Analysis of a Real-World HTTP Segment Streaming Case

Tomas Kupka, Carsten Griwodz, Pål Halvorsen
University of Oslo & Simula Research Laboratory
{tomasku,griff,paalh}@ifi.uio.no

Dag Johansen
University of Tromsø
dag@cs.uit.no

Torgeir Hovden
Comoyo
torgeir@comoyo.com

ABSTRACT

Today, adaptive HTTP segment streaming is a popular way to deliver video content to users. The benefits of HTTP segment streaming include its scalability, high performance and easy deployment, especially the possibility to reuse the already deployed HTTP infrastructure. However, current research focuses merely on client side statistics like for example achieved video qualities and adaption algorithms. To quantify the properties of such streaming systems from a service provider point of view, we have analyzed both sender and receiver side logging data provided by a popular Norwegian streaming provider. For example, we observe that more than 90% of live streaming clients send their requests for the same video segment with an inter-arrival time of only 10 seconds. Moreover, the logs indicate that the server sends substantially less data than is actually reported to be received by the clients, and the origin server streams data to less clients than there really are. Based on these facts, we conclude that HTTP segment streaming really makes use of the HTTP cache infrastructure without the need to change anything in the parts of Internet that are not controlled by the streaming provider.

Categories and Subject Descriptors

C.4 [**Performance of systems**]; D.2.8 [**Software Engineering**]: Metrics—*performance measures*; H.3.3 [**Information Search and Retrieval**]

General Terms

Measurement, Performance

Keywords

Live adaptive HTTP segment streaming, streaming, Microsoft Smooth streaming, streaming performance

EuroITV'13, June 24–26, 2013, Como, Italy.

1. INTRODUCTION

The number of video streaming services in the Internet is rapidly increasing. The number of videos streamed by these services is on the order of tens of billions per month where YouTube alone delivers more than four billion video views globally every day [21]. Furthermore, with only a few seconds delay, many major (sports) events like NBA basketball, NFL football and European soccer leagues are streamed *live*. For example, using the modern adaptive HTTP segment streaming technology, events like the 2010 Winter Olympics [22], 2010 FIFA World Cup [14] and NFL Super Bowl [14] have been streamed to millions of concurrent users over the Internet, supporting a wide range of devices ranging from mobile phones to HD displays.

As a video delivery technology, streaming over HTTP has become popular for various reasons. For example, it is scalable, runs on top of TCP, provides NAT friendliness and is allowed through most firewalls. The idea of splitting the original stream into segments and upload these to web-servers in multiple qualities (bitrates) for adaptive delivery has been ratified for an international standard by ISO/IEC as MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [19]. From the users' perspective, the video segment bitrate is selected based on observed resource availability and then downloaded like traditional web objects. Such an approach where video is streamed over TCP has been shown to be effective as long as congestion is avoided [20], and adaptive HTTP streaming has been proved to scale to millions of concurrent users [1]. Such a solution is therefore used by Microsoft's Smooth Streaming [22], Adobe's HTTP Dynamic Streaming [5] and Apple's live HTTP streaming [17] – technologies which are used in video streaming services from providers like Netflix, HBO, Hulu, TV2 sumo, Viaplay and Comoyo.

Current research has focused on various aspects of efficient client side delivery [6, 9, 15, 18]. To quantify the properties of such streaming systems from a service provider point of view, we have analyzed both sender and receiver side logging data provided by the Norwegian streaming provider Comoyo [7]. In this paper, we focus on live streaming events. In such scenarios, the segments are produced and made available for download periodically, i.e., a new segment becomes available as soon as it has been recorded, completely encoded and uploaded to a web-server, possibly located in a content distribution network (CDN) like Akamai or Level 3 [13]. The question that we answer in this paper is how adaptive HTTP segment streaming systems behave as a whole from a service provider point of view. For example, we observe that

for live streaming, about 90% of clients download the same video segment within a 10 second period, i.e., 90% of the users are at most 10 seconds behind the most "live" client. Furthermore, we find that the clients report receiving much more video segments than the server actually sent. Based on the client IP distribution and the IP distribution as seen by the server, we conclude that HTTP cache proxies must be heavily used between the origin server and the clients. This proves for example that the existing HTTP infrastructure is efficiently used and offloads the origin server in real world networks.

The rest of this paper is organized as follows: In Section 2, we discuss examples of related work. Section 3 sketches the network setup used by Comoyo [7]. Section 4 through 6 analyse the provided logs. The analysis is split into client log analysis in Section 5 and into server log analysis in Section 6. We conclude the paper in Section 7, where we also discuss future work.

2. RELATED WORK

The popularity of HTTP segment streaming led to many real world studies for example, Müller et al. [15] collected data using different commercial streaming solutions as well as the emerging MPEG-DASH standard over 3G while driving a car. They compared the average bitrate, number of quality switches, the buffer level and the number of times the playout was paused because of re-buffering. A similar study was performed by Riiser et al. [18]. In their paper, they evaluated commercial players when on the move and streaming over 3G. They additionally proposed a location aware algorithm that pre-buffers segments if a network outage is expected on a certain part of a journey (based on previous historical records for that location). Houdaille et al. [9] focused on a fair sharing of network resources when multiple clients share the same home gateway and Akhshabi et al. [6] compared rate adaption schemes for HTTP segment streaming using synthetic workloads.

