# An Evaluation of Live Adaptive HTTP Segment Streaming Request Strategies

Tomas Kupka
University of Oslo
tomasku@ifi.uio.no

Pål Halvorsen
University of Oslo
Simula Research Laboratory
paalh@ifi.uio.no

Carsten Griwodz
University of Oslo
Simula Research Laboratory
griff@ifi.uio.no

***Abstract*—Nowadays, several live and on-demand streaming solutions use HTTP for signaling and data delivery. A frequently used technique is to chop a continuous stream into segments, encode these in multiple qualities and make these available for download using plain HTTP methods. This approach has become known as dynamic adaptive segment streaming over HTTP. Its advantage is that the deployed web infrastructure is easily reused, even for live segment streaming. In this case, however, it is not strictly bulk traffic. We show in this paper, that the streaming source is essentially an on-off source. Furthermore, this paper analyzes several client-controlled segment request strategies for live adaptive HTTP segment streaming. We present experimental results showing the benefits and drawbacks of each strategy with respect to achieved video quality, smoothness of playback and end-to-end delay. We show that it matters how clients request segments. The results indicate strongly that synchronization of client requests has a negative impact on router queues and leads to increased packet loss, and should thus be avoided to achieve a high goodput.**

***Index Terms*—live HTTP streaming, segment request strategy, Content Delivery Network**

## I. Introduction

Streaming over HTTP has become a popular solution for video delivery over the Internet. It runs on top of TCP, provides NAT friendliness and is allowed by most firewalls. It has been shown that video streaming over TCP [1] is possible as long as congestion is prevented and that HTTP streaming can scale to millions of users [2].

The general idea of adaptive HTTP segment streaming is to chop the original stream into segments and upload these to webservers. The video segments can then be downloaded like traditional web objects. In the case of live streaming, the segments are produced periodically, with a new segment becoming available shortly after it has been recorded and encoded completely. Furthermore, to cope with varying bandwidth, each segment may be encoded in multiple qualities (and thus bit rates). Bandwidth adaptation is a matter of downloading the segment in a different quality. The most prominent approaches today are Microsoft's Smooth Streaming [3], Move Networks [2], Adobe's HTTP Dynamic Streaming [4] and Apple's live HTTP streaming [5]. Moreover, it is under standardization by 3GPP and ISO as DASH [6].

The two major goals for *live* adaptive HTTP segment streaming are the maximization of the user perceived quality and the minimization of the time between video capturing and video presentation. We call this time the end-to-end (e2e) delay. It is the propagation delay from the source to the user's screen and can vary throughout a streaming session. It expresses the liveness of the stream.

In HTTP segment streaming, it is a client's responsibility to download the next segment before the previous segment is completely played out. This implies deadlines by which segments need to be ready. If a segment is not ready, a deadline miss occurs, and the playback stalls.

This paper builds on our earlier adaptive HTTP streaming systems [7]–[9]. Here, we investigate segment request strategies that clients can implement. No modifications to the server side are required. We investigate how client-side strategies *alone* affect the sharing of a bandwidth bottleneck and the resulting video quality and e2e delay. We consider a live video source and a webserver that handles client requests. We neglect the time it takes to encode and upload the segments to the server, because this time cannot be influenced by the request strategies.

Our scenario represents a building block for a larger distributed streaming architecture like for example the streaming from 2010 winter Olympics provided by Smooth Streaming [3]. We show in this paper that our results are also applicable to content delivery network (CDN) like networks. Moreover, our results indicate that a good strategy avoids network congestion by not synchronizing client requests, i.e., non-synchronized requests lead to higher video quality and smaller e2e delay.

## II. Client request strategies

In this section, we describe our model for live adaptive HTTP segment streaming and the options of a client-side request strategy. We also explain why we investigate only a subset of all possible client-side strategies.

### A. The segment streaming model

Figure 1 depicts the essentials of our model. At time $t_i$, a segment $i$ has been encoded, uploaded and made available. Like in several commercial systems, the segment playout duration [8] is constant and equals $t_{i+1} - t_i$. The letter **A** denotes the point in time when a user starts streaming. We call this time client arrival time.

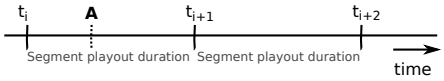



Figure 1. The segment streaming model

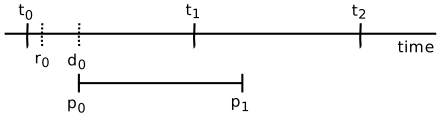



Figure 2. Immediate playout start with video skipping

### B. The option set of a strategy

We considered four important options for a segment request strategy.

*1) First request option ($P_{fr}$):* At client arrival **A**, the client requests segment $i$, which can be the latest available segment or the next that will become available, i.e., t = 0 or 1 in Figure 1. We do not consider $i < 0$ because it implies high e2e delays and $i > 1$ because it implies high start up latency (the client needs to wait until the segment becomes available).

*2) Playout start option ($P_{playout}$):* A client can start the playout immediately after the first segment is downloaded or delay the playout. To avoid high e2e delays already in the design of a request strategy, we limit the playout delay to at most one segment duration.

*3) Next request option ($P_{nr}$):* A client can request the next segment at several points in time. We consider two of them. A client can send the request some time before the download of the current segment is completed (download-based request) or send the next request some time $\epsilon$ before the playout of the currently played out segment ends (playout-based request).

