselection of variants (*which quality* to download) can be controlled. Thus, anticipatory buffering and variant selection based on the previously described algorithms can be performed by simply extending the HLS protocol with two small extensions to the playlist parameters.

C. Testbed

In order to analyze our algorithms and to test our HLS protocol extension in a real system, we developed a testbed that allows us to run extensive tests with real hardware and compare the results of these tests with our simulations. We describe our testbed setup here and will present the simulation and testbed measurement results in Section V.

The testbed is based on the general architecture explained before. The UEs are smartphones and tablets with a customized Android OS and a modified VLC video player. Our modifications enable VLC to parse the additional playlist parameters and adapt its buffer size accordingly. It also outputs extended information about the buffer size and the downloaded segments which is used for our measurements. The customizations to the Android OS are only necessary to control the behavior of the smartphones and tablets and have no influence on the video streaming itself.

The radio access in the testbed is implemented with 802.11g wireless LAN [23] without any modifications and four access points. As explained before, scheduling only happens on the application layer, thus changes to the wireless MAC are not necessary. The access points are normal PCs with wireless LAN cards and Linux with hostapd running on them.

A fifth PC serves as central control and measurement unit and as a host for running the anticipatory scheduling controller. All phones are connected to this PC via USB and are controlled with the Android debug bridge (ADB); the access points are controlled via an SSH connection. With the ADB we execute arbitrary shell commands on the phones and emulate simple user interaction like starting or stopping a video stream. No data is transmitted via USB; it only serves to make experiments repeatable. The resulting overall testbed architecture and setup can be seen in Figure 6.

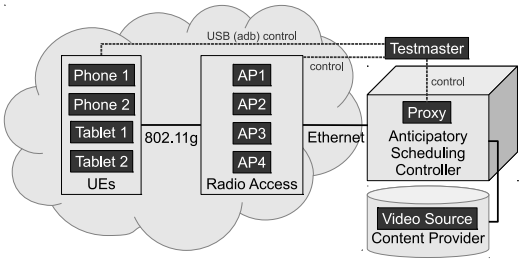


Figure 6. Testbed Architecture

For the HLS video stream content we used the publicly available movie “Tears Of Steal” (http://www.tearsofsteel.org/) which we converted using the VLC framework. The segments and playlists are served by an unmodified Apache webserver.

The anticipatory scheduling controller, which intercepts and modifies the playlist requests from the UEs, is implemented as a transparent HTTP proxy using the Python framework Twisted (http://twistedmatrix.com). The access points redirect all traffic coming from the UEs to the proxy thus it is not necessary to change any preferences on the UEs.

We wanted to be able to run a lot of repeatable and comparable tests, which is why the movement of the UEs is emulated and not done physically. Movement emulation works by limiting the link speed and enforcing handovers between access points. We achieve this by using standard traffic shaping capabilities of Linux on the access points and on the phone. From a predefined scenario we get the data rate for every UE and base station per time slot. These values are then set as speed limits on our access points at the corresponding time. Handover events between the access points are also precalculated from the scenario and then triggered on the phones. With this setup we can run tests without the need to physically move the UEs.

We automatically start the video stream via the ADB connection to the phones and collect information about the streaming (i.e. *when* a segment has been actually loaded in *which quality*). The results returned by the testbed runs are in the same format as the simulation results and allow a direct comparison.

V. Simulation and Testbed Results

In this section we present both simulation results and results from measurements with the previously described testbed. For the simulation we use our own Python implementation. Before presenting the results we define the evaluation scenarios.

In addition to the optimization problem (MIQCP) and Fill we also include two greedy scheduling algorithms, QualityFirst and BufferFirst, that represent the state-of-the-art behavior of a HLS video player. Both algorithms are also described in our previous work [14] and greedily fill a fixed-size segment

buffer with two different strategies: QualityFirst schedules segments in the highest possible quality before filling the whole buffer. BufferFirst schedules filling the whole buffer before switching to a higher quality. Neither strategy uses any form of anticipation of future data rates.