However, in general, studies, that we are aware of, focus all on the client side. In other words, they measure the user experience in one form or an other from the client side perspective. We believe this is mainly because the commercial content providers are protective of their data, and it is not easy to get any statistics or logs. However, the server-side performance is equally important in order to scale the service to 1000s of users. In this respect, we have a collaboration with a commercial streaming provider Comoyo [7], and we present the insights gained from both server and client side log analysis in this paper.

3. COMOYO'S STREAMING SETUP

Comoyo [7] is a Norwegian streaming provider, providing access to an extensive movie database of about 4000 movies. Additionally, Comoyo provides live streaming of various events like the Norwegian premier league. Most of the soccer games are played on Sundays and some on Wednesdays, and hence, these two days are the busiest times for football live streaming. The logs that are analysed in this paper are from a Wednesday, but first, we shall look at Comoyo's network infrastructure.

The network infrastructure for both on-demand and live services is illustrated in Figure 1. The difference is only in the source of the stream, which can be a live stream coming in over a satellite or a pre-recorded stream coming from the movie database. The on-demand movies are streamed either using the classic progressive streaming based on Windows Media Video (wmv) or via Microsoft's Smooth streaming [22] based on HTTP segment streaming, which is also used for live streaming. For live streaming, the video content is available in 0.5, 1, 2 and 4 MBit/s, and the audio in 96 Kbit/s.

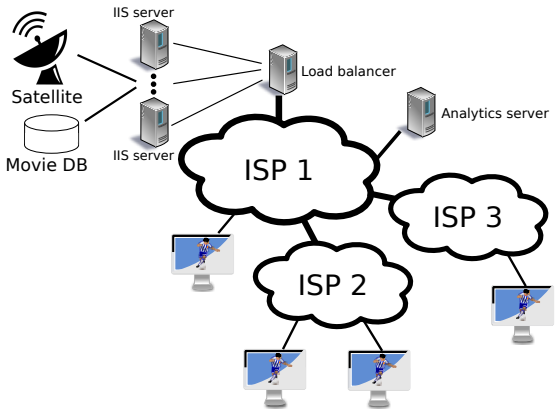


Figure 1: Streaming network infrastructure

Since, we focus only on Smooth streaming in this paper, we shall describe it briefly here. Smooth streaming [22] is a streaming technology by Microsoft. The file format used is based on ISO/IEC 14496-12 ISO Base Media File Format [2]. The reason for choosing it as its base is that it natively supports fragmentation. There are two formats specified, the disk file format and the wire file format. Each video (there is one for each bitrate) is stored as a single file in the disk file format. Additionally to the files containing the video data there are two more files on the server. One of these files (the *.ism* file) describes the relationship between the media tracks, bitrates and files on the disk. This file is only used by the server. The other file is for the client. It describes the available streams (codec, bitrates, resolutions, fragments, etc.) and it is the first file that the client requests to get information about the stream.

The Smooth streaming clients use a special URL structure to request a video fragment. For example, a client uses the following URL to request a SOCCER stream fragment that begins 123456789 time units from the content start[1] in a 1 Mbit/s bitrate from comoyo.com:

```
http://comoyo.com/SOCCER.ism/QualityLevels(1000000)/
Fragments(video=123456789)
```

The server parses the URL and extracts the requested fragment from the requested file. The official server that has these capabilities is the IIS server [16] with an installed Live Smooth Streaming extension [3]. It looks up in the *.ism* file the corresponding video stream and extracts from it the requested fragment. The fragment is then send to the client. Note that the fragments are cacheable since the request URLs of the same fragment (in the same bitrate) from two different clients look exactly the same and as such can be cached and delivered by the cache without consulting the origin server.

[1]The exact unit size is configurable, but is usually 100ns.

As Figure 1 shows, a number of IIS servers is deployed behind a load balancer that distributes the incoming requests. The load balancer is connected directly to the Internet, i.e., ISP 1 in the figure. However, content is not served exclusively to subscribers of ISP 1, but also to subscribers of other ISPs as illustrated in the figure. Besides the IIS servers; an analytics server is deployed and connected to the Internet. This server's role is to collect various information from the video clients. For example, the clients notify the analytics server when the video playout is started, stopped or when the quality changes.

4. LOG ANALYSIS

For our study, we were provided with two types of logs. We call these the server log and the client log based on where the information is collected and logged. We picked one server log and the corresponding client log from the same day, 23.05.2012. These logs were found representative for similar live streaming events. The logs are 24-hour logs collected for 8 soccer games. The server log contains information about every segment request as received by the load balancer. The most important information is the time of the request, the request URL, client's IP address, response size and streaming type, which can be progressive streaming, on-demand Smooth streaming or live Smooth streaming. Table 1 lists all the other important information that is logged about each request.