*4) Deadline miss handling option ($P_{miss}$):* When a segment is downloaded after the previous segment has been played out completely (a deadline miss), the client can skip the first part of the downloaded segment equal to the deadline miss and keep the current e2e delay or start the playout from the beginning of the segment extending the e2e delay.

To reduce the complexity and make the results easily comparable, we decided to use only one quality adaptation algorithm. Here, the first segment is retrieved at lowest quality, and the quality of all other segments is based on the download rate of the previous segment and the time available until segment's deadline. To compensate for small bandwidth variations, the algorithm chooses the quality so that the download ends some time before the segment's deadline. We plan to investigate other algorithms in the future.

### C. Reduction of option combinations

Each option ($P_{fr}$, $P_{playout}$, $P_{nr}$, $P_{miss}$) has two possibilities, resulting in 16 possible combinations. However, some combinations can be eliminated.

Figure 2 illustrates a case when $P_{playout}$ is set to immediate playout and $P_{miss}$ to video skipping. Segment 0 is requested at $r_0$, and the download is finished at $d_0$. The playout starts at $p_0 = d_0$, and $p_0$ sets all consecutive deadlines ($p_1$, $p_2$, ..) implicitly. The time it takes to download the first segment which is downloaded at the lowest quality is given by $d_0 - r_0$. We see that $d_0 - r_0 \approx p_i - t_i$ which means that, under the same network conditions, the second segment can only be downloaded at the lowest quality, and the server is going to be idle between $d_1$ and $t_2$. This applies to all consecutive segments because $P_{miss}$ is set to video skipping. We therefore ignore combinations where $P_{miss}$ is set to video skipping and $P_{playout}$ is set to immediate playout.

By design, every deadline miss leads to an e2e delay longer than one segment duration, if $P_{playout}$ is set to delay the playout to the next $t_i$ and $P_{miss}$ is set to e2e delay extension. Because these options lead to a very high e2e delay, we ignore combinations that have $P_{miss}$ set to e2e delay extension and $P_{playout}$ set to playout delay.

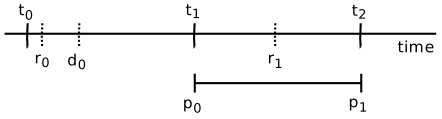



Figure 3. Delayed playout, video skipping and playout based requests

Figure 3 illustrates a scenario where $P_{nr}$ is based on the playout time, $P_{miss}$ is handled by video skipping, and $P_{playout}$ is delayed until the time next segment becomes available. All clients that download their first segment between $t_0$ and $t_1$ start their playout at $t_1$. With this, all consecutive deadlines are fixed at $t_i$, $i > 2$. All clients request the next segment at the same time ($P_{nr}$), e.g., $r_1$ in Figure 3. Consequently, the server is idle between $t_1$ and $r_1$. The wastage of resources is apparent, and therefore, we ignore combinations with these options.

Out of the 6 remaining strategies, 4 extend the e2e delay in case of a deadline miss ($P_{miss}$ set to e2e delay extension). 2 of these 4 strategies wait with the first request until a new segment becomes available ($P_{fr}$ set to delay the first request). Even though this leads to a very small initial e2e delay, it leads also to many deadline misses in the beginning of a streaming session, because the clients essentially synchronize their requests at $t_i$. This leads to bad performance, as we show later and the result are many user annoying playback stalls. We therefore do not further evaluate these two strategies.

We discuss the 4 remaining strategies in the following section, which we later evaluate with respect to the streaming performance.

### D. e2e delay-extending strategies

The two remaining e2e delay-extending strategies focus on keeping the e2e delay as small as possible, but do not delay the first request. We call them Moving e2e Delay Byte Based

Table I
EVALUATED STRATEGIES

| Strategy | $P_{fr}$ | $P_{playout}$ | $P_{nr}$ | $P_{miss}$ |
|---|---|---|---|---|
| $MoBy$ | immed. | immed. | download b. | extend e2e delay |
| $MoVi$ | immed. | immed. | playout b. | extend e2e delay |
| $CoIn$ | immed. | delayed | download b. | skip video |
| $CoDe$ | delayed | delayed | download b. | skip video |

Requests ($MoBy$) and Moving e2e Delay Playout Based Requests ($MoVi$).

$MoBy$ is illustrated in Figure 4. A client requests the latest segment that is available at the time of its arrival **A**, i.e., segment $t_0$ in Figure 4. The playout starts immediately after the downloaded is finished, $p_0 = d_0$. The next segment request is sent to the server when all but $link\ delay * link\ bandwidth$ bytes of the currently downloaded segment are fetched. This pipelining of requests ensures that the link is fully utilized [10]. Please note, that in Figure 4, the next segment is not available at a time that would allow request pipelining. Therefore, it is requested later when it becomes available[1].

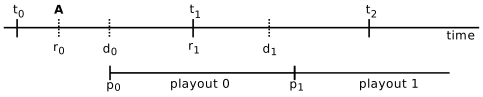



Figure 4. Strategy $MoBy$

Figure 5 illustrates the $MoVi$ strategy. The only difference between $MoVi$ and $MoBy$ is that $MoVi$ sends the next request when there are $\frac{p_1 - p_0}{2}$ seconds of playout left. This leads to a more evenly distributed requests over $[t_i, t_{i+1}]$ as shown in the result section.

