Signal Processing: *Image Communication* ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Using bandwidth aggregation to improve the performance of quality-adaptive streaming

Kristian Evensen [a,*], Dominik Kaspar [a], Carsten Griwodz [a,b], Pål Halvorsen [a,b], Audun F. Hansen [a], Paal Engelstad [c]

[a] *Simula Research Laboratory, Norway*
[b] *Department of Informatics, University of Oslo, Norway*
[c] *Telenor ASA, Norway*

ARTICLE INFO

Keywords:
Multihoming
Bandwidth aggregation
Video streaming
HTTP
Quality adaptive streaming

ABSTRACT

Devices capable of connecting to multiple, overlapping networks simultaneously is becoming increasingly common. For example, most laptops are equipped with LAN- and WLAN-interface, and smart phones can typically connect to both WLANs and 3G mobile networks. At the same time, streaming high-quality video is becoming increasingly popular. However, due to bandwidth limitations or the unreliable and unpredictable nature of some types of networks, streaming video can be subject to frequent periods of rebuffering and characterized by a low picture quality.

In this paper, we present a multilink extension to the data retrieval part of the DAVVI adaptive, segmented video streaming system. DAVVI implements the same core functionality as the MPEG DASH standard. It uses HTTP to retrieve data, segments video, provides clients with a description of the content, and allows clients to switch quality during playback. Any DAVVI-data retrieval extensions can also be implemented in a DASH-solution.

The multilink-enabled DAVVI client divides video segments into smaller subsegments, which are requested over multiple interfaces simultaneously. The size of each subsegment is dynamic and calculated on the fly, based on the throughput of the different links. This is an improvement over our earlier subsegment approach, which divided segments into fixed size subsegments. The quality of the video is adapted based on the measured, aggregated throughput. Both the static and the dynamic subsegment approaches were evaluated with on-demand streaming and quasi-live streaming. The new subsegment approach reduces the number of playback interruptions and improves video quality significantly for all cases where the earlier approach struggled. Otherwise, they show similar performance.

1. Introduction

Streaming high-quality video is rapidly increasing in popularity. Video aggregation sites, like YouTube and Vimeo, serve millions of videos every day, various events are broadcasted live over the Internet and large investments are made in video-on-demand services. One example is Hulu,[1] which is backed by over 225 content companies and allow users to legally stream popular TV-shows like Lost, Glee, and America's Got Talent. This paper is an extended version of our MMSYS 2011-paper "Improving the Performance of Quality-Adaptive Video Streaming over Multiple Heterogeneous Access Networks".

* Corresponding author. Tel.: +47 98062414.
E-mail address: kristrev@simula.no (K. Evensen).

[1] http://www.hulu.com/about

0923-5965/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2011.10.007

However, high-quality video has a high bandwidth requirement. For example, the bitrate of HD-video when streaming is several MB/s. This might not be a problem in areas with a highly developed broadband infrastructure, but a single average home connection to the Internet might not be able to support this quality. For example, the average broadband connection in the United States is about 4 MB/s [1]. Due to bandwidth limitations or the unreliable and unpredictable nature of some types of networks, for example WLAN and 3G (HSDPA), streaming video can be subject to frequent periods of rebuffering, characterized by a low picture quality and playback interruptions.

Today, devices capable of connecting to multiple, overlapping networks simultaneously are common. For example, most laptops are equipped with LAN- and WLAN-interface, and smart phones can often connect to both WLANs and 3G-networks. One way to alleviate the bandwidth problem is to increase the available bandwidth by aggregating multiple physical links into one logical link. Adaptive, segmented streaming solutions, like DASH, divide videos into segments, and these can be requested and sent over independent links simultaneously, achieving bandwidth aggregation. Also, the same segment is encoded at multiple quality levels (bitrates). This allows clients to adapt the requested video quality according to the available resources, for example to ensure a smooth playback.

We have previously developed and presented a client-side request scheduler that retrieves video segments in several encodings over multiple heterogeneous network interfaces simultaneously [2]. To improve performance even further, the segments are divided into smaller logical subsegments, and the request scheduler performed well in our experiments. It reduced the number of playback interruptions and increased the average video quality significantly. However, this subsegment approach has a weakness—segments are divided into fixed-sized subsegments which, in combination with limited receive buffers, have a significant effect on multilink-performance. Unless the buffer is large enough to compensate for the link heterogeneity, this static approach is unable to reach maximum performance. Increasing the size of the receive buffer alleviates the problem. However, it might not be acceptable, desirable or even possible with a larger buffer, as it adds delay and requires more memory.

In this paper, we present an improved subsegment approach. Subsegment sizes are dynamic and calculated on the fly, based on the links' performance. By doing this, the request scheduler avoids idle periods by allocating the ideal amount of data (at that time) to each link. The request scheduler and both subsegment approaches were implemented as extensions to the DAVVI [3] streaming platform, which offers the same core functionality as DASH [4,5]. Both DAVVI and DASH-based solutions encode the same segments at multiple bitrates, provide the client with a description of the content and allow clients to switch quality during playback. Even though DAVVI's equivalent of DASH's Media Presentation Description (MPD) is structured differently, they both contain enough information to support the same quality adaptation and data retrieval techniques. A client will first select the appropriate quality, based on for example the measured throughput and the amount of buffered data, and then request the selected segment using HTTP. In other words, any modifications to the data retrieval part of DAVVI can also be applied to DASH.

The two subsegment approaches were evaluated with on-demand streaming and live streaming with and without buffering, in a controlled network environment and with real world wireless links. In the context of this paper, live/liveness is defined as how much the stream lags behind the no-delay broadcast. The dynamic subsegment approach significantly reduces the number of playback interruptions, and improves the video quality when multiple links are used. When the buffer is large enough to compensate for the link heterogeneity, both the old and the new subsegment approaches show similar performance. When buffers are small, our new solution achieves a higher video quality.

The rest of the paper is organized as follows. Section 2 contains a presentation of related work, while Section 3 describes DAVVI, more on how it compares to DASH and our multilink modifications. Our testbed setup is introduced in Section 4, and the results from our experiments are discussed in Section 5. Finally, we give the conclusion and prospects for future work in Section 6.

2. Related work

HTTP is currently one of the, if not the, most common protocol used to stream video through the Internet, and multi-quality encoding and file segmentation is a popular way to allow quality adaptation and increase performance. By picking the quality most suited to the current link performance, a smoother playback can be achieved. Also, file segmentation allows content providers to build more scalable services that offer a better user experience due to increased capacity. Commercial examples of HTTP-based streaming solutions built upon segmentation of the original content, include Move Networks [6], Apple's QuickTime Streaming Server [7] and Microsoft's Smooth Streaming [8]. The goal of MPEG DASH is to provide a standardized alternative to these proprietary solutions.

Picking the most appropriate server is a non-trivial problem that has been studied extensively. Parallel access schemes, like those presented in [9] and [10], try to reduce the load on congested servers by automatically switching to other servers for further segment requests. These parallel access schemes assume that excessive server load or network congestion create the throughput bottleneck. We assume that the bottleneck lies somewhere in the access network. However, the scheduling problem is similar—either the client or server has more available bandwidth than the other party can utilize.

Parallel access schemes are not suitable for achieving live or quasi-live streaming (sometimes referred to as "progressive download"), as they have no notion of deadlines. Also, the additional complexity introduced by automatically adapting the video quality is not solved by these parallel access schemes. Still, with some modifications, the techniques developed within the field of parallel

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

access can be applied to multilink streaming. Our earlier subsegment approach was inspired by the work done in [11], where the authors divide a complete file into smaller, fixed-size subsegments. The new, dynamic subsegment approach uses some of the ideas found in [12], most notably using the current throughput to calculate the size of the subsegments.

Although our solution can be extended to support multiple servers, our current research focuses on client-based performance improvements of using multiple network interfaces simultaneously. Wang et al. pursued a similar goal in [13], where the server streams video over multiple TCP connections to a client. However, such push-based solutions have limited knowledge about the client-side connectivity, and introduce a significant delay before detecting if a client's interface has gone down or a device has lost the connection with its current network. Also, push-based solutions, such as [14], cannot easily be extended to support multiple servers. Since we assume that the bottleneck is in the access network, we favor a pull-based scheme, allowing the client to adjust the quality and subsegment-request schedule.

3. System components

In many cases, for example with wireless networks, a single link is often insufficient due to the bandwidth requirements of streaming high quality video. To show how multiple independent links can be used to achieve a higher video quality, we extended the DAVVI streaming system [3] with support for more than a single network interface. This section describes DAVVI, the improvements we made to the data delivery subsystem and explains how these improvements would fit and could be implemented in a DASH access client.

3.1. Video streaming

DAVVI is an HTTP-based streaming system where each video is divided into fixed length, independent (closed-GOP) segments[2] with constant duration (2 s). A video is encoded in multiple qualities (bitrates), and the constant duration of the segments limits the liveness of a stream—at least one segment must be ready and received by the client before playback can start.

DAVVI stores video segments on regular web servers. A dedicated streaming server is not needed, the video segments are retrieved using normal HTTP GET-requests. Because no additional feedback is provided by the server and because it is the client that monitors the available resources, the client is responsible for prefetching, buffering, and adapting video quality. The quality can be changed whenever a new segment is requested, but the user can not see the change immediately. In our case, each segment contains 2 s of video, which has been shown to be a good segment length. According to the work done in [15], changing video quality more frequently than every 1–2 s annoys the user. However, the 2 s segment length is a limit imposed by DAVVI, in our future work, we plan to look at how the duration of a segment affects the subsegment approaches and thereby performance. For example, one possibility would be to use H.264 SVC-encoding and allow changing quality immediately, but then forbid a new change within 1 s.