The client log is a collection of events collected from all the clients by the analytics server while the clients are streaming. Collected events are described in Table 2. The information logged for every event is described in Table 3. Specifically, every client event includes the information to which user in which session and at what time the event happened. Note that within a session a user can only stream one content, and a session can not be shared between multiple users. A typical sequence of events in one session is shown in Figure 2. In our analysis, we first analyse the logs separately and then point out the differences and inconsistencies and draw conclusions.

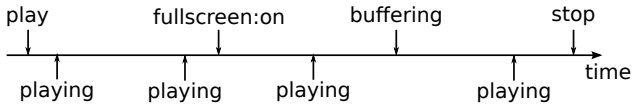


Figure 2: An example of events sent within a session

5. SERVER LOG ANALYSIS

Our analysis of the server log showed that there were 1328 unique client IP addresses in the log over the tracked 24-hour period. From all the IP addresses, 56% was tagged as live Smooth streaming, 6% as on-demand Smooth streaming and 44% as progressive streaming. 6% of all IP addresses was associated with more than one type of streaming, e.g., showed up one time as live Smooth streaming and another time as progressive streaming. In other words, if each user had a unique IP address there would had been 744 people watching the live stream.

Client IP	Client's IP address
HTTP method	HTTP method used (e.g. GET)
Request URL	The URL of the request
Response size	Response size in bytes
Useragent	Client's user agent
Cookie	Cookie information in the request
HTTP status	Status code of the response
Time taken	Time taken by the server in ms
Streaming type	The type of streaming. Possible values are progressive streaming, on-demand Smooth streaming, live Smooth streaming
Timestamp	Timestamp of the request in UNIX epoch format

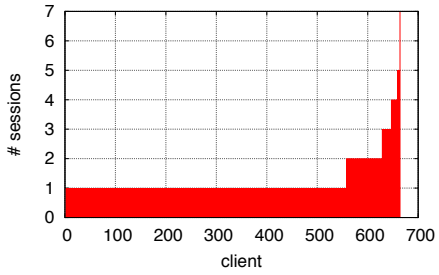
Table 1: Information about every request in the server log file

play	The start of the playout
playing	Regularly sent every 60 seconds when the playout is in progress
position	Sent when the user seeks to a different position in the video
pause	Sent when the user pauses the playout
bitrate	Sent when the player switches to a different bitrate, i.e., resutling in a different video quality
buffer	Sent when an buffer underrun occurs on the client side
stop	Sent at the end of streaming
subtitles:on	Turn on subtitles
subtitles:off	Turn off subtitles
fullscreen:on	Switch to fullscreen
fullscreen:off	Quit fullscreen mode

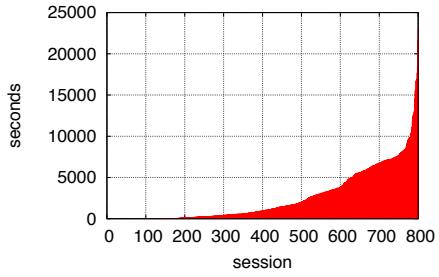
Table 2: The reported client events

User device info.	This information includes the type of browser, OS, etc.
User ID	Identification of the user
Content ID	Identification of the content
Event ID	See Table 2
Event timestamp	Time of the event in seconds that elapsed since midnight
Event data	E.g. for bitrate event the bitrate the player switched to
Session ID	The session identification
Viewer	Always SilverlightPlayer v. 1.6.1.1
Client IP	Client's IP address as seen by the analytics server
Geographic info.	This information includes country, city etc.
ISP	Client's ISP

Table 3: Information about every client event



(a) Sessions per client IP



(b) Session duration

Figure 3: Sessions statistics based on the server log

Sessions.

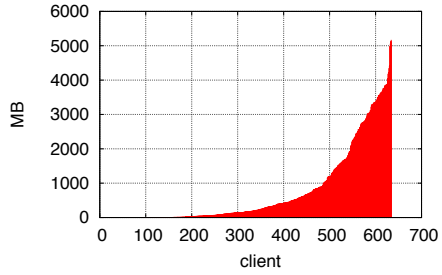
Since the IIS server is basically a "dumb" HTTP server, it does not keep state about an ongoing session. It therefore does not log the user identification, content id or the session id (as is done by the analytics server in the client log, see Table 3). For this reason, we assumed (only for the server log) that a client is uniquely identified by its IP address, i.e., we assumed there is one client per IP address[2] (this proved to be a reasonable assumption, see Section 6).

The content streamed can be extracted from the URL that is included in the server log, since the URL has a fixed format for Smooth streaming as is described in Section 3. It is therefore possible to find out the content id, bitrate and the playout time in seconds of the segment within a video stream by parsing the URL. Unfortunately, it does not include the session id. We therefore approximated the session id with the combination of the client IP address (= client id) and the content id, e.g., if a user downloaded URLs with the content id A and B we say that the user had two sessions. Figure 3(a) shows that most of the clients had only one session. This is not what we find in the client logs as we will see later. It is also interesting to measure the live session duration. To do this, we calculated the time difference between the first and last request within a session. The session duration distribution is shown in Figure 3(b). The average session duration is 42 minutes (related to the break after 45 minutes in the game). Note that 107 clients are associated with multiple sessions, that is why there are more live sessions than live clients.