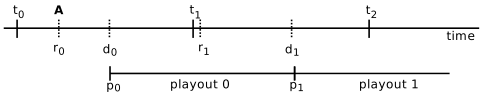



Figure 5. Strategy $MoVi$

### E. Constant e2e delay strategies

The last two strategies of the remaining 4 strategies keep a constant e2e delay of one segment duration throughout a streaming session. If a deadline is missed, the first part of the segment equal to the deadline miss is skipped. Both of these strategies request a segment when it becomes available and their request synchronization leads, in theory, to even bandwidth distribution among all clients. We call these two strategies Constant e2e Delay Immediate Request ($CoIn$) and Constant e2e Delay Delayed Request ($CoDe$).

Strategy $CoIn$ is illustrated in Figure 6. A client first requests the latest segment on the server at $r_0 = A$. The next segment is requested at $t_1$. The e2e delay is fixed and equals $t_{i+1} - t_i$, i.e., a segment that becomes available at $t_i$ is presented at $t_{i+1}$. This means that the client has $t_{i+1} - t_i$ seconds to download the next segment.

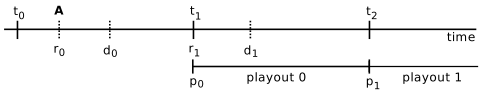



Figure 6. Strategy $CoIn$

Figure 7 illustrates strategy $CoDe$. The difference between $CoIn$ and $CoDe$ lies in the first request. $CoDe$ delays the first request until the next segment becomes available, i.e., $r_1 \neq A$. $CoDe$ forces all clients to send their requests at the same time starting with the first segment (in contrast to $CoIn$).

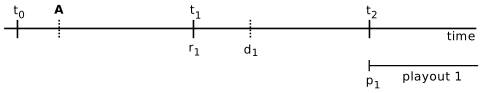



Figure 7. Strategy $CoDe$

All the strategy properties are summarized in Table II. These properties are only valid when no deadline misses occur.

Table II
STRATEGY SUMMARY ($s$ = SEGMENT DURATION)

| Strategy | Startup delay | e2e delay | Available download time |
|---|---|---|---|
| $MoBy$ | $d_0 - t_0$ | $d_0 - t_0$ | $d_0 - t_0$ |
| $MoVi$ | $d_0 - t_0$ | $d_0 - t_0$ | $\leq 0.5s$ |
| $CoIn$ | $\mu = 0.5 \times s$ | $s$ | $s$ |
| $CoDe$ | $\mu = 1.5 \times s$ | $s$ | $s$ |

## III. EMULATION SETUP

Since large-scale real-world experiments require both a lot of machines and users, we performed experiments in our lab using an emulated network with real networking stacks. Our hardware setup is shown in Figure 8. We used Linux OS (kernel v2.6.32) with *cubic* TCP congestion control with SACK and window scaling on. For traffic shaping, we used the HTB qdisc with a *bfifo* queue on the egress interfaces of the client and server machine. The queue size was set to the well known rule-of-thumb $RTT * bandwidth$ [11]. The queue represented a router queue in our approach. The client machine emulated requests from clients and the server machine emulated a webserver[2].

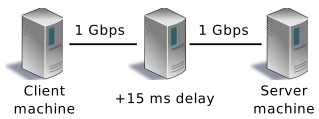



Figure 8. Emulation setup

[1]Segment availability can be checked by consulting a tracker file [8].

[2]All data was served from memory to avoid non-networking factors like disk speed.

For video quality, we used low, medium, high and super segment qualities with segment sizes of 100KB, 200KB, 300KB and 400KB, respectively. The segment duration was 2 seconds as proposed by [8], [12].

We ran two sets of experiments. The first set, called *short sessions scenario*, included 1000 clients, each downloading 15 segments, which equals approximately the duration of a short news clip. The client interarrival time was modeled as a Poisson process with 100 ms interarrival time (equals 600 client arrivals per minute). The second set included 300 clients each downloading 80 segments, which represent an unbounded live stream. We used the same Poisson process. This set is called *long sessions scenario*. To examine the influence of arrival time, we reran all experiments also with constant interarrival time of 100 ms.

The server served a peak of about 300 concurrent clients in both scenarios. The short sessions scenario was limited by the interarrival time of 100ms and the number of segments per client. The long sessions scenario was limited by the number of clients.

We evaluated four bandwidth limitations restricting the maximal quality clients could download for both scenarios: 40 MB/s, 55 MB/s, 65 MB/s and 80 MB/s. We repeated each experiment 20 times.

Our results are based on stable system state. We consider the system to be in a stable state when the number of active clients is about constant, i.e., client arrival rate equals client departure rate. For the short sessions scenario, this means examining segments 30 to 50, and for the long sessions scenario, segment 20 and beyond.

## IV. Results and analyses

We were interested in the interaction of concurrent segment downloads and its impact on video quality and e2e delay. We therefore analyzed the goodput (and the corresponding video quality) of each strategy as well as the parallelism induced by each strategy.

### A. Strategy goodput

Figure 9 and Figure 11 show the collected goodput statistics for short and long sessions scenarios. Each box plot value[3] represents the sum of bytes received by all clients during one experiment divided by the stream's duration.