A. Scenario

The basic structure for the evaluation scenario, for both simulation and testbed measurements, is a line of base stations with the users moving through the scenario from the first base station to the last base station as illustrated in Figure 7. To reduce the available data rate and to create the need for buffering, we remove cells from the scenario, as illustrated with base stations B and D. The more cells we remove, the more gaps without any available data rate occur and the more segments have to be buffered to avoid playback interruptions. The users all move as a group from the first base station to the last base station (e.g., a public transport scenario). Apart from the pattern in which the base stations are removed, the scenario parameters for the simulation and the testbed measurements are the same.

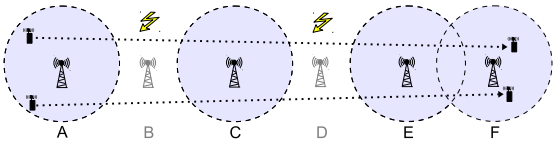


Figure 7. Scenario

The wireless radio is modeled according to 3GPP Long Term Evolution (LTE) [24]. The base stations are placed equidistantly with an inter-site distance of 1500 meters, which is slightly larger than a normal urban scenario in order to augment the effects resulting from removing cells. We consider four active users in the scenario because the testbed setup only contains four devices and we want to maintain comparability between the simulation and the testbed measurements.

The path loss in dB between the base stations and the users is obtained by $128.1 + 37.6 \cdot \log_{10}(d) + S_{\ln}$ [24], where d represents the distance between the base station and the user in kilometers and $S_{\ln}$ is a normal random variable with zero mean and standard deviation of 10 dB to model slow fading.

For the channel capacity we assume an asymptotically error-free communication channel, modeled by the Shannon equation with the following parameters: 10 MHz bandwidth, 46 dBm transmission power, isotropic antennas with 0 dB gain, -174 dBm/Hz noise power spectral density and -149 dBm/Hz average interference. The maximum data rate for a base station is limited to 30 Mbit/s to account for the small number of users in the scenario. The allocation of data rates to the users in each time slot is up to a wireless resource scheduler, which is in our case a simple proportional fair scheduler.

The maximum buffer size for the greedy scheduling algorithms is set to 3 segments, which corresponds to the default setting for VLC on Android.

The video quality levels and the resulting required data rates are taken from the test video we generated from the clip "Tears of Steel". The resulting segment sizes for the three video quality levels are 1.77 MB (low), 3.69 MB (medium) and 4.51 MB (high). As the real file size of all segments varies slightly by a few hundred kilobytes due to the video encoding, we use the maximum size over all generated video segments in one video quality level as the parameter for the scheduling algorithms. We use a segment length of 10 seconds, corresponding to the recommended value in the HLS standard.

1) Simulation Scenario: For the simulation scenario we are not limited to the number of physical devices we have in the testbed. Thus we use a total of 44 base stations and a video length of 44 segments.

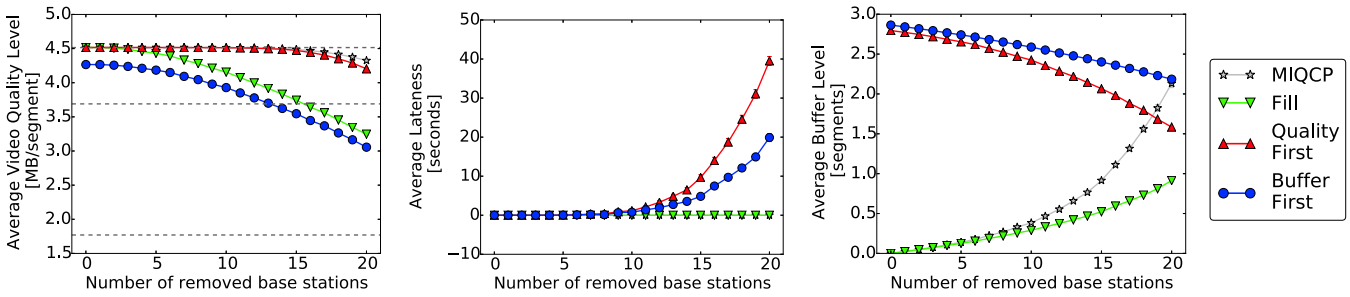
To induce the need for buffering we randomly remove base stations from the scenario. The number of removed base stations varies from 0 to 20, which means that in the worst case half of all base stations are removed. The removed base stations are selected uniformly, whereas the first and last 2 of the 44 base stations are never removed to avoid side effects.