For this paper, we look at three types of streaming, on-demand streaming, live streaming with buffering and live streaming without buffering. *On-demand streaming* is the most common type of streaming and is used as our base case. It assumes "infinite" receive buffers and is only limited by network bandwidth. Because the entire video is available in advance, segments are requested as soon as there is room in the receive buffer. We use an alternative encoding and linear download, so we do not have the common concept of a base layer that could be downloaded first with quality improvements as time permits. On-demand streaming is used together with full-length movies and similar content, meaning that video quality and continuous playback are the most important metrics.

Live streaming with buffering is very similar to on-demand streaming, except that the whole video is not available when the streaming starts. As defined in the introduction, live in the context of this paper is liveness, and by delaying playback by a given number of segments (the startup delay), a trade off between liveness and smoothness is made. Provided that all requested segments are received before their playout deadline, the total delay compared to the no-delay broadcast is *startup_delay* + *initial_segments_transfer_time*. Any errors occurring during transfer cause a further reduction in liveness.

Live streaming without buffering has liveness as the most important metric and is the opposite of on-demand streaming. Segments (requests) are skipped if the stream lags too far behind the broadcast, and a requirement for being as live as possible is that the startup delay is the lowest that is allowed by the streaming system. In our case, this limit is 2 s (one segment), so the client lags *2 s*+ *initial_segment_transfer_time* behind the no-delay broadcast when playback starts, and skips segments if the lag exceeds the length of one segment.

3.2. Multilink support

Several changes were made to the DAVVI streaming system client (equivalent to the DASH Access Client) to support multiple links. We implemented our multilink HTTP download and pipelining mechanisms [16], as well as the request scheduler and subsegment approaches described in Section 3.3. The scheduler is responsible for distributing segment requests among the links efficiently, while the subsegment approaches try to make sure that each link is used to its full capacity.

The routing table on the client must be configured properly to allow DAVVI, or any other application, to use multiple links simultaneously. The network subsystem must be aware of the default interface and know how to reach other machines in the connected networks, and packets must be sent through the correct interfaces.

[2] DAVVI divides videos into closed-GOP segments in order to allow for searching and creating arbitrary playlists on the fly.

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

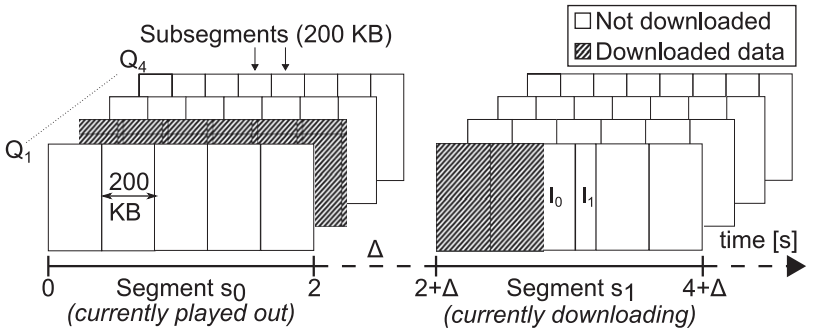

Fig. 1. In this example, the transfer of segment s_0 in quality Q_2 has finished via interfaces I_0 and I_1. As the throughput dropped, the interfaces currently collaborate on downloading the third subsegment of a lower quality segment.

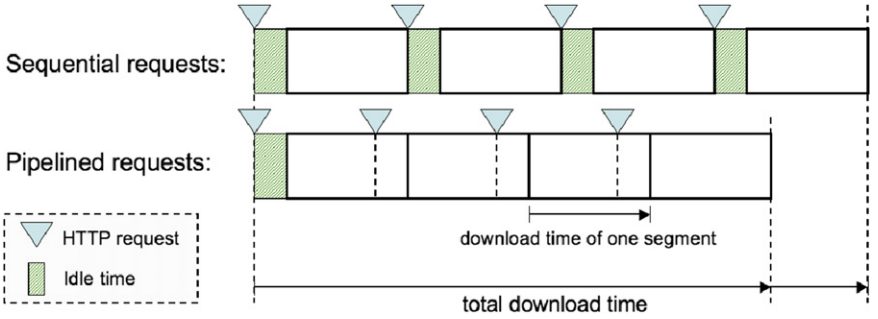

Fig. 2. From a client-side perspective, HTTP pipelining eliminates the time overhead incurred by a sequential processing of requests and replies.

Once the routing table is configured, multilink-support in the application can be enabled by binding network sockets to the desired interfaces. This is supported by all major operating systems.

3.2.1. Subsegments of varying size

Even though DAVVI divides the video into segments, the segments can still be large. Therefore, they are divided into smaller logical subsegments to reduce latency, increase the granularity of the request scheduler and allow the transfer of video over multiple interfaces simultaneously. Using the *range retrieval request*-feature of HTTP/1.1, it is possible to request a specific part of a file (a subsegment). For example, if the first 50 kB of a 100 kB large file are requested, bytes 0–49 999 are sent from the server.

The subsegment approach decides how complete segments are divided into subsegments, and how the links are allocated their share of the data. For example, Fig. 1 shows how a 200 kB subsegment would be divided between two links with a bandwidth ratio of 3:2, according to the dynamic subsegment approach presented in Section 3.3. Interface zero requests 120 kB and interface one 80 kB.

3.2.2. Request pipelining

Dividing segments into subsegments introduces two challenges. First, subsegments cause an increase in the number of HTTP GET-requests. This reduces performance, as the client spends more time idle waiting for responses. HTTP pipelining, illustrated in Fig. 2, is used to reduce both the number and the duration of these idle periods. Initially, two subsegments are requested on every interface, and then, a new subsegment is requested for each one received. This ensures that the server always has a request to process, and that there is always incoming data.

The second challenge is related to the size of the subsegments, and only applies when fixed size subsegments are used. If they are too small, an entire subsegment might have been sent from the server before the next is requested, causing interfaces to become idle. This can be alleviated by having a fixed subsegment size. For example, our earlier work [16] has shown that 100 kB is well suited, as it allows sufficient flexibility for the request scheduler, and is large enough to take advantage of HTTP pipelining.

The reason the subsegment size challenge does not apply to subsegment approaches that calculate the size dynamically is that the size of the subsegments matches the links' performance. Thus, the size of the subsegment is equal to what the link can transfer.

3.3. Quality adaptation and request schedulers

To use multiple links efficiently, segments must be requested according to the available resources. If a slow interface is allocated a too large share of a segment, the performance of the whole application might suffer. For example, the segment may not be ready when it is supposed to be played out, causing a deadline miss and an interruption in playback.

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

The request scheduler is responsible for distributing requests and adjusting the desired video quality. In combination with the subsegment approaches, it is the most important part of our multilink streaming approach. Without a good scheduler and subsegment approach, adding multiple interfaces can cause a drop in performance and have a significant effect on the user experience. For example, the quality adaptation might be too optimistic and select a higher quality than the links can support, or links are not utilized to their full capacity.

In this paper, we compare the performance of two subsegment size approaches. The underlying request scheduler is identical for both approaches, i.e., the same technique is used to measure the link characteristics (throughput and RTT) and adjust the video quality. The video quality adaptation is outlined in Algorithm 1. First, the client calculates how much content it has already received and is ready for playout (*transfer_deadline*), and estimates how long it takes to receive already requested data (*pipeline_delay*). The *pipeline_delay* is subtracted from the *transfer_deadline* to get an estimate of how much time the client can spend receiving new data without causing a deadline miss. This estimate is then compared against estimates of the time it takes to receive the desired segment in the different qualities, and the most suited quality is selected.

Algorithm 1 (*Quality adaptation mechanism*).

```
transfer_deadline = time_left_playout + (segment_length *
num_completed_segments)
pipeline_delay = requested_bytes_left/aggregated_throughput
for quality_level = "super" to "low" do
  transfer_time = segment_size[quality_level]/aggregated_throughput
  if transfer_time < (transfer_deadline − pipeline_delay) then
    return quality_level
  end if
  reduce quality_level
end for
```

The two approaches differ in how they divide segments. The *static subsegment approach*, which is the one that was used in [2], divides each segment into fixed-sized 100 kB subsegments. Requests for subsegments are distributed among the links, and provided that there are more subsegments available, new requests are pipelined as soon as possible.

However, our earlier work did not sufficiently consider the challenges introduced by limited receive buffers and timeliness. In addition to the *last segment problem* [16], caused by clients having to wait for the slowest interface to receive the last subsegment of a segment, the static subsegment approach is unable to reach maximum performance unless the receive buffer is large enough to compensate for the link heterogeneity. This problem is discussed in more detail in Section 3.4.

Increasing the buffer size is in many cases not acceptable, desirable or even possible. We, therefore, decided to improve on our old subsegment approach by allocating data to the links in a more dynamic fashion. The segments are now divided into blocks of *number_of_interfaces* * 100 kB (limited by the total segment size), where 100 kB is a well suited share of data to request over one link, as discussed earlier and presented in [16]. These blocks are then divided into subsegments, and the size of each subsegment is calculated based on the measured throughput of the interface it will be requested through. Pipelining is still done as soon as possible, and the algorithm is outlined in Algorithm 2.

Algorithm 2 (*Dynamic subsegment approach [simplified]*).

```
share_interface = throughput_link/aggregated_throughput
size_allocated_data = share_interface * subsegment_length
if size_allocated_data > left_subsegment then
  size_allocated_data = left_subsegment
end if
update left_subsegment
request new Subsegment(size_allocated_data)
```

By allocating the data dynamically based on performance, the need for a big buffer is removed, and the effect of the *last segment problem* is reduced. The problem can still occur, but because the performance of the links is used when allocating data, it has a smaller effect. When dividing segments dynamically, the performance for a given buffer size should ideally be the same for all link heterogeneities. This approach is hereby referred to as the *dynamic subsegment approach*.