The goodput is, like a regular bulk download, reduced by the protocol header overhead, and additionally by the time the client spends waiting until the next segment becomes available. For short sessions, Figure 13[4] shows the number of concurrently active clients over time. We see that the number of clients drops (clients are idle waiting) towards the end of each 2 second interval. We further discuss the effects and consequences in the section on parallelism.

[3]Minimum, maximum, quartiles, median and outliers are shown. Please note that in many cases, these overlap in the graphs.

[4]The same pattern was observed also for long sessions scenario.

The goodput of strategies $CoIn$ and $CoDe$ increases as more bandwidth becomes available. The reason for this behaviour is that these strategies keep a constant e2e delay and so provide a full segment duration of time for segment download.

Figure 14 and Figure 15 show the improvement in terms of segment quality. The height of the graph corresponds to the total number of segments downloaded. The number of segments of each quality is represented by the grey scale starting at the top with the lightest color representing the super quality. The trend is clear, i.e., the more bandwidth the higher the quality.

Strategies $MoBy$ and $MoVi$ do not follow the same trend as $CoIn$ and $CoDe$. $MoBy$ and $MoVi$ increase the goodput up to the bandwidth limitation of 55MB/s. The goodput increases slowly or stagnates for higher bandwidths. We can observe the same for segment quality. Figure 16 and 17 indicates that these two strategies are trading goodput for smaller e2e delay, i.e., the stream is more "live".

It is clear from Figure 14 and especially Figure 15 that strategy $MoVi$ is able to achieve high quality quickly and still keep a small e2e delay. The reason lies in parallelism, which we discuss in the next section.

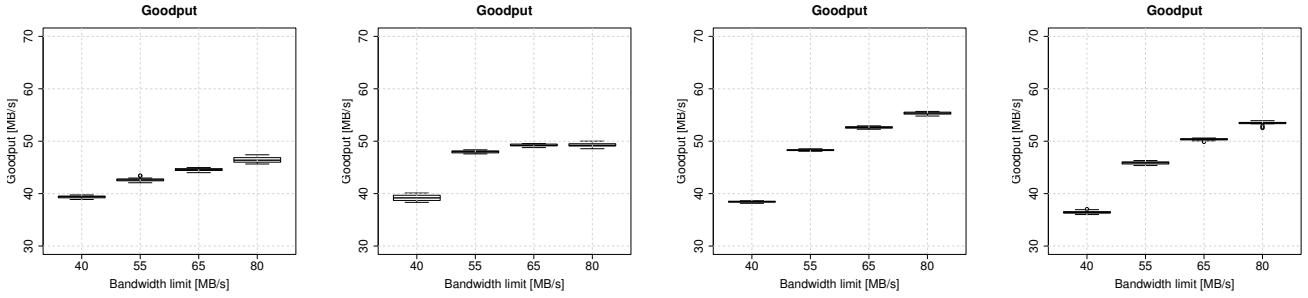
### B. Parallelism and its consequences

Figure 13 shows a representative 20 second snapshot of the number of concurrently active clients. The strategy that stands out is $MoVi$. The maximum number of concurrently downloading clients is about one third compared to the other strategies. Yet, as discussed in Section IV-A, $MoVi$ does not suffer severe quality degradation (and it does not discard segments).

The mixture of $P_{nr}$ based on playout and $P_{playout}$ set to immediate playout makes $MoVi$ requests distributed according to client arrival distribution, e.g., the requests are distributed over each interval. This is an important difference to the other strategies.

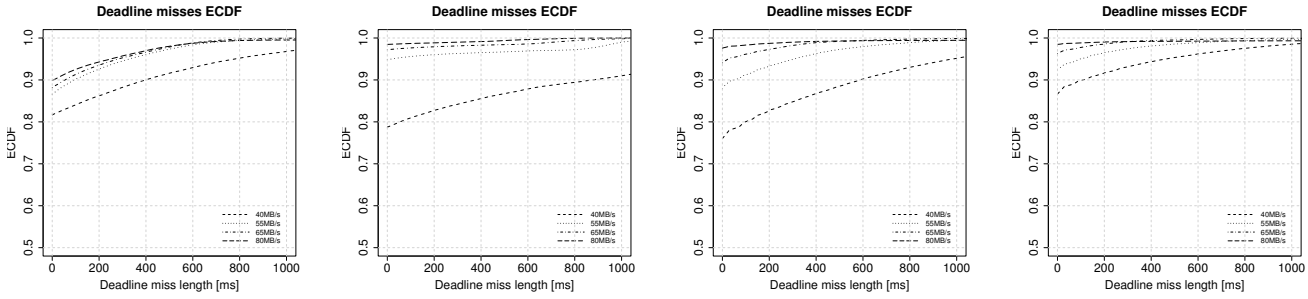
We found that the reason for $MoVi$'s good overall performance is the number of dropped packets by the shaping layer. Figure 12 shows a representative plot of the number of packets dropped by the emulated router queue. The queue size is set to $RTT*bandwidth$ and accepts therefore only a certain number of packets. We observed that synchronized requests overflow the router queue much more often than requests distributed over time as is the case for $MoVi$.