2) Testbed Scenario: In the testbed, which we described in Section IV-C, the scenario is limited by the number of physical devices in the testbed. We have again 4 users, the phones and tablets in the testbed, but in contrast to the simulation only 4 base stations. The base stations are again arranged in a line but with only one fixed gap without any available data rate in the middle .In order to vary the need for buffering we perform measurements with a gap equal to the range of 2 and 4 base stations.

B. Results

For both the simulation and the testbed measurements, we evaluate three different metrics: the average downloaded video quality level in MB per segment, the lateness averaged over all users in seconds and the average buffer fill level in segments. All plots are based on multiple simulation or testbed runs and show confidence intervals at 95% confidence level, small intervals might be covered by the plot markers.

1) Simulation Results: The simulation results for the average video quality are shown in Figure 8a. The dashed lines indicate the reference value of the high and medium video quality levels. MIQCP delivers the overall highest video quality level, which decreases only slightly once more than 10 base stations are removed from the scenario. This indicates that MIQCP can fully exploit the available data rate in order to deliver and buffer high quality segments whenever possible. The QUALITYFIRST algorithm delivers the overall second highest video quality level, which is only slightly less than the one from the MIQCP. This corresponds to the expected behavior of the greedy algorithm. The BUFFERFIRST algorithm exhibits the opposite behavior and delivers the overall lowest video quality level, which also corresponds to the expected behavior. The FILL algorithm provides the same high video quality level as the MIQCP when only a small number of base stations is removed and enough data rate is available. When more base stations are removed the



(a) Average Quality (b) Average Lateness (c) Average Buffering

Figure 8. Simulation Results

delivered video quality level from the FILL scheduler decreases, but is still higher compared to the BUFFERFIRST algorithm.

Figure 8b shows the results for the average lateness over all users in the simulation. MIQCP and the FILL algorithm are able to prevent any lateness. For both the QUALITYFIRST and BUFFERFIRST algorithms lateness increases when more than 10 base stations are removed. Because of the objective to download segments in higher quality levels instead of buffering more segments, the QUALITYFIRST algorithm incurs the highest lateness.

The simulation results for the average buffer fill level are shown in Figure 8c. Both greedy scheduling algorithms always try to fill their buffer up to the maximum buffer level of 3 segments. Because the greedy scheduling algorithms have no mechanism to reduce buffer usage, the buffer levels only decrease when there is not enough available data rate to fill the buffer entirely as more base stations are removed from the scenario. The MIQCP and the FILL scheduler are designed to minimize buffer usage where possible, thus both start off with very little buffering and only increase the buffer usage as more base stations are removed from the scenario. After removing more than 10 base stations from the scenario the MIQCP uses more buffer space than the FILL algorithm. This is caused by the preference of the MIQCP objective function to download segments with a higher video quality level before minimizing the buffer level. The FILL algorithm, on the other hand, will switch to lower video quality levels before buffering more segments instead.