3.4. Considerations: static vs. dynamic

The switch from a static to a dynamic subsegment approach was motivated by the buffer requirement imposed by the *static* subsegment approach. Unless the buffer is large enough to compensate for the link heterogeneity, the client is unable to reach maximum performance.

With a short startup delay and small buffer, the request scheduler is only allowed a little slack when requesting the first segment after the playout has started. Assuming that the links are heterogeneous and none exceed the bandwidth requirement for the stream by a wide margin, this forces the scheduler to pick a segment of lower quality. Smaller segments consist of fewer subsegments, so the slowest link is allocated to a larger share of the data, and has a more significant effect on throughput. This continues until the throughput and quality stabilizes at a lower level than the links might support. In other words, the request scheduler is caught in a vicious circle.

Furthermore, increasing the receive buffer size and startup delay improves the situation. A larger receive buffer allows the scheduler more slack, so the first segment after the startup delay is requested in a higher quality than with a small buffer. Larger segments consist of more subsegments than smaller ones, so the slowest interface is made responsible for less data. Provided that the buffer is large enough, the links are allocated their correct share of subsegments (or at least close to). Thus, throughput measurements are higher and a better video quality distribution is achieved.

On the other hand, when dividing the segments into subsegments *dynamically*, the buffer size/startup delay

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

problem is avoided. Each link is allocated their correct share of a segment (at that time), so the slower links are made responsible for less data. However, there are challenges when dividing segments dynamically as well. In the first version of the dynamic subsegment approach, we used the size of the segment to determine a link's share. As it turned out, the performance of this approach suffers when faced with dynamic network environments. Links are often allocated too much data, making the approach vulnerable to throughput and delay variance. Therefore, we limited the amount of data used for calculating a link's share to *number_of_interfaces* ∗ 100 kB, as presented earlier.

3.5. DAVVI and DASH

The DAVVI streaming system does not follow the DASH standard, however, it follows the same design principles. Also, the information required to adapt the quality according to Algorithm 1 is present in both DAVVI and DASH-based solutions. Finally, because both make use of HTTP, the dynamic subsegment approach can be ported directly to DASH.

The DASH-standard defines the structure (an XML-schema) of the MPD, and a video is represented by five levels. The top level, the actual MPD, contains attributes that are valid for the entire stream, for example if it is on-demand or live video (*@type*) and the startup delay (*@minBufferTime*). The second level is called *period*, and an MPD consists of one or more periods. A period is used to describe the entire video clip, and can be used when for example an MPD contains information about several episodes of a series. There is no equivalent to the period-element in DAVVI.

Each period can contain one or more *groups*, which again consists of one or more *representations*. A representation describes one or more video segments, and a group describes the range of attribute values that are valid for each representation it contains. Examples of group attributes include the bandwidth requirement (*@minBandWidth* and *@maxBandwidth*) and minimum resolution (*@minWidth* and *@minHeight*). The representations then specifies which values are valid. If groups are not needed, representations can be added directly to the period. Finally, each representation contains one or more segment elements. The segment element provides information about the segment that will be downloaded, for example the URL. Subsegments can be used when data is requested from a specific index in a segment, and the DASH segment and subsegment maps directly to the DAAVI terms.

DAVVI's equivalent to the MPD is a text file, where each segment is represented by one line. This line contains the filename of the segment and the bandwidth requirement for each of the four quality levels. If we assume that the video segments described by the MPD are independent (i.e., there is no decoding dependencies), a comparable MPD file would consist of one period for each video, and the representations would be added directly to the period. One representation would be needed for each quality layer, and the *bandwidth* parameter used to describe the minimum bandwidth requirement for that level (according to the standard). Every segment would be contained in a *SegmentInfo*-element, and a representation would contain all segments belonging to that quality level.

In order to implement the quality adaption mechanism (Algorithm 1) in DASH, first, each representation's *bandwidth* attribute, together with the constant length (in time) of each segment, will be used to estimate the *transfer_time*. After the highest possible quality has been found, the segment will be divided into subsegments by our multilink component. An initial estimation of the size of each video segment (in bytes) can also be derived from the representation's *bandwidth* attribute. As HTTP always includes the complete segment length in replies to range-requests, the correct value can be used for calculating the size of the following subsegments. Since all the required information is present or can easily be retrieved (the aggregated throughput is calculated by the client and the segment size is known), no modifications have to be made to the subsegment approaches.

4. Experimental setup

To evaluate the performance of the two subsegment approaches, two testbeds were created. We wanted to measure the performance in the real world and in a controlled environment, to fully control all parameters.

The controlled environment-testbed, shown in Fig. 3, consists of a client and a server (Apache 2) connected using two independent 100 MB/s Ethernet links. Both client and server run Linux 2.6.31, and to control the different link characteristics, the network emulator *netem* is used with a hierarchical token bucket queueing discipline. For measuring the real world performance, we made experiments in a wireless scenario where the client was connected to one public WLAN (IEEE 802.11b) and an HSDPA network. The characteristics of these networks are summarized in Table 1, and the reason we choose wireless networks is that they present a more challenging environment than fixed links.

To get comparable results, the same VBR-encoded video clip was used in all the experiments. The clip shows a football match has a total playout duration of 100 min

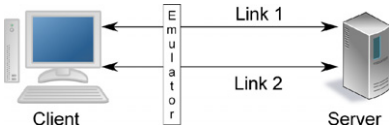

Fig. 3. Our controlled environment-testbed.

Table 1
Observed characteristics of used links.

Characteristic	WLAN	HSDPA
Average experienced throughput (kB/s)	287	167
Average RTT for header-only IP packets (ms)	20	100
Average RTT for full-size IP packets (ms)	30	220

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

Table 2
Quality levels and bitrates of the soccer movie.

Quality level	Low	Medium	High	Super
Minimum bitrate (kB/s)	524	866	1491	2212
Average bitrate (kB/s)	746	1300	2142	3010
Maximum bitrate (kB/s)	1057	1923	3293	4884

(3127 segments of 2 s) and was available in four qualities. We chose a subset of 100 segments, and the bandwidth requirements are shown in Table 2.

If the experiments had been performed with a DASH streaming solution, @duration would be equal to 2 s, each period would contain four representations (groups) and @minBufferTime is specified using a command line parameter. Also, we have not focused on how the MPD (or MPD-equivalent) is retrieved. We assume that the MPD stored at the client always contains the required information.

5. Results and discussion

When evaluating the performance of the two subsegment approaches, we measure the video quality and deadline misses. The video quality (the number of segments downloaded in each quality) depends on the bandwidth aggregation. That is, an efficient aggregation results in a higher throughput. Thus, the quality increases. Deadline misses are of highest importance from a user's perspective, with respect to perceived video quality. The number of deadline misses depends on the subsegment approach. A poor approach allocates too much data to slower interfaces, causing data to arrive late and segments to miss their deadlines.

In our earlier work [2], we compared the single-link and multilink performance of the static subsegment approach, as well as the performance of the request scheduler. In this paper, the focus is on the differences between the two subsegment approaches in a multilink scenario. With multiple links, bandwidth and latency heterogeneity are the two most significant challenges, so we decided to look at their effect on performance, both in a completely controlled environment, with emulated network dynamics and in a real-world wireless environment. Multilink request schedulers and subsegment approaches have to take heterogeneity into account, otherwise the performance is less than ideal, and sometimes worse than the performance when using only one of the links [2].

The combined bandwidth of the emulated links was always 3 MB/s, which is equal to the average bandwidth requirement for the highest quality of the video clip used in our experiments. The startup delay was equal to the buffer size in all the experiments, forcing the application to fill the buffer completely before starting playback.

5.1. Bandwidth heterogeneity

To measure how bandwidth heterogeneity affects the performance of the two subsegment approaches, the controlled testbed was used and configured to provide different levels of bandwidth heterogeneity. The goal with using multiple links simultaneously was that the performance should match that of a single 3 MB/s link, in other words, the aggregated logical link should perform just as well as an actual 3 MB/s link.

5.1.1. On-demand streaming

For an on-demand scenario, Fig. 4 shows the video quality distribution for a buffer size of two segments (4 s delay). The bandwidth ratio is shown along the *x*-axis, and the *X*:*Y* notation means that one link was allocated *X*% of the bandwidth, while the other link was allocated *Y*%. The bars represents the four video qualities, and the *y*-value of each bar is its share of the received segments. The reason we did not divide the *y*-axis into the four quality-levels and plot the average quality is that the *y*-value of a bar would end up between qualities. As the quality level "Medium.5" (or similar) does not exist, we decided to plot the quality distributions instead.

When a single link was used, the expected behavior can be observed. As the available bandwidth increased, so too did the video quality. Also, the static and dynamic subsegment approaches achieved more or less the same video quality.

However, the situation was different with multiple links. The dynamic subsegment approach adapted to the heterogeneity, the performance was close to constant irrespective of link heterogeneity, and significantly better than when a single link was used. However, the performance never reached the level of a single 3 MB/s link (even though the difference was small), due to the additional overhead introduced when using multiple links.

The static subsegment approach, on the other hand, suffered from the problem discussed in Section 3.4, when the heterogeneity increased, the achieved video quality decreased. When the bandwidth ratio was 80:20, the single link performance exceeded multilink.