Because the $P_{nr}$ option of $MoVi$ is based on playout time, more specifically a client requests the next segment $\frac{p_1-p_0}{2}$ seconds before the playout of the previous segment ends (Section 2.3), $MoVi$ clients have at most half of the time available to download a segment compared to $CoIn$ and $CoDe$. Yet, they are still able to download the same or higher quality segments (Figure 14 and 15). Moreover, the e2e delay is in many cases lower than the segment duration (Figure 16 and 17). The only time the strategy has a problem is when there is not enough bandwidth available to sufficiently distribute client downloads and the downloads start to overlap too much.



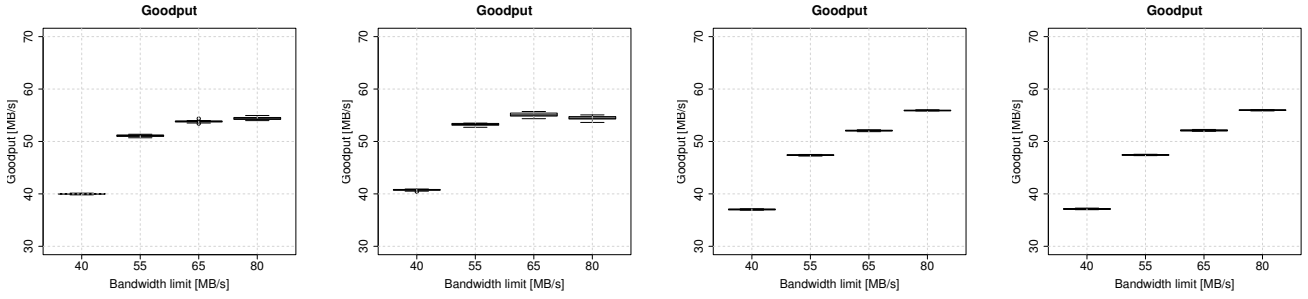

(a) MoBy (b) MoVi (c) CoIn (d) CoDe

Figure 9. Short sessions scenario goodput




(a) MoBy (b) MoVi (c) CoIn (d) CoDe

Figure 10. Short sessions scenario deadline misses' empirical distribution function (ECDF)




(a) MoBy (b) MoVi (c) CoIn (d) CoDe

Figure 11. Long sessions scenario goodput

Figure 18 shows a box plot of segment download speed as experienced by the clients. The graph includes all bandwidths experienced by all clients for every segment in every experiment run. Because of the small number of concurrent clients, $MoVi$ is able to make more out of the available bandwidth. The other strategies loose more data in competition for space in the router queue.

If a $MoVi$ client determines that enough time is available for downloading higher quality segments, it probes by downloading a higher quality segment at the next opportunity. It then experiences more competition because it enters the download slots of other clients, and withdraws immediately if there is not enough bandwidth. This approach reduces the number of concurrent clients and thus, the pressure on the router. The other strategies can never probe. The requests of active clients are always synchronized. The increased competition leads to more unpredictable bandwidth distribution among clients and makes it harder for the clients to estimate available bandwidth based on the download of the previous segment.

The deadline-miss graphs in Figure 10 (long sessions scenario results are very similar) indicate that the $MoVi$ strategy is able to estimate the bandwidth as well or better than the other strategies.

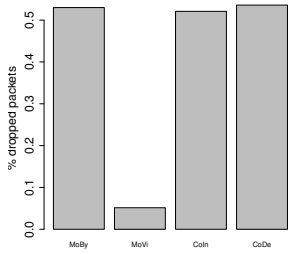
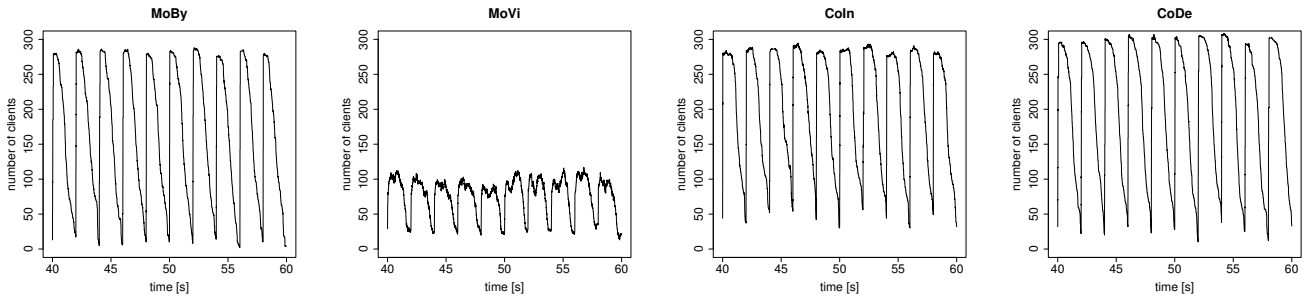
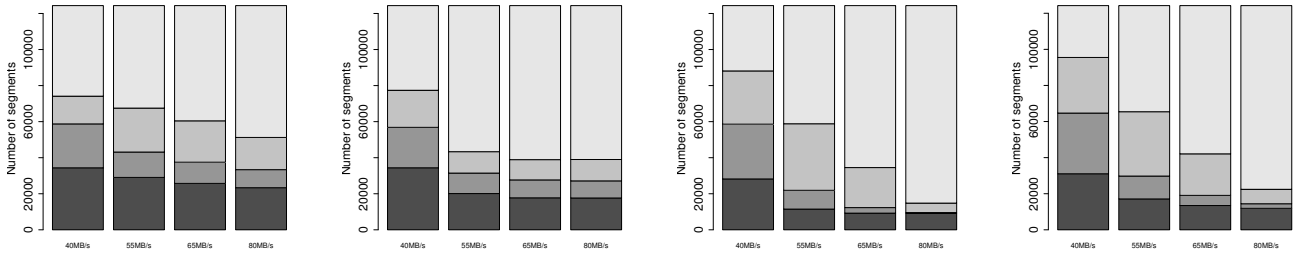