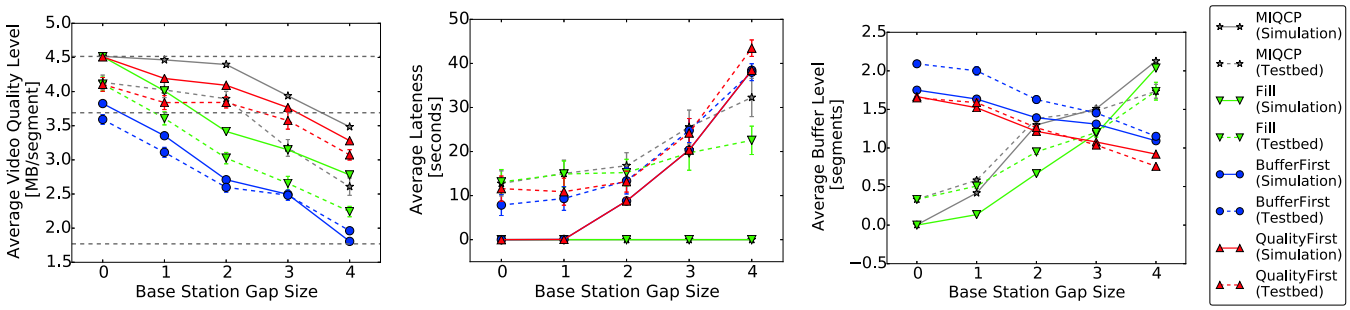
2) Testbed Measurements: The plots in Figure 9 show a comparison between simulation results with the testbed scenario and the measurements obtained from the testbed. The results from the simulation are plotted with a solid line and the testbed measurements with a dashed line, both using the same markers to distinguish between the schedulers.

Ideally, the simulation results and the testbed measurements should be identical. Differences in the results are due to the following effects, which are present in the testbed but not considered in the simulation:

- *Continuous time & rounding effects*
  The simulation is based on a discrete time model with time slots, whereas the testbed runs in real time. In order to compare the simulation and testbed results the measurements are converted to discrete time. This, for example, implies that a segment that is actually downloaded after 61 seconds, but should have been downloaded at or before 60 seconds, is treated as equally late as a segment that is downloaded after 69 seconds.
- *Network protocol side effects*
  The simulation does not consider underlying network protocols for the transport of the HLS segments. In contrast to that the testbed uses real HLS over TCP/IP over 802.11g wireless LAN with its own wireless resource scheduler. We are only sure that the data rate limits we use in the calculation of the schedules are not exceeded, but we cannot ensure that they are actually fully achieved in the testbed. Both TCP congestion control and the wireless resource scheduler can influence the actual data rates in the testbed, which result in longer segment downloads, which are then treated as late.
- *Video player issues*
  In case the video player in the testbed encounters issues while decoding the video, the timing between the downloads from the player and the schedule can be disturbed. For example, if VLC decides to skip frames from the video the playback runs ahead of the calculated schedule, and subsequent segments are needed for playback before their download was scheduled to be complete. This can happen because the video player runs on a real Android device and has to share the CPU with the system and background processes.

Despite all these complicating factors, the measurement results for the average video quality in Figure 9a show only little differences between the simulation and testbed. This indicates that our mechanisms for quality selection work in our testbed implementation as well as expected based on the simulation.

Figure 9b shows the results for the average lateness in the testbed. The measurement results for the greedy schedulers again show only a small difference compared to the simulation, but the measurement results for the MIQCP and the FILL scheduler show a significantly higher lateness for the testbed. We discovered that this is due to the buffer minimization in



(a) Quality (b) Lateness (c) Buffer usage

Figure 9. Testbed Measurement Results (dashed lines) compared to Simulation Results (solid lines)

these two schedulers: being forced to use a low buffer level makes the video player more susceptible to the timing side effects we previously described.

The results for the average buffer fill level in Figure 9c again show only a small difference between the simulation results and the testbed measurements.

Taking into account the side effects from the testbed setup, we can sum up that our testbed implementation of the anticipatory scheduling works as forecasted by the simulation results. This agreement of results between two different and independent evaluation methodologies lends considerable evidence to the utility and feasibility of our proposed anticipatory scheduling scheme.

VI. Conclusion and Future Work

We have presented an approach to efficiently exploit anticipatory knowledge of future wireless data rate for wireless video streaming. Our simulation results and testbed measurements consistently show that adapting buffer and video quality to the anticipated wireless data rate essentially eliminates playback interruptions while maintaining a high video quality level.

The presented results only show the performance of offline schedulers, where the data rates are anticipated for all future time slots. In our future work we will also focus on online schedulers with limited anticipatory knowledge.

References

[1] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20132018." [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf

[2] W. Law, "Delivering Over The Top Video at Scale - Akamai at OTTCon 2013," 2013.