As discussed in [2], the static subsegment approach requires the buffer to be large enough to compensate for the bandwidth heterogeneity. A rule of thumb is that the buffer size shall be equal to the ratio between the links. For example, with a bandwidth ratio of 80:20, the ideal buffer size is five segments, because the fast interface can receive four segments for every one segment over the slow interface. However, this is only correct with a CBR-encoded video. With a VBR-encoded video, the segments are of different sizes and have different bandwidth requirements. The latter explains why a buffer size of four segments was sufficient for the multilink performance to exceed that of a single link with a bandwidth ratio of 80:20, as seen in Fig. 5. This figure shows how increasing the startup delay and buffer size improved the video quality when the bandwidth ratio was 80:20.

Fig. 6 shows the average number of deadline misses for the bandwidth ratios. As expected when faced with static links, both subsegment approaches performed well. The bandwidth measurements and quality adaption were accurate, there were close to no deadline misses, except for when the buffer was unable to compensate for heterogeneity. The deadline misses when the bandwidth ratio was 80:20 were caused by the slow interface delaying the reception and thereby playback of some segments.

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

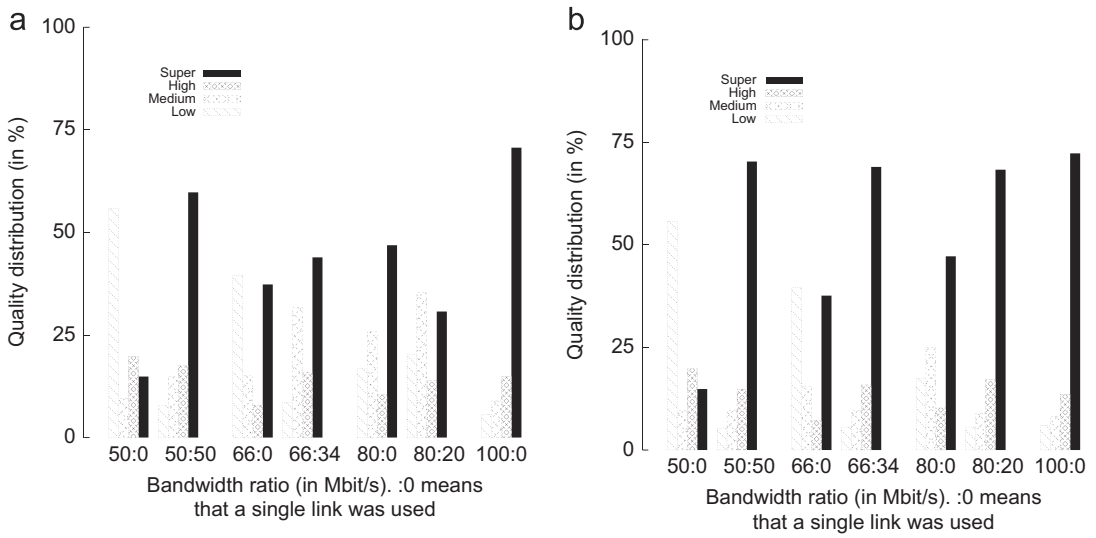

Fig. 4. Video quality distribution for different bandwidth heterogeneities, buffer size/startup delay of two segments (4 s) and on-demand streaming. (a) Static subsegment approach, (b) dynamic subsegment approach.

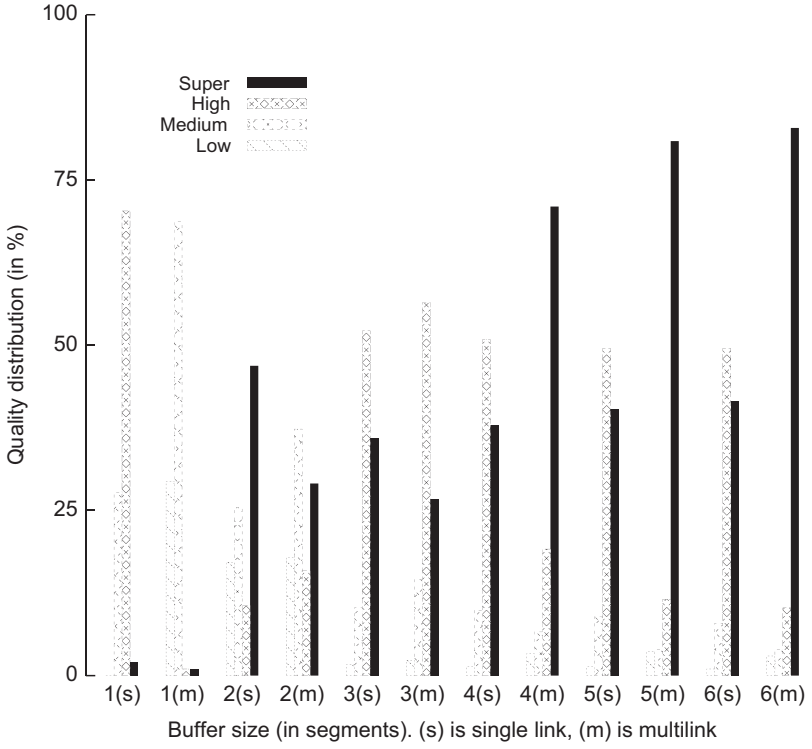

Fig. 5. Quality distribution plotted against the startup delay/buffer size for a bandwidth ratio of 80:20, on-demand streaming and static subsegment approach.

However, all deadline misses were significantly lower than the segment length, the worst observed miss was only ~0.3 s.

5.1.2. Live streaming with buffering

When live streaming with buffering was used, the experimental results were similar to those of the on-demand streaming tests. The single link performance of the two subsegment approaches was more or less the same, and when multiple links were used, the dynamic subsegment approach showed similar performance irrespective of bandwidth heterogeneity, while the performance of the static subsegment approach suffered from the buffer being too small to compensate for the link heterogeneity. The number of deadline misses was also the same as with on-demand streaming. The reason for

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

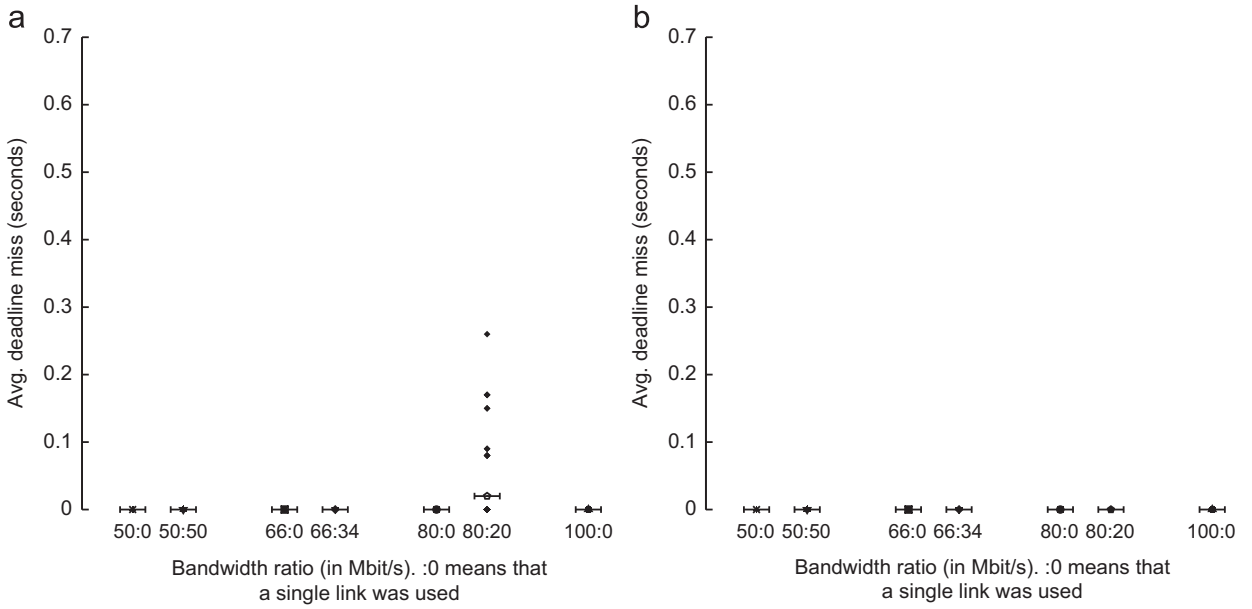

Fig. 6. Deadline misses for different levels of bandwidth heterogeneity with on-demand streaming, buffer size/startup delay of two segments (4 s). (a) Static subsegment approach, (b) dynamic subsegment approach.

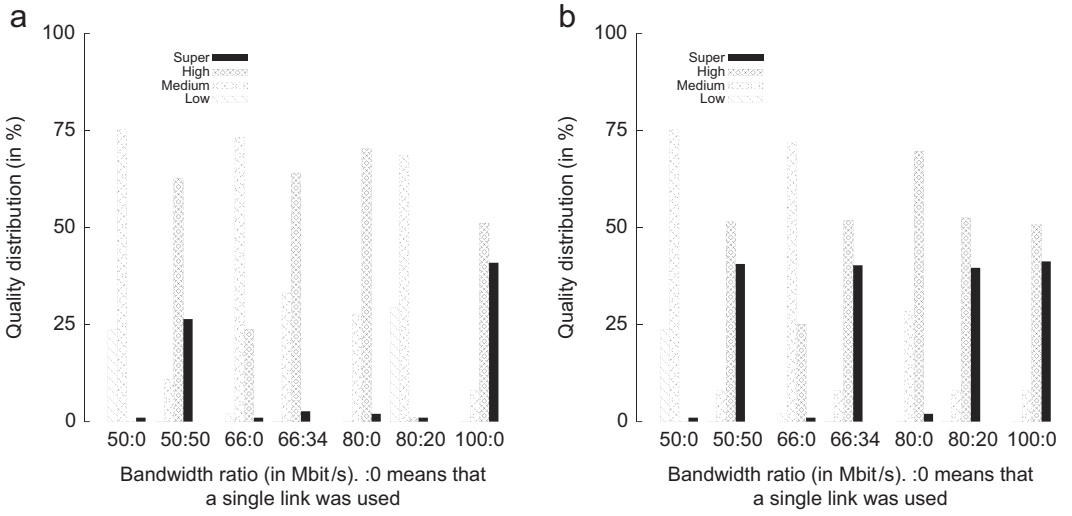

Fig. 7. Video quality distribution for different levels of bandwidth heterogeneity, buffer size/startup delay of one segment (2 s startup delay) and live streaming with buffering. (a) Static subsegment approach, (b) dynamic subsegment approach.

these similar results is that segments were always ready also when live streaming with buffering was used. The client was never able to fully catch up with the no-delay broadcast.