Figure 12. Number of packets dropped by the emulated router queue for 55MB/s bandwidth limitation



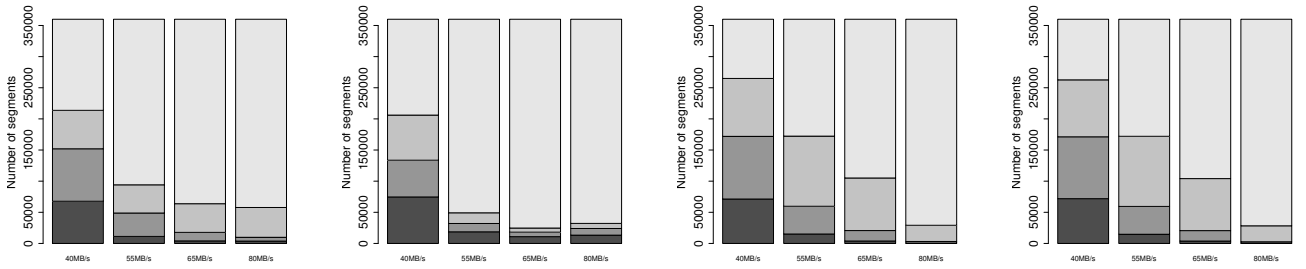

(a) MoBy (b) MoVi (c) CoIn (d) CoDe

Figure 13. Concurrent downloads in the short sessions scenario (55MB/s)



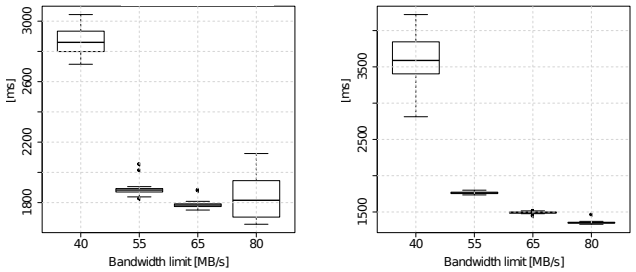

(a) MoBy (b) MoVi (c) CoIn (d) CoDe

Figure 14. Short sessions quality distribution of downloaded segments (from super quality at the top to low quality at the bottom)



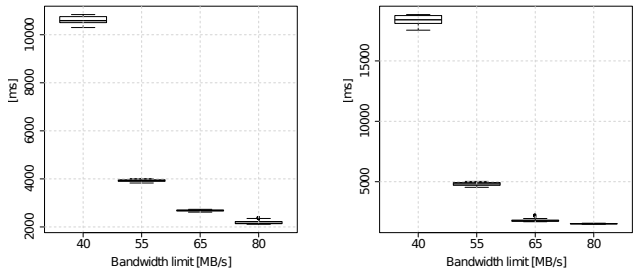

(a) MoBy (b) MoVi (c) CoIn (d) CoDe

Figure 15. Long sessions quality distribution of downloaded segments (from super quality at the top to low quality at the bottom)




(a) MoBy (b) MoVi

Figure 16. Short sessions scenarios e2e delay (note: e2e delay y-axes have different scale)

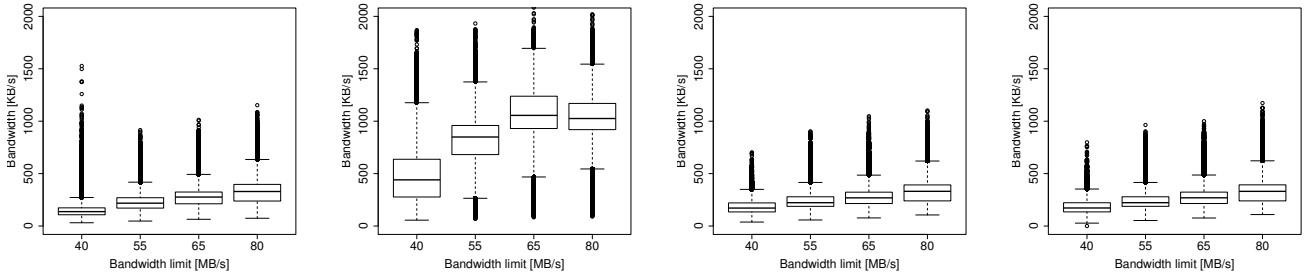



(a) MoBy (b) MoVi

Figure 17. Long sessions scenarios e2e delay (note: e2e delay y-axes have different scale)

### *C. Deadline misses and the bandwidth fluctuations*

There is, theoretically, only one factor that influences the deadline misses, i.e., bandwidth fluctuation. Under perfect conditions without bandwidth fluctuations, clients never overestimate the bandwidth available, and a deadline miss never occurs. However, the large number of concurrent streams




(a) MoBy (b) MoVi (c) CoIn (d) CoDe

Figure 18. Client segment download rates in the long sessions scenario

combined with TCP congestion control creates fluctuations. In the case of $MoVi$, the fluctuations do not have such a big impact because the number of concurrent clients is small (Figure 13), and the download bandwidths are high compared to the other strategies.