[3] R. Pantos, W. May, and Apple Inc., "HTTP Live Streaming," http://tools.ietf.org/html/draft-pantos-http-live-streaming-11, April 2013.

[4] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over HTTP dataset," in *Proc. of the 3rd Multimedia Systems Conf.*, 2012.

[5] I. Sodagar, "The MPEG-DASH Standard for Multimedia Streaming Over the Internet," *MultiMedia, IEEE*, vol. 18, no. 4, Apr. 2011.

[6] Z. Lu and G. de Veciana, "Optimizing Stored Video Delivery For Mobile Networks: The Value of Knowing the Future," in *Proc. of the IEEE Int. Conf. on Comp. Comm. (INFOCOM)*, 2013.

[7] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Video streaming using a location-based bandwidth-lookup service for bitrate planning." *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 8, no. 3, p. 24, 2012.

[8] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: analysis and applications," in *Proc. of the 4th ACM Multimedia Sys. Conf.*, 2013.

[9] J. Yao, S. S. Kanhere, and M. Hassan, "Mobile Broadband Performance Measured from High-Speed Regional Trains." in *Proc. of the IEEE Vehicular Technology Conference (VTC Fall)*, 2011.

[10] ——, "An empirical study of bandwidth predictability in mobile computing." in *Proc. of the 3rd ACM Int. Workshop on Wireless network testbeds, experimental evaluation and characterization - WiNTECH*, 2008.

[11] ——, "Improving QoS in High-Speed Mobility Using Bandwidth Maps." *IEEE Trans. Mob. Comput.*, vol. 11, no. 4, pp. 603–617, 2012.

[12] J. Fardous and S. S. Kanhere, "On the use of location window in geo-intelligent HTTP adaptive video streaming." in *Proc. of the IEEE Int. Conf. on Networks (ICON)*, 2012.

[13] N. Bui, F. Michelinakis, and J. Widmer, "A Model for Throughput Prediction for Mobile Users," in *Proc. of European Wireless 2014*, 2014.

[14] M. Dräxler and H. Karl, "Cross-Layer Scheduling for Multi-Quality Video Streaming in Cellular Wireless Networks," in *Proc. of Int. Wireless Communications & Mobile Computing Conf. (IWCMC)*, 2013.

[15] ——, "SmarterPhones: Anticipatory Download Scheduling for Segmented Wireless Video Streaming," in *1st KuVS Workshop on Anticipatory Networks*, 2014.

[16] S. Khan, Y. Peng, E. Steinbach, M. Sgroi, and W. Kellerer, "Application-driven cross-layer optimization for video streaming over wireless networks," *IEEE Comm. Magazine*, vol. 44, no. 1, pp. 122–130, 2006.

[17] T.-Y. Huang, R. Johari, and N. McKeown, "Downton abbey without the hiccups: Buffer-based rate adaptation for http video streaming," in *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, 2013.

[18] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran, "Probe and adapt: Rate adaptation for http video streaming at scale," *Selected Areas in Communications, IEEE Journal on*, vol. 32, no. 4, pp. 719–733, April 2014.

[19] J. Hao, R. Zimmermann, and H. Ma, "Gtube: Geo-predictive video streaming over http in mobile environments," in *Proceedings of the 5th ACM Multimedia Systems Conference*, 2014.

[20] A. Bokani, M. Hassan, and S. Kanhere, "Http-based adaptive streaming for mobile clients using markov decision process," in *Packet Video Workshop (PV), 2013 20th International*, 2013.

[21] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz, "A comparison of quality scheduling in commercial adaptive HTTP streaming solutions on a 3G network," in *Proc. of the 4th Workshop on Mobile Video*, 2012.

[22] C. Müller, S. Lederer, and C. Timmerer, "An evaluation of dynamic adaptive streaming over HTTP in vehicular environments," in *Proc. of the 4th Workshop on Mobile Video*, 2012.

[23] "IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, 2012.

[24] 3GPP, "Further advancements for E-UTRA physical layer aspects," Tech. Rep. 36.814 V9.0.0, Mar. 2009.