With on-demand streaming, it makes no sense to talk about liveness. However, in live streaming with buffering, liveness is one of the most important criteria. With a startup-delay/buffer size of two segments, the static subsegment approach added an additional worst-case delay of 4 s compared to the no-delay broadcast. The dynamic subsegment approach caused an additional worst-case delay of 2.5 s.

Fig. 7 shows the effect of increasing the liveness to the maximum allowed by DAVVI. Both the startup delay and the buffer size were set to one segment (2 s delay). The dynamic subsegment approach was able to cope well with the increased liveness requirement, and showed a significant increase in performance compared to using a single link. Also, the performance was independent of the bandwidth heterogeneity. The static subsegment approach, on the other hand, struggled because of the small buffer. In addition to pipelining losing almost all effect, it only worked within a segment, the problem discussed in Section 3.4 came into play. The performance

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

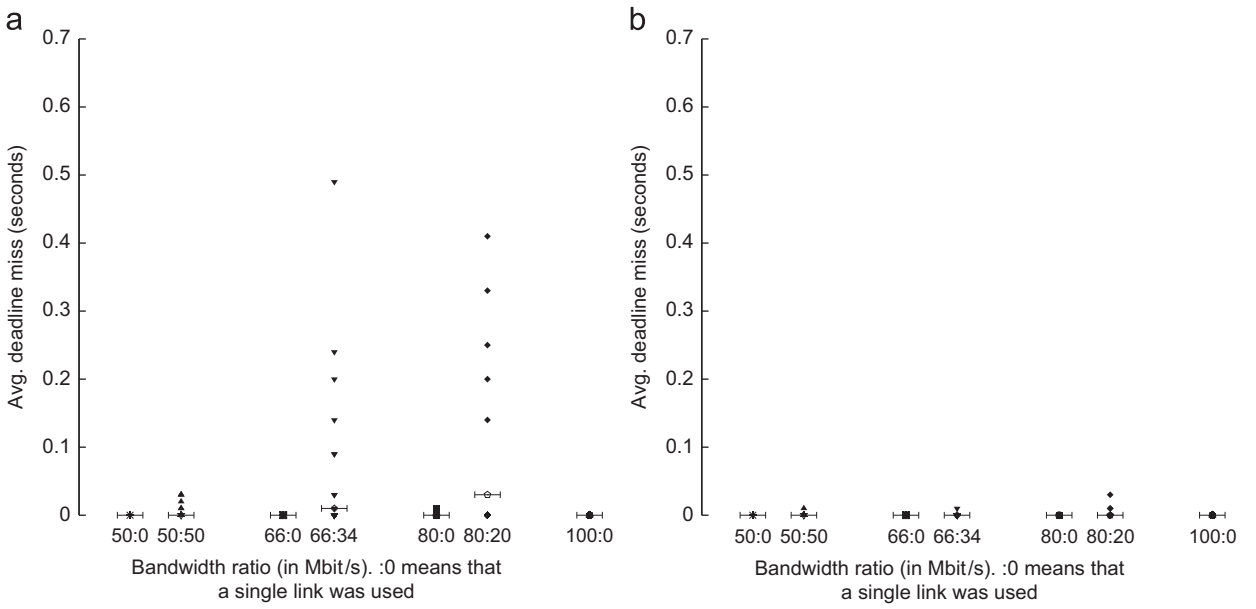

Fig. 8. Deadline misses for a buffer size of one segment (2 s startup delay) and various levels of bandwidth heterogeneity, live streaming with buffering. (a) Static subsegment approach, (b) dynamic subsegment approach.

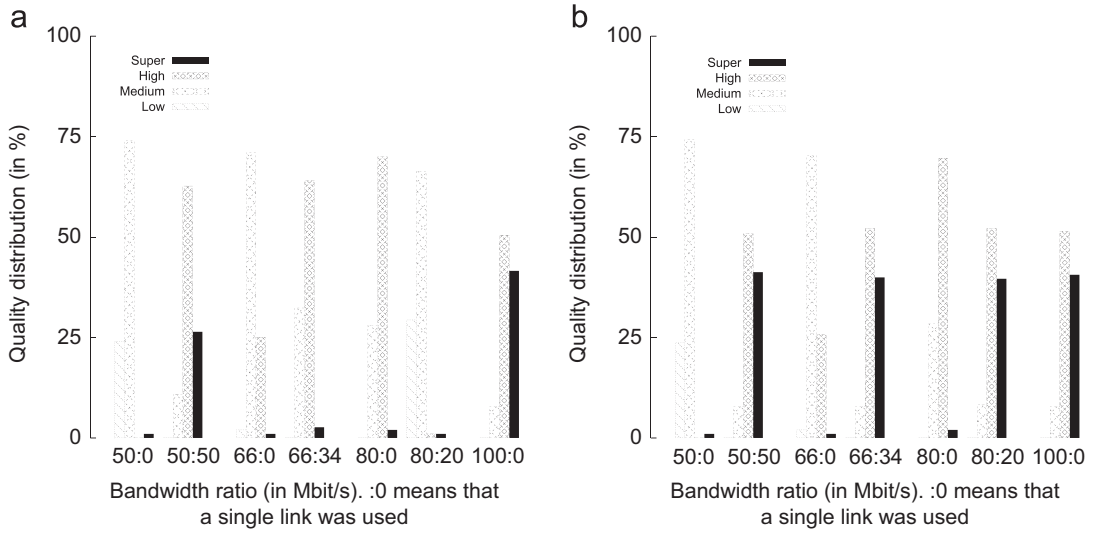

Fig. 9. Video quality distribution for a buffer size of one segment (2 s startup delay), live streaming without buffering and bandwidth heterogeneity. (a) Static subsegment approach, (b) dynamic subsegment approach.

hit was reflected in the deadline misses shown in Fig. 8. While the dynamic subsegment approach was able to avoid almost all deadline misses, the static subsegment approach caused several misses. When the dynamic subsegment approach was used, a worst-case additional delay of 2.3 s was observed, compared to 6 s with the static subsegment approach.

5.1.3. Live streaming without buffering

The goal with skipping segments is that the stream shall be as live as possible, the client chooses not to request old segments. Skipping leads to interruptions in playback, but did not affect the video quality, as shown in Fig. 9. The results were the same as for live streaming with buffering and a buffer size/startup delay of one segment—the dynamic subsegment approach improved the performance significantly, while the static subsegment approach suffered from the problem discussed in Section 3.4. The deadline misses were similar to Fig. 8, in other words, the dynamic subsegment approach was able to avoid most deadline misses, unlike the static subsegment approach.

However, the number of skipped segments was the same for both subsegment approaches, with a worst case of two segments. This was because of the first segment, which is requested in the highest quality to get the most

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

accurate measurements, at the expense of a longer startup latency. The highest quality level consists of the largest files, i.e., the files that will be divided into the most subsegment (“samples”). Also, the first segment is the only segment that cannot cause a deadline miss, so it is safe to request in any quality. However, for live streaming without buffering, a more conservative approach should be considered.

The subsegment approaches assume that all links are equal and initially allocate the same amount of data. If the links are not homogeneous, which was the case in almost all of our experiments, or able to support the quality, the segment takes longer than 2 s to receive and one or more segments are skipped. The deadline misses and initial segment transfer time with the static subsegment approach caused a worst case additional total delay of 1.86 s, which is less than the length of a single segment, and explains why the static subsegment approach did not skip more segments than the dynamic subsegment approach.

5.2. Latency heterogeneity

When measuring the effect of latency heterogeneity on video quality and deadline misses, we used one link that had a constant RTT of 10 ms. The other link was assigned an RTT of r ms, with $r \in \{10, 20, \ldots, 100\}$. The bandwidth of each link was limited to 1.5 MB/s, and a buffer size of two segments was used (according to the rule of thumb presented earlier and [2]).

5.2.1. On-demand streaming

Fig. 10 depicts the video quality distribution for different levels of latency heterogeneity. As shown, RTT heterogeneity did not have a significant effect on video quality, independent of subsegment approach. The bandwidth ratio was 50:50, and both subsegment approaches achieved close to the same quality distribution as in the on-demand bandwidth heterogeneity experiments (for a 50:50 bandwidth ratio), shown in Fig. 4, for all latency heterogeneities. The reason for the performance difference between the two approaches is that the dynamic subsegment approach is able to use the links more efficiently.

For both subsegment approaches, a slight decrease in quality as the heterogeneity increased can be observed, indicating that the RTT heterogeneity at some point will have an effect. The reason for the quality decrease, is that it takes longer to request, and thereby receive each segment. The approaches measure a lower throughput, and potentially reduces the quality of the requested segments.

HTTP pipelining is used to compensate for high RTT and RTT heterogeneities. However, pipelining is not possible when the buffer is full and the next segment cannot be requested immediately. Also, TCP throughput is lower for short transfers and high delay.