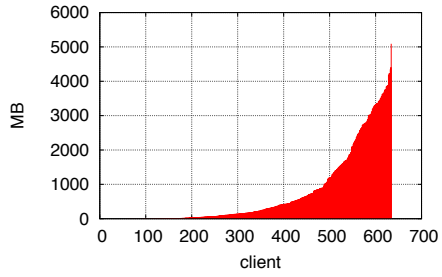
Byte transfers.

We noticed that some segments with the same playout time (included in URL, see Section 3) and within the same session were downloaded more than once, each time in a different bitrate. We conjecture this can happen because of two different reasons. First, multiple clients with different bandwidth limitations can be hiding behind the same IP (which can not be distinguished based on the available information). Second, the bitrate adaptation algorithm changes its decision and re-downloads the same segment in a different bitrate. Either way, bytes are unnecessarily downloaded, i.e., if there were multiple clients behind the same IP they could have downloaded the segment once and shared it via a cache; if the adaptation algorithm downloaded multiple bitrates of a segment it could have just downloaded the highest bitrate. We calculated the number of wasted bytes as the total number of bytes downloaded minus the bytes actually used. For this purpose, we assumed that only the highest bitrate segment for the same playout time within the same session is used by the client, i.e., if the same IP address downloads for the same playout time segments with 1 Mbit/s, 2 Mbit/s and 3 Mbit/s bitrate, only the 3 Mbit/s segment is played out and the other two segments are dis-

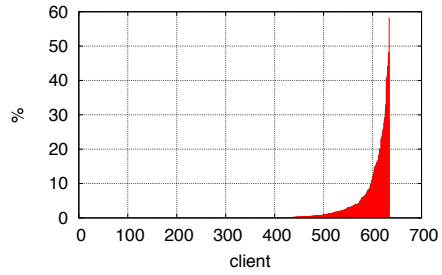
[2]We assume that, for example, there are no two clients sharing the same IP because they are behind the same NAT.



(a) Downloaded bytes

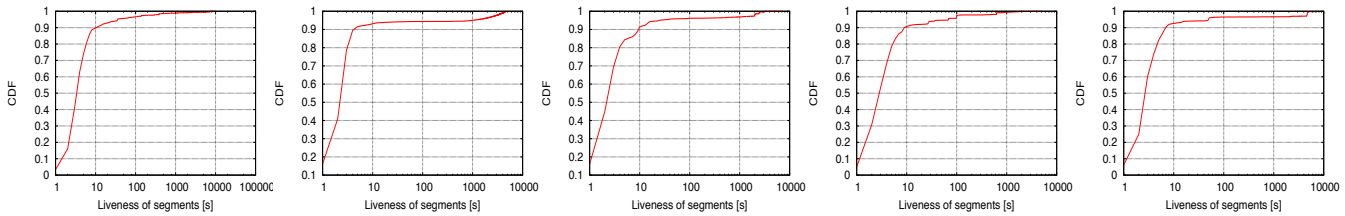


(b) Used bytes ($Downloaded - Wasted$)



(c) Wasted bytes in percent of the total bytes downloaded

Figure 4: Per client bytes statistics



(a) Game 1 (b) Game 2 (c) Game 3 (d) Game 4 (e) Game 5

Figure 5: Liveness (absolute value) of segments based on the server log

carded (wasted). Based on the server log, approximately 13 GB out of the 467 GB downloaded in total were wasted this way. The 13 GB or 3% of the whole data could have been possibly saved if a different adaptation strategy or an HTTP streaming aware cache was used.

Figure 4 shows that there are clients that waste more than 50% of their downloaded bytes. Specifically, there is one client from Greece that contributes with 15.5% (2 GB) to the total amount of wasted bytes, even though its share on the total amount of downloaded bytes is only 0.85%. We think that this person might have recorded the stream in every quality, so there is also room for improvement on detecting and preventing behaviour like this (if stated in the service agreement).

Liveness.

Each log entry, i.e., request of segment i of a particular game, includes the server timestamp when the server served that particular download request, T_i^d. It also includes the timestamp of the playout time of the video segment, T_i^p. Since we do not know the relationship between T_i^d and T_i^p (we just know that they increase with the same rate), we find the minimum of $T_i^p - T_i^d$ over all segments i for each game and assume that this is the minimal liveness (time the client is behind the live stream) for that game. In other words, the video segment with the minimal $T^p - T^d$ has liveness 0, i.e., is as live as possible. In this respect, we compute the liveness of all other segments in each game. The results are plotted in Figure 5 for the 5 most popular games. We see that about 90% of all segments in each game was less than 10 seconds behind the live stream (the liveness measured relatively to the most live segment as explained above). This means that 90% of the requests for a particular segment came within a 10 seconds period. There are also some video segments that were sent with quite some delay. One plausible explanation is that these are from people that joined the stream later and played it from the start.