The number of deadline misses shown in Figure 10 indicates that the bandwidth estimation, and therefore also the quality adaptation, is not always perfect. However, we see that with more bandwidth available, the number of deadline misses decreases.

We believe that the reason for deadline miss improvement is the usage of time safety. The adaptation algorithm chooses segment quality so that the download ends at least $ts$ seconds before segment's deadline. The minimal value of $ts$ is the time safety. A deadline miss occurs only if the download time proves to be longer than the estimated download time plus the time safety. To make the influence of the time safety on results as small as possible we used a small value of 150ms for all scenarios. However, the number of bytes that can be downloaded within the time safety increases with available bandwidth. This way the safety basically grows with bandwidth, which results in fewer deadline misses as the bandwidth grows.

### D. Influence of client interarrival times distribution

We reran short and long session scenarios with constant client interarrival times. Our results showed no major differences. We observed very similar results for segment quality distribution, number of concurrent clients over time, e2e delay and deadline misses. Because of space limitations, we do not show any figures in this section.

## V. Discussion

### A. And the winner is...

Among the usable strategies we selected for our experiments in Section II, the question is which is the best? Strategy $CoDe$ results in good quality and few deadline misses in low bandwidth experiments. However, the synchronization of requests ($P_{fr}$ and $P_{miss}$) also leads to high concurrency which leads to bandwidth wastage due to congestion, as explained in the previous section.

Our results show that strategy $MoVi$ leads to much less concurrency. $MoVi$ is able to more evenly distribute requests over time due to $P_{nr}$ based on playout. This results in higher quality and a smaller e2e delay. Thus, we believe that $MoVi$ presents the best mix of options and any change of its options would lead to worse results. We can also generally conclude that synchronization of requests is not a good idea and should be avoided.

### B. Implications for streaming with buffering

We have shown in this paper that in order to reduce problems caused by congestion, it is better to distribute client requests over time. This reduces the number of concurrent downloads and increases the goodput.

One of the goals of our experiments was to have a small e2e delay. Nevertheless, our results also apply for live streaming that uses buffers. Imagine a flash crowd scenario in which all client requests come almost at the same time. After some initial time all clients have filled their buffers and start to synchronize their requests. According to our results if this is not avoided, a shared router queue is likely to overflow and a degradation of user experience is to be expected.

### C. Server utilization estimation

Estimating server utilization is not a straightforward process for adaptive HTTP segment streaming. Figure 19 shows a plot of server bandwidth as it was recorded every 100 ms throughout one short sessions experiment.

The figure also shows a moving average value of the bandwidth with a history window of 20 seconds. For about the first 30 seconds, the average goes up. After that, the client departure rate reaches the client arrival rate and the average is almost constant. The total number of active clients only decrease towards the end of the experiment, therefore, the average goes down again.

It is interesting to consider if the stable state (and so our scenario) would also be reached with domain name system (DNS) load balancing in place. This would only be the case if clients are not redirected before the stable state is reached.

The maximum of 300 concurrent clients is reached at around second 30. At that point, the bandwidth utilization is about 50% measured by the moving average. This leads to the conclusion that our results are also applicable to DNS load balancing if the utilization threshold for redirection is above 50% (Changing the window size of the moving average could

help, but only if clients can be assumed not to share the same local DNS server. Nevertheless, the possibility of assigning all clients to one server exists).

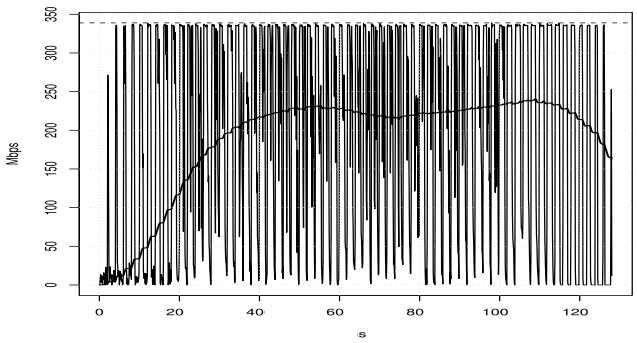



Figure 19. Server bandwidth consumption with average history window of 20 seconds. On the x-axes is the time and on the y-axes the bandwidth. (Strategy $MoBy$ with 40MB/s bandwidth limit)

## VI. Implications for multi-server scenarios

Here, we discuss implications of our findings for CDNs based on DNS load balancing. Figure 20 shows the principle of DNS load balancing CDN. When a client wants to download a segment, it issues a request to its local DNS server. If the information is not cached, the request is forwarded to the root DNS server. The root DNS server redirects the local DNS server to a CDN provider's DNS server. This server finds the best server available and replies with the server's IP address. The local DNS server caches and forwards the information to the client. The client downloads the data from the provided server.

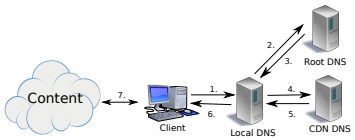



Figure 20. CDN DNS load balancing

A small time to live (TTL) of the DNS cache entry leads to frequent DNS requests. On the other hand, a large TTL leads to less load balancing opportunities for the CDN. Therefore, a compromise between granularity of load balancing and the DNS server load is required, e.g., Akamai CDN uses a default value of 20 seconds [13]. In fact, the TTL value is not enforcible, and providers might choose a higher value in order to provide faster response times or to reduce the risk of DNS spoofing [14].