The deadline misses were also similar to the 50:50-case in the bandwidth heterogeneity experiments shown in Fig. 6. As expected in a static environment, both subsegment approaches made accurate decisions and no deadline misses were observed.

5.2.2. Live streaming with buffering

As with bandwidth heterogeneity, the results when measuring the effect of latency heterogeneity on live streaming with buffering were very similar to those with on-demand streaming. The quality distribution and deadline misses were not affected for the levels of heterogeneity we have used. However, a slight decrease in video quality as the RTT heterogeneity increases can be seen also here. The worst case additional delay compared to the no-delay broadcast was 2 s for both subsegment approaches.

Reducing the buffer size/startup delay to one, caused a similar reduction in performance as the ones seen in

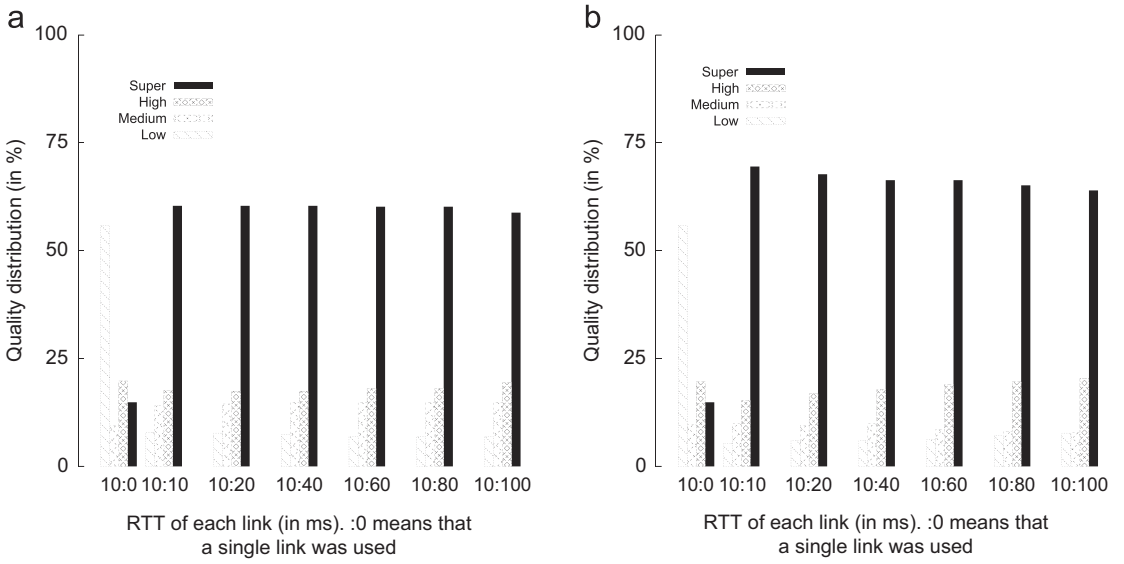

Fig. 10. Video quality distribution for two-segment buffers and various levels of latency heterogeneity. (a) Static subsegment approach, (b) dynamic subsegment approach.

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

Figs. 7 and 8 (for a 50:50 bandwidth ratio). However, as for a buffer size of two segments, the latency heterogeneity did not affect the quality distribution or deadline misses. Both subsegment approaches caused a worst additional case additional delay of 2.5 s.

5.2.3. Live streaming without buffering

The observed video quality and deadline misses using live streaming without buffering were similar to the earlier latency heterogeneity experiments. RTT heterogeneity did not have a significant impact on video quality, however, a slight decrease can be observed, indicating that the RTT heterogeneity will affect the performance of the approaches at some point. As in the bandwidth heterogeneity experiments for live streaming without buffering, the number of skipped segments and the total delay compared to the no-delay broadcast were the same for both approaches. When multiple links were used, zero segments were skipped, and a worst case additional delay of 1.86 s was observed for both subsegment approaches, caused by the first segment. Even though the initial assumptions that both links are homogeneous were correct, the links were unable to support the bandwidth requirement for this segment.

5.3. Emulated dynamics

Dynamic links impose different challenges than static links, the scheduler has to adapt to often rapid changes in the network. To expose the two subsegment approaches to dynamic links while still having some control over the parameters, we created a script which emulates our observed real-world network behavior. The sum of the bandwidth of the two links was always 3 MB/s, but at random intervals of t seconds, $t \in \{2,\ldots,10\}$, the bandwidth bw MB/s, $bw \in \{0.5,\ldots,2.5\}$ of each link was updated. The RTT of link 1 was normally distributed between 0 ms and 20 ms, while the RTT of link 2 was uniformly distributed between 20 ms and 80 ms. A buffer size of six segments was used to compensate for the worst case bandwidth heterogeneity, according to the rule of thumb presented earlier and in [2], except for in the live streaming without buffering experiments. Each subsegment approach was tested 30 times for each type of streaming, and the results shown are the averages of all measurements.

5.3.1. On-demand streaming

The aggregated throughput when combining emulated link dynamics with on-demand streaming is shown in Fig. 11. With both subsegment approaches, adding a second link gave a significant increase in throughput, and thereby achieved video quality. Also, as in the other experiments where the buffer size was large enough to compensate for link heterogeneity, both approaches gave close to the same video quality distribution, with a slight advantage to the dynamic subsegment approach. The average aggregated throughput oscillated between the average bandwidth requirement for "High" and "Super" quality, the quality distribution is presented in Table 3.

In terms of deadline misses, shown in Fig. 12, both approaches were as accurate. When a single link was used, misses occurred, however, none were severe. The worst case observed miss for both approaches was less than 0.5 s. With multiple links, both approaches avoided all deadline misses.

Table 3
Quality distribution, emulated dynamics and on-demand streaming.

Subsegment approach	Low (%)	Medium (%)	High (%)	Super (%)
Static, single-link	31	27	28	15
Static, multilink	4	4	11	81
Dynamic, single-link	30	26	29	15
Dynamic, multilink	3	3	10	83

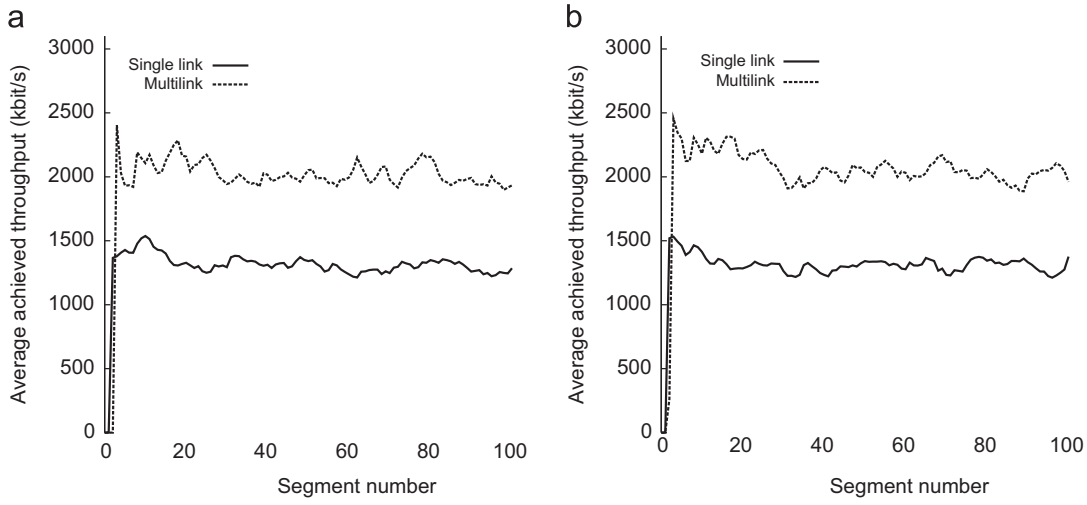

Fig. 11. Average achieved throughput of the schedulers with emulated dynamic network behavior, on-demand streaming. (a) Static subsegment approach, (b) dynamic subsegment approach.

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

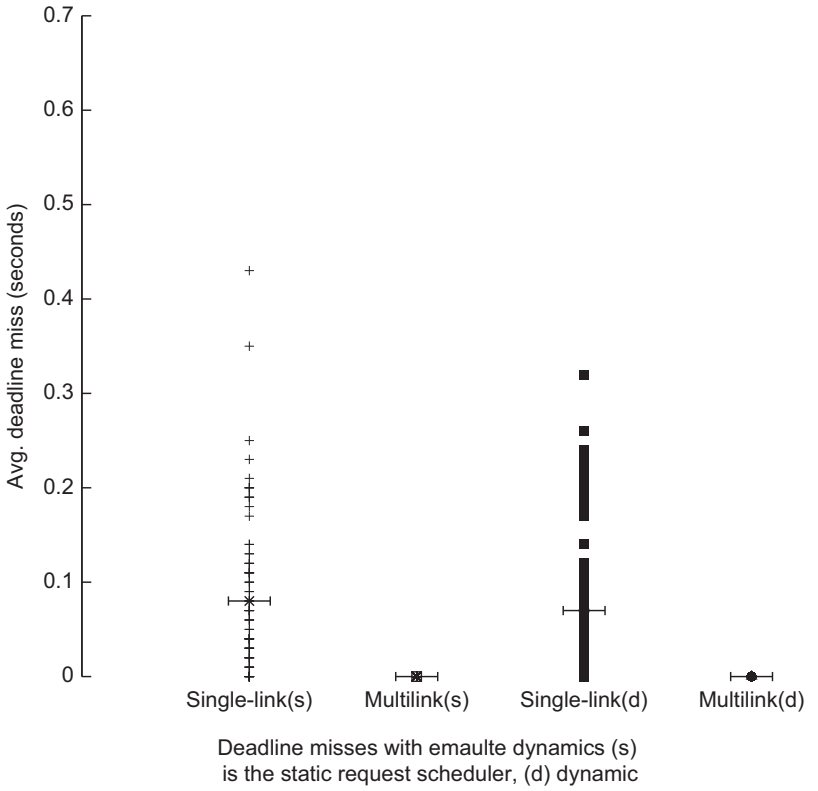

Fig. 12. Deadline misses with on-demand streaming and emulated dynamics.