6. CLIENT LOG ANALYSIS

Every player that streams a live soccer game reports to the analytics server the events described in Table 3. The type of events reported is summarized in Table 2. The client log includes all reported events for a duration of 24-hours. The general statistics based on the client log are summarized in Table 4.

Client Location.

There were 6567 unique client IP addresses in the client log. This is significantly more than the corresponding 748 live streaming client addresses found in the corresponding server log. Not surprisingly most of the IP addresses was located in Norway (Figure 6), but there were also IP addresses from almost all corners of the world (Figure 7). The international IP addresses correlated with the areas that are

Number of IP addresses	6567
Number of users	6365
Number of sessions	20401 (or 3.21 per user on average)
Number of sessions with at least one buffer underrun	9495 (or 46% of total)
Number of sessions with at least one bitrate switch	20312 (or 99.5% of total)
Number of content ids	15
Number of countries	36
Number of cities	562
Number of ISPs	194
Number of users with multiple ISPs	113 (or 2% of all users)
Number of IPs with multiple users	31

Table 4: Statistics from the client log

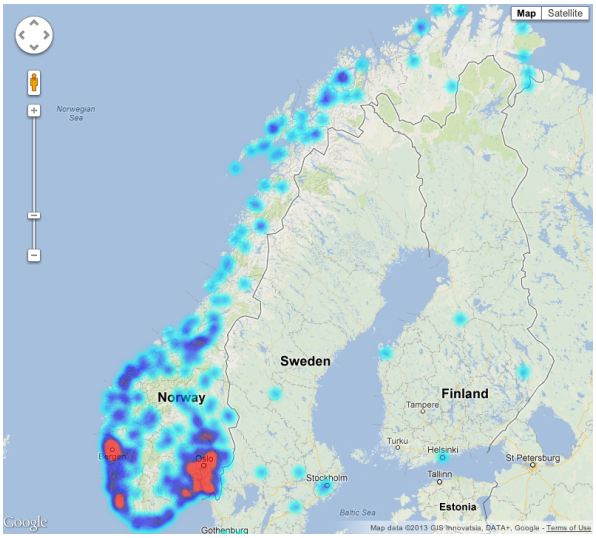


Figure 6: Geographical client distribution in Norway (the highest density of clients is in the red areas).

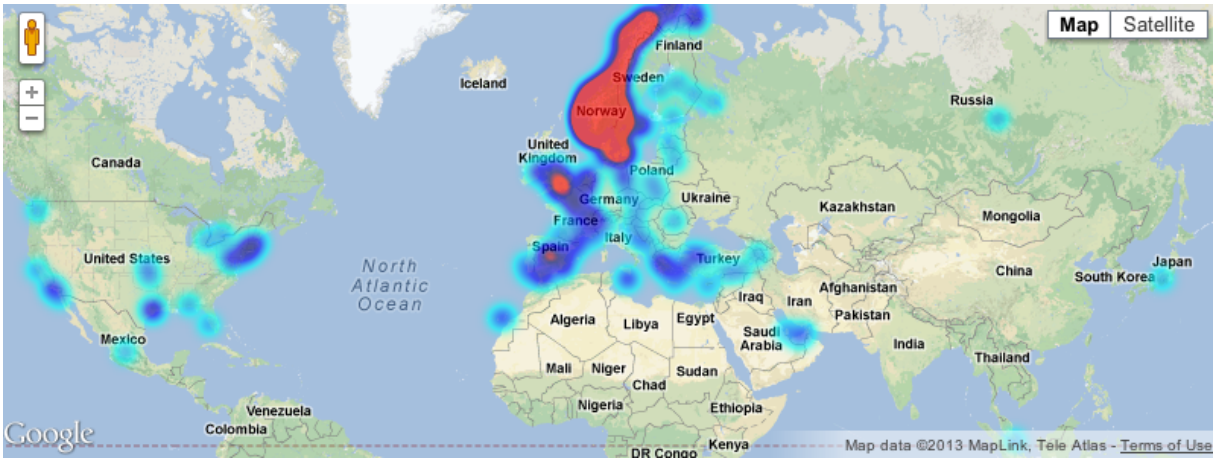


Figure 7: Geographical client distribution in the world (the highest density of clients is in the red areas).

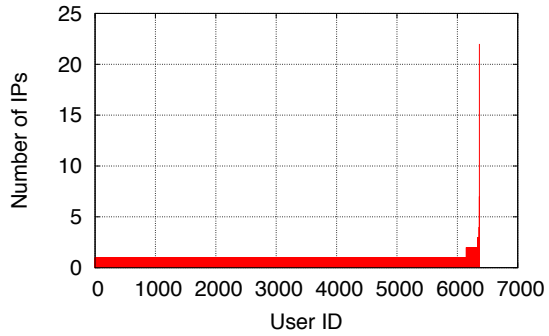
know for high Norwegian population like Spain and England. Furthermore, the assumption in the server log analysis was that an IP address matches one user/client. Figure 8 shows that this assumption is quite reasonable. Only a very small number of users used more than one IP address, and an IP address was mostly used by only one user.