DNS load balancing relies on separate connections for each web object. However, separate connections are inefficient in the case of live HTTP segment streaming. To save the overhead associated with opening a new TCP connection, it is reasonable for a client to open a persistent TCP connection to a server and reuse it for all segment requests. Requests are sent over this connection as segments become available. This differs from a persistent connection used by web browsers. Firstly, the connections last longer (a football match takes 90 minutes). Secondly, clients do not download data continuously (Figure 13). This makes it difficult to estimate the server load by algorithms that assume that all clients download data continuously, e.g., moving average. Moreover, the adaptive part of the streaming leads to jumps in bandwidth requirements rather than to smooth transitions. These are all reasons that make it harder for conventional DNS load balancing to make an accurate decision. An HTTP redirection mechanism in the data plane could solve this. However, to the best of our knowledge, this mechanism is not implemented by current CDNs. The insufficiencies of server-side load balancing for live adaptive segment streaming can be overcome by client-side request strategies as described in this paper.

## VII. Conclusion

Our evaluation of client-side request strategies for live adaptive HTTP segment streaming shows that the way the strategy requests segments from a server can have a considerable impact on the success of the strategy with respect to bandwidth utilization and achieved video quality. The chosen strategy influences not only the video quality and e2e delay, but also the efficiency of bandwidth usage.

Starting with the assumption that it is desirable for clients to request video segments from a server that provides live content as soon as possible after their availability, we examined three strategies that do this and a fourth that doesn't. The behavior of the synchronized strategies leads to synchronous requests and server responses. The unsynchronized strategy does not initiate downloads based on segment availability but on the playtime of earlier segments.

While it was obvious that synchronized requests lead to competition for bandwidth, we have shown that this competition leads to a severe amount of bandwidth wastage. Synchronized clients can only base their adaptation decision on average goodput in a complete segment length, because they can never avoid competition with other clients. Using strategies that do not synchronize requests, on the other hand, bandwidth wastage is avoided because the number of concurrently active clients is much lower. Clients can probe for higher qualities, and withdraw when their bandwidth demands increase the amount of competition and therefore the segment download time. We have finally considered how these observations could help scheduling decisions in a multi-server scenario.

In our future work, we plan to evaluate our scenario with TCPs that detect congestion by packet delay (e.g. Vegas, Compound TCP) and could therefore prevent congestion situations. Our ultimate goal is to extend our scenario to multiple servers and look at a possible coexistence of our strategies with other web traffic.

## VIII. Acknowledgements

This work has been performed in the context of the iAD (Information Access Disruptions) centre for Research-based Innovation funded by Norwegian Research Council, project number 174867.

## REFERENCES


[1] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via TCP: an analytic performance study," in *ACM MM*, 2004.

[2] "Move Networks," 2009. [Online]. Available: http://www.movenetworks.com

[3] "SmoothHD," 2009. [Online]. Available: http://www.smoothhd.com

[4] "HTTP dynamic streaming on the Adobe Flash platform," 2010. [Online]. Available: http://www.adobe.com/products/-httpdynamicstreaming/pdfs/httpdynamicstreaming_wp_ue.pdf

[5] R. Pantos (ed), "HTTP Live Streaming," 2009. [Online]. Available: http://www.ietf.org/internet-drafts/draft-pantos-http-live-streaming-00.txt

[6] T. Stockhammer, "Dynamic adaptive streaming over HTTP - standards and design principles," in *ACM MMSys*, 2011, pp. 133–144.

[7] H. Riiser, P. Halvorsen, C. Griwodz, and D. Johansen, "Low overhead container format for adaptive streaming," in *ACM MMsys*, Feb. 2010.

[8] D. Johansen, H. Johansen, T. Aarflot, J. Hurley, Å. Kvalnes, C. Gurrin, S. Sav, B. Olstad, E. Aaberg, T. Endestad, H. Riiser, C. Griwodz, and P. Halvorsen, "DAVVI: A prototype for the next generation multimedia entertainment platform," in *ACM MM*, Oct. 2009, pp. 989–990.

[9] K. R. Evensen, T. Kupka, D. Kaspar, P. Halvorsen, and C. Griwodz, "Quality-adaptive scheduling for live streaming over multiple access networks," in *NOSSDAV*, 2010, pp. 21–26.

[10] P. Rodriguez and E. W. Biersack, "Dynamic parallel access to replicated content in the Internet," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 455–465, 2002.

[11] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *Comput. Commun. Rev.*, vol. 34, no. 4, pp. 281–292, 2004.

[12] P. Ni, A. Eichhorn, C. Griwodz, and P. Halvorsen, "Fine-grained scalable streaming from coarse-grained videos," in *NOSSDAV*, Jun. 2009.

[13] A. jan Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind Akamai (travelocity-based detouring)," in *ACM SIGCOMM*, 2006, pp. 435–446.

[14] A. Hubert and R. van Mook, "Measures to prevent DNS spoofing," 2007. [Online]. Available: http://tools.ietf.org/html/draft-hubert-dns-anti-spoofing-00