Table 4
Quality distribution, emulated dynamics and live streaming with buffering.

Subsegment approach	Low (%)	Medium (%)	High (%)	Super (%)
Static, single-link	30	26	28	16
Static, multilink	4	4	11	81
Dynamic, single-link	29	26	29	15
Dynamic, multilink	3	3	11	82

Table 5
Quality distribution, emulated dynamics and live streaming without buffering.

Subsegment approach	Low (%)	Medium (%)	High (%)	Super (%)
Static, single-link	41	44	14	1
Static, multilink	15	45	35	5
Dynamic, single-link	44	41	14	1
Dynamic, multilink	2	28	55	15

5.3.2. *Live streaming with buffering*

As with both bandwidth and latency heterogeneity, the performance of live streaming with buffering was similar to the on-demand streaming experiments, seen in Fig. 11. A significant increase in performance was seen when a second link was added, and the quality distributions are found in Table 4. The deadline misses were also the same as in the on-demand experiments (Fig. 12), when multiple links were used no misses occurred. The worst-case additional delay compared to the no-delay broadcast was 2.3 s, caused exclusively by the initial segment transfer time.

5.3.3. *Live streaming without buffering*

The live streaming without buffering experiments were performed with the same settings as the other emulated dynamics experiments, except that a buffer size and startup delay of one segment was used. This was, as discussed earlier, done to increase the liveness to the maximum that DAVVI allows (one segment).

As in the earlier live streaming without buffering experiments, the two subsegment approaches performed differently, the static approach was outperformed by the dynamic approach. This was because the dynamic subsegment approach adapts better to smaller buffers, and the performance difference is reflected in the quality distribution, presented in Table 5, and seen in Fig. 13. While the static subsegment approach most of the time achieved a throughput that exceeded the average requirement for "Medium" quality, the dynamic subsegment approach exceeded the requirement for "High" quality.

However, both subsegment approaches experienced deadline misses, as shown in Fig. 14. None were severe, as earlier, the worst case observed miss was around 0.5 s. However, if continuous playback had been important, a bigger buffer and startup delay should have been used. This, of course, would involve making a trade-off between liveness and quality of the user experience. The deadline misses are also reflected in the number of skipped

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

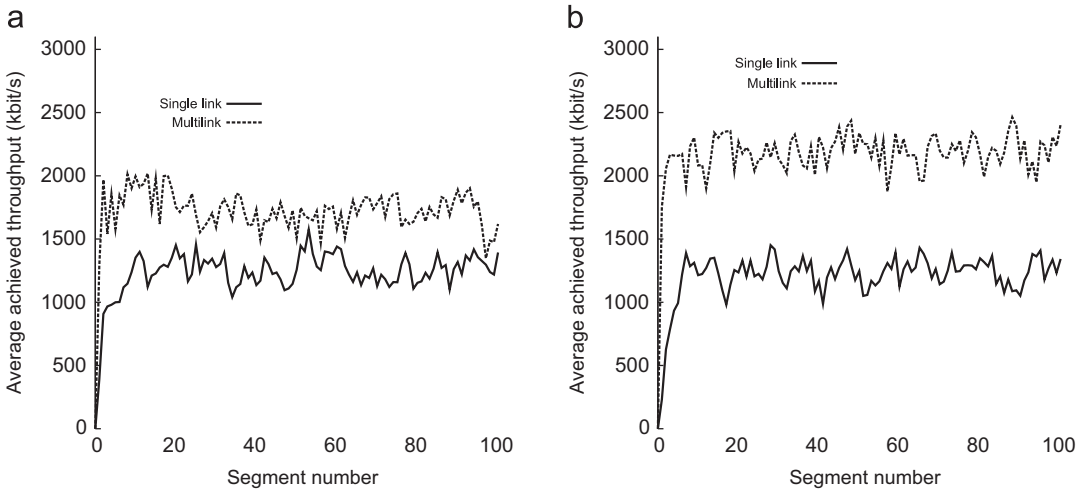

Fig. 13. Average achieved throughput of the schedulers with emulated dynamic network behavior, live streaming without buffering. (a) Static subsegment approach, (b) dynamic subsegment approach.

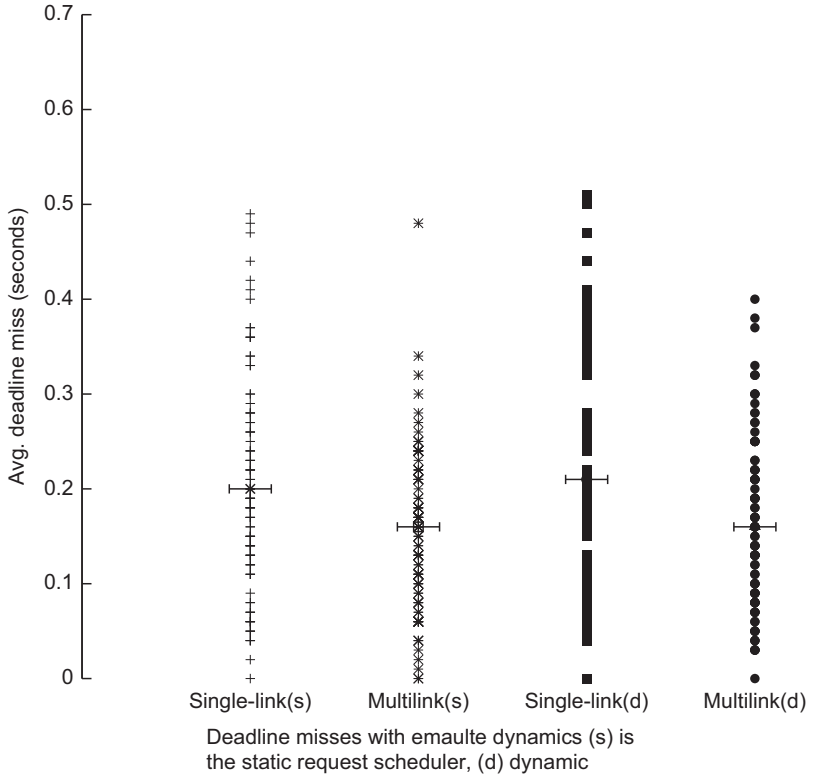

Fig. 14. Deadline misses with live streaming without buffering and emulated dynamics.

segments, on average both subsegment approaches skipped five segments.

5.4. Real world networks

Our real world experiments were conducted with the networks described in Table 1, and a buffer size of three was used to compensate for the worst-case measured bandwidth heterogeneity (except when measuring the performance for live streaming without buffering). The tests were run interleaved to get comparable results, and the experiments were performed during peak hours (08–16) to get the most realistic network conditions. That is, we did not want to have the full capacity of the networks to ourself.

5.4.1. On-demand streaming

The average aggregated throughput for on-demand streaming and real world networks can be found in

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

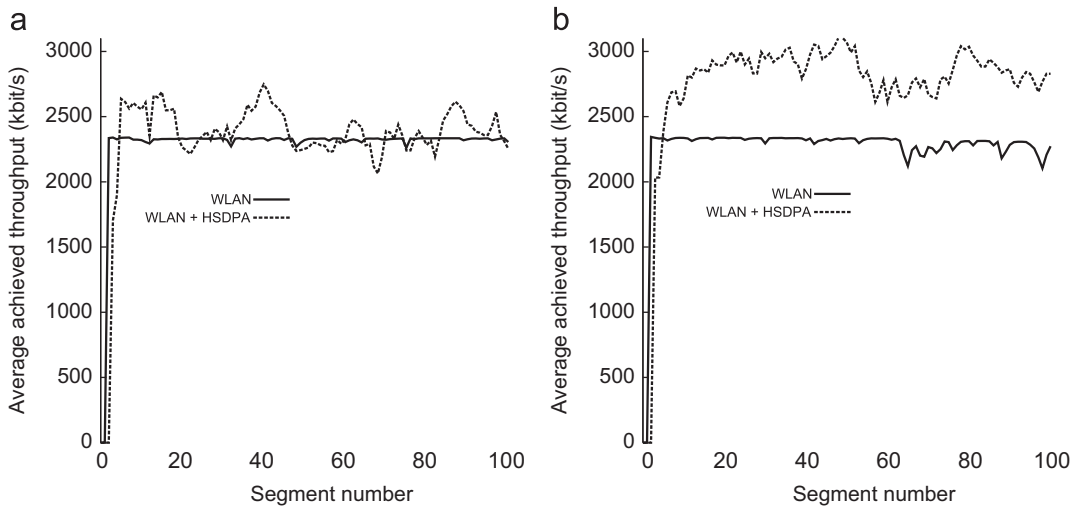

Fig. 15. Average achieved throughput of the schedulers with real-world networks, on-demand streaming. (a) Static subsegment approach, (b) dynamic subsegment approach.

Fig. 15. There was a significant difference in performance between the two subsegment approaches. While the dynamic subsegment approach showed an increase in performance when a second link was added, the static subsegment approach did not benefit that much. In fact, sometimes the aggregated throughput was less than when a single link was used. The reason for the performance difference was, as earlier, that the dynamic subsegment approach is able to utilize the links more efficiently, it adapts better to the buffer size. When the static approach was used, the HSDPA link was allocated too much data. The performance difference is also reflected in the quality distribution shown in Table 6.