Furthermore, Figure 10 shows that most of the users used only one ISP (an IP address is hosted by one ISP only). Moreover, we saw that the majority of users used either Canal Digital Kabel TV AS (2560) or Telenor Norge AS (2454) as their ISPs. The other 1442 users were served by other providers (192 different network providers in total). We could see that the quality of the streaming in terms of the number of buffer underruns (periods the streaming is paused because of slow download) depends on the ISP, see Figure 9. For example, the majority of sessions at the Oslo Airport experienced buffer underruns (the airport provides a WiFi network).

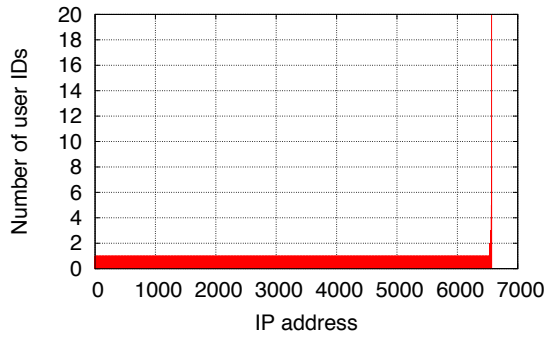
Proxy Caching.

We hypothesise that the significant difference in the number of IP addresses in the server log and the client log is due to the use of cache proxies as known from traditional HTTP networks. The proxies reply to client requests before they get to the load balancer (Figure 1) and are logged. We expect the caches to be somehow related to ISPs, i.e., we would expect a cache on the ISP level to reply to all but the first request of the same segment, e.g., the open source proxy Squid [4] only sends the first request to the origin server all consecutive requests are served from the partially downloaded data in the cache.

Our hypothesis is supported by the IP to ISP ratio. The server log IP to ISP ratio for live streaming is 0.08 whereas the client log ratio is 0.03. In other words, we find 11% of the client log IP addresses in the server log, but find over 30% of ISPs from the client log in the server log (see Table 5 for exact numbers). We therefore hypothesise that HTTP caches located between the client and the server reply to a large number of client requests. This means that, as expected, one of the large benefits of HTTP segment streaming is the reuse of existing HTTP infrastructure.



(a) Number of IP addresses per user



(b) Number of users per IP address

Figure 8: User to IP address mapping

Content and Sessions.

On the content side, we observe in Figure 11 that only about 2000 users streamed more than one content, i.e., one football game. The most popular content was game 1 with over 7500 sessions followed by 4 games with about 2000 to about 4000 sessions.

We also estimated the session durations based on the client log. This is not a straightforward task since only a small fraction of the sessions is started with a *play* event and terminated with a *stop* event (we suspect the user closes the browser window without first clicking on the stop button in

Type	Server log	Match in client log
live Smooth streaming	748 IPs	723 IPs (or 97%) hosted by 58 ISPs
on-demand Smooth streaming	74 IPs	5 IPs (or 7%) hosted by 3 ISPs
progressive streaming	581 IPs	390 IPs (or 67%) hosted by 32 ISPs

Table 5: IP to ISP statistics

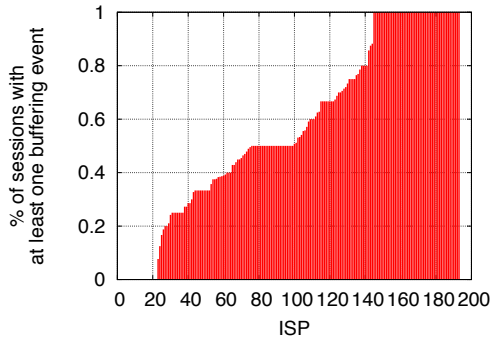
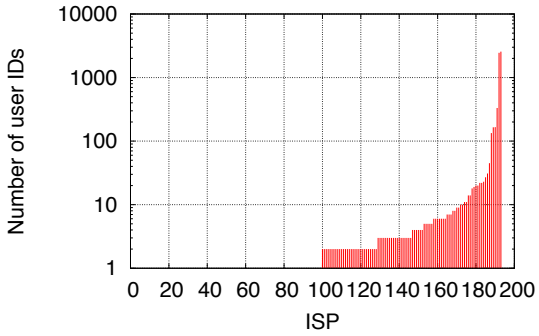
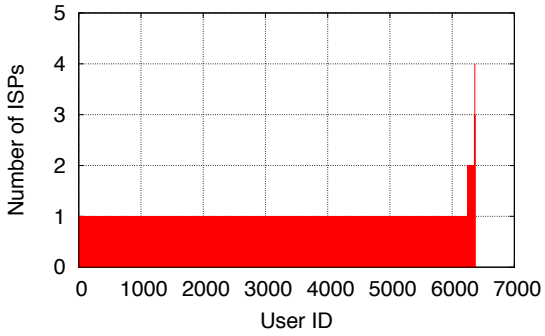