In terms of deadline misses, both subsegment approaches performed equally. Except for some outliers caused by significant and rapid changes in the network conditions, like congestion and interference, both approaches were able to avoid all misses when multiple links were used.

5.4.2. *Live streaming with buffering*

The performance with live streaming with buffering was, as in the other live streaming with buffering experiments, similar to the on-demand performance. The quality distribution is shown in Table 7, and both approaches avoided almost all deadline misses when multiple links were used. A worst-case additional delay compared to the no-delay broadcast of 4 s was observed for both subsegment approaches.

5.4.3. *Live streaming without buffering*

When live streaming without buffering was combined with our real world networks, the performance was similar to that presented in Section 5.3.3. The static subsegment approach struggled with the small buffer, while the dynamic approach adapts better, which resulted in a significantly improved performance. The only significant difference compared to the results in Section 5.3.3 is that the quality distribution for both approaches were better due to more available bandwidth and more stable links, as can be seen in Table 8. This was also reflected in the deadline misses and a lower number of skipped segments.

One common technique for streaming clients to reduce the number of deadline misses is to downshift the quality more aggressively. That is, segments are requested in a lower than possible quality to achieve a smooth playback. As can be seen in Fig. 16, our solution does not do this.

Table 6
Quality distribution, real world networks and on-demand streaming.

Subsegment approach	Low (%)	Medium (%)	High (%)	Super (%)
Static, single-link	1	8	51	40
Static, multilink	5	6	10	79
Dynamic, single-link	3	11	46	41
Dynamic, multilink	3	2	9	86

Table 7
Quality distribution, real world networks and live streaming with buffering.

Subsegment approach	Low (%)	Medium (%)	High (%)	Super (%)
Static, single-link	1	10	49	40
Static, multilink	5	4	7	84
Dynamic, single-link	1	9	49	41
Dynamic, multilink	3	2	5	91

Table 8
Quality distribution, real world networks and live streaming without buffering.

Subsegment approach	Low (%)	Medium (%)	High (%)	Super (%)
Static, single-link	0	27	68	5
Static, multilink	10	12	45	32
Dynamic, single-link	0	27	68	5
Dynamic, multilink	1	10	35	55

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

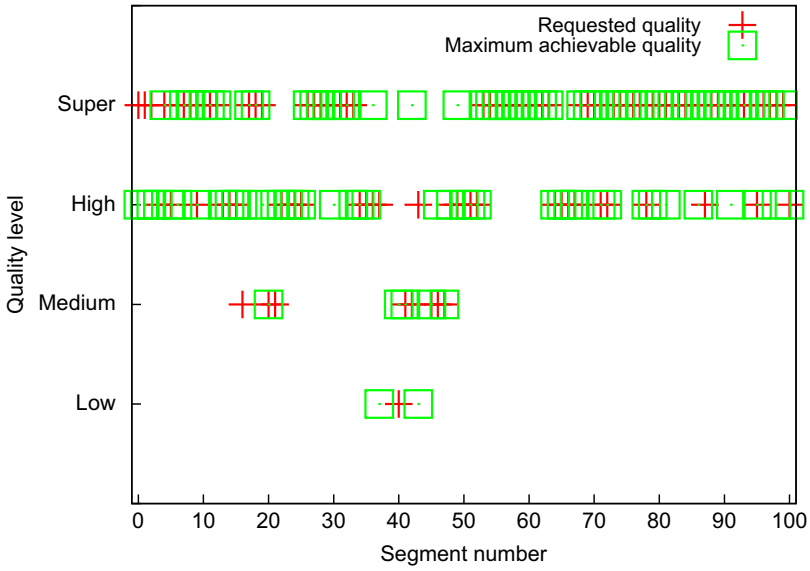

Fig. 16. Comparison of the actual and possible video quality for one live streaming without buffering experiment, real-world networks.

This figure compares the requested quality with the maximum possible quality, based on the measured throughput. With a few exceptions caused by the dynamic link behavior, the requested quality matched what the link could support. Similar observations were made for the other types of streaming as well.

In the future, we plan to look at shifting quality more intelligently, for example based on a history of the observed throughput. Even if adding a second link reduced the number of deadline misses in all the experiments performed for this paper, deadline misses still occurred (especially with live streaming without buffering). By adjusting quality less aggressively and thereby trading quality for smoothness, the number of misses should be even further reduced.

6. Conclusion

In this paper, we have presented and evaluated two different subsegment approaches for DASH-like video streaming. The approaches were implemented in the DAVVI streaming system, which offers the same core functionality as DASH, together with a request scheduler which retrieves video segments in several different bitrates for quality adaption over multiple heterogeneous network interfaces simultaneously. The static subsegment approach was based on our earlier work, presented in [2], and divides the segments into smaller fixed-sized subsegments to achieve efficient bandwidth aggregation. This increases performance compared to a single link and cause a significant increase in video quality. However, for the client to reach maximum performance with this approach, the buffer size has to be large enough to compensate for link heterogeneity.

To avoid the buffer requirement and allow quasi-live streaming at high quality, we developed a subsegment approach which calculates the sizes of the subsegments dynamically, based on the current interfaces' throughput. The two approaches were evaluated in the context of on-demand and live streaming with and without buffering (startup delay) over emulated and real networks. Only when the buffers were large enough to compensate for link heterogeneity, the static and dynamic subsegment approaches performed the same. In all the other scenarios, the dynamic subsegment approach was able to alleviate the buffer problem and showed similar performance independent of link heterogeneity for a given buffer size. Even though DAVVI [3] and DASH differ in how the represent a video, they offer the same functionality. Also, all the information needed to do DAVVI-like quality adaption and subsegmenting is present in DASH, and the actual data retrieval is identical. In other words, similar video quality gains would be seen with a multi-link-extended DASH access client.

In our future work, we plan to analyze how increasing or decreasing the duration of a segment affects quality decisions and the performance of the subsegment approaches. In addition, we want to look into tweaking the dynamic subsegment approach, e.g., by adding weights to different measurements and calculations, adapting quality more intelligently (for example based on a history of the observed throughput), experimenting with AVC-encoding and live streaming without buffering, and, as many multihomed devices are designed to run on battery, evaluate the effect bandwidth aggregation has on battery consumption.

References

[1] Ars Technica, US broadband's average speed: 3.9 Mbps, Technical Report, 2010 〈http://bit.ly/6TQROA〉 (Online).

[2] K. Evensen, T. Kupka, D. Kaspar, P. Halvorsen, C. Griwodz, Quality-adaptive scheduling for live streaming over multiple access networks, in: Proceedings of the 20th International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV'10, ACM, New York, NY, USA, 2010, pp. 21–26.

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007

[3] D. Johansen, H. Johansen, T. Aarflot, J. Hurley, A. Kvalnes, C. Gurrin, S. Zav, B. Olstad, E. Aaberg, T. Endestad, H. Riiser, C. Griwidz, P. Halvorsen, Davvi: a prototype for the next generation multimedia entertainment platform, in: Proceedings of the 17th ACM International Conference on Multimedia, MM'09, ACM, New York, NY, USA, 2009, pp. 989–990.
[4] MPEG DASH, ISO/IEC 23001-6 CD (N11578), 2011 〈http://bit.ly/a5rRJK〉 (Online).
[5] T. Stockhammer, Dynamic adaptive streaming over http: standards and design principles, in: Proceedings of the Second Annual ACM Conference on Multimedia Systems, MMSys'11, ACM, New York, NY, USA, 2011, pp. 133–144.
[6] Move Networks, Internet television: challenges and opportunities, Technical Report, Move Networks, Inc., 2008.
[7] Apple Inc., Mac OS X Server—QuickTime Streaming and Broadcasting Administration, 2007.
[8] A. Zambelli, IIS smooth streaming technical overview, Technical Report, Microsoft Corporation, 2009.
[9] P. Rodriguez, E.W. Biersack, Dynamic parallel access to replicated content in the internet, IEEE/ACM Transactions on Networking 10 (2002) 455–465.
[10] F. Wu, G. Gao, Y. Liu, Glitch-Free Media Streaming, Patent Application (US2008/0022005), 2008.
[11] A. Miu, E. Shih, Performance analysis of a dynamic parallel downloading scheme from mirror sites throughout the Internet, Technical Report, Massachusetts Institute of Technology, 1999.
[12] J. Funasaka, K. Nagayasu, K. Ishida, Improvements on block size control method for adaptive parallel downloading, International Conference on Distributed Computing Systems Workshops, vol. 5, 2004, pp. 648–653.
[13] B. Wang, W. Wei, Z. Guo, D. Towsley, Multipath live streaming via TCP: scheme, performance and benefits, ACM Transactions on Multimedia Computing, Communications, and Applications 5 (2009) 1–23.
[14] E. Biersack, W. Geyer, Synchronized delivery and playout of distributed stored multimedia streams, Multimedia Systems 7 (1999) 70–90.
[15] P. Ni, A. Eichhorn, C. Griwodz, P. Halvorsen, Fine-grained scalable streaming from coarse-grained videos, in: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV'09, ACM, New York, NY, USA, 2009, pp. 103–108.
[16] D. Kaspar, K. Evensen, P. Engelstad, A.F. Hansen, Using HTTP pipelining to improve progressive download over multiple heterogeneous interfaces, in: Proceedings of the IEEE International Conference on Communications, pp. 1–5.

Please cite this article as: K. Evensen, et al., Using bandwidth aggregation to improve the performance of quality-adaptive streaming, Signal Processing-Image Communication (2011), doi:10.1016/j.image.2011.10.007