Figure 9: The percentage of sessions with at least one buffer underrun by ISP

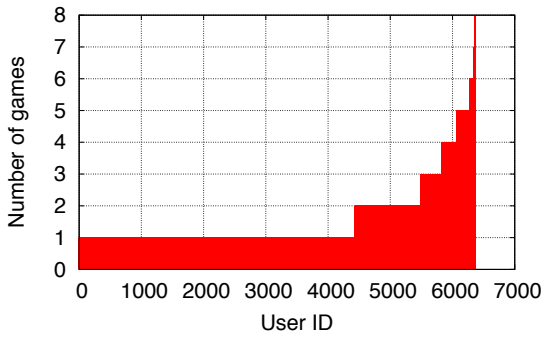


(a) Number of user IDs per ISP

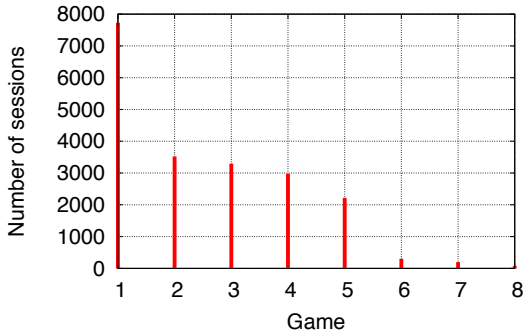


(b) Number of ISPs per user ID

Figure 10: ISP to user statistics



(a) Number of games streamed by a user



(b) Number of sessions per game (8 games were scheduled for that Sunday)

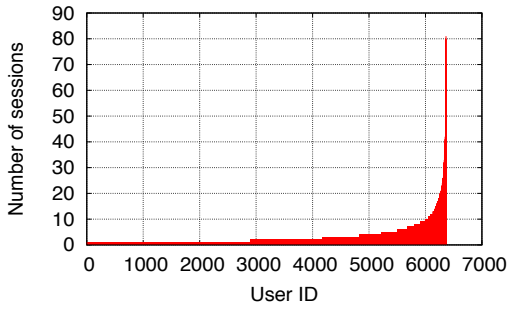
Figure 11: Content statistics

the player). We therefore calculated the session duration as the time difference between the first and last *playing* event[3]. Session durations calculated this way are, in general, shorter than they really were because they do not include the time before the first *playing* event and the time after the last *playing* event. Note also that the session duration represents the time the player was playing. Particularly, it does not represent the time the player window was open, but rather the time the player was playing a stream, e.g., the time when the stream was paused is excluded. Figure 12(b) shows that the calculated session durations ranged from 100 seconds to more than 3 hours. Please note that 6826 sessions did not contain a *playing* event and therefore are not represented in this graph (e.g., they were too short for a *playing* event to be sent).

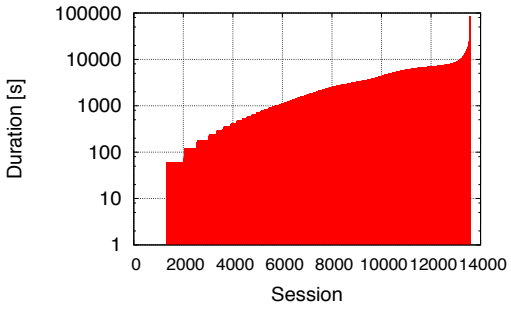
Figure 12(c) shows that in the majority of sessions the bitrate was switched at least 3 times. However, Figure 12(d) shows that even with bitrate (quality) switching, the clients were not able to prevent buffer underruns. The logs report that more than a half of the sessions experienced at least one buffer underrun.

It is clear, from Figure 13, that the pressure on a streaming system like this is the biggest when a event begins. In this case all the games started at 5 PM UTC, and that is the time most of the clients joined. Figure 14 shows the number of active sessions over time.

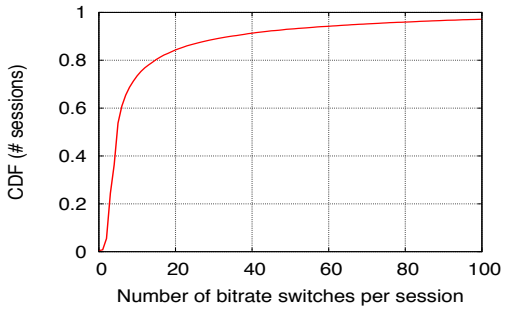
[3]The *playing* event is sent regularly when the live stream is playing.



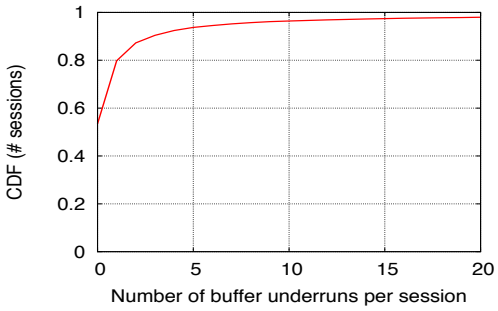
(a) Number of sessions per user



(b) Session duration based on the *playing* event



(c) Number of bitrate switches in a session



(d) Number of buffer underruns in a session

Figure 12: Session statistics based on the client log

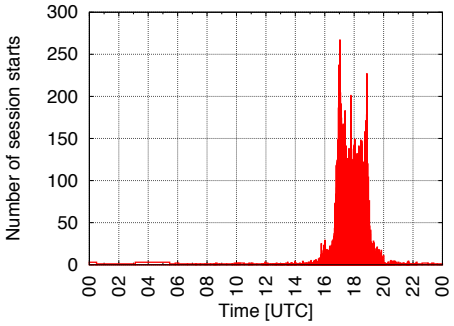


Figure 13: Number of sessions that were started per minute (all games started at 17:00 UTC)

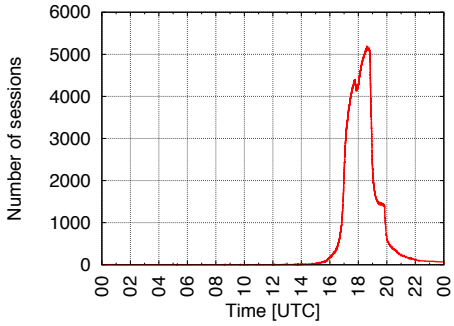
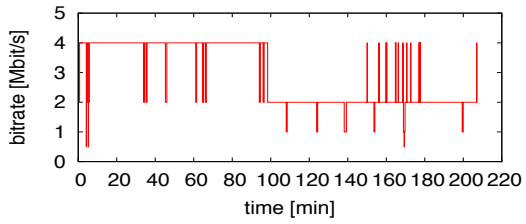


Figure 14: Number of sessions at each point in time (minute resolution)

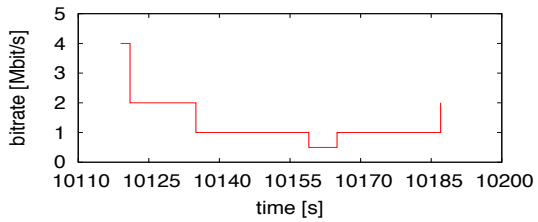
Byte transfers.

We also estimated the number of downloaded bytes based on the client log. We were very conservative and our estimations were very likely smaller than what the reality was. We split every session into periods bordered by *playing* events and/or *bitrate* events so that there was no *pause*, *stop*, *position*, *play* or *buffer* (buffer underrun) event inside a period. This ensures that the player was really playing during a period of time defined like this (session examples with bitrate switching are shown in Figure 15). A bitrate (the video content was available in 0.5, 1, 2 and 4 Mbit/s, and the audio in 96 KBit/s) was additionally assigned to each period based on the client *bitrate* event reports. The period duration multiplied by the corresponding bitrate gave us a good estimate of the bytes downloaded by the client in that period since Constant BitRate (CBR) encoding is used for the live streaming. The sum of bytes received in each period of a session gave us the total number of bytes downloaded in a session.

It is interesting that the final sum of bytes received by all sessions is many times higher than the number of bytes served by the server. The server log reports 551 GB in total whereas the estimate based on the client log is about 4.5 times higher. Even if only the smallest available bitrate is assigned to every period, the total amount of data received by the clients is 2.5 times higher than the value from the server log. This is another evidence for the presence of HTTP caches that respond to a significant number of requests.



(a) Dynamic bandwidth session.



(b) Zoomed in to minutes 168 to 170.

Figure 15: Example of bitrate adaptation throughout a session based on client reports

7. CONCLUSION AND FUTURE WORK

In this paper, we analysed the logs of adaptive HTTP segment streaming provided by Comoyo [7], a Norwegian streaming provider. We analysed two types of logs; one from the origin server and one from the analytics server to which the clients report. The analysed data included all streaming sessions ranging from very dynamic to very static sessions, see example in Figure 15.

We observed that about 90% of the requests for the same segment in live streaming is sent within a period of 3 to 10 seconds depending on the content (analysed football game). This gives a great potential for caching. This also means that a HTTP proxy cache needs to cache a segment only very shortly.

We also deduced from the segments downloaded multiple times that at least 3% of the bytes downloaded could have been saved if more HTTP caches or a different adaptation strategy had been used. Moreover, based on the number of bytes downloaded and the IP address distribution in the server and the client log, we found evidence that segments must be delivered from other sources than from the examined servers. This suggests the presence of HTTP caching proxies in the Internet that had served the requests before they got to the load balancer. This means that the origin server is offloaded and that the idea of reusing the (unmodified) HTTP infrastructure really works in real scenarios.

Furthermore, since a significant portion of the data is distributed by HTTP caches that are most likely not aware of what they are serving, we conclude that it is important to look at how HTTP streaming unaware server performance can be increased by client side or very light server side modifications. In other words, it is important to look at the performance of HTTP servers that actually deal with HTTP segment streaming traffic.

The increase of the performance of such a server is, of course, only interesting if there is some kind of bottleneck. We can eliminate the disk as being the bottleneck for live streaming as the segments are small and clients are interested only in the most recent segments of the live stream. As such, the disk I/O poses no problem with current hardware. After the disk bottleneck elimination, we are left with the problem of a network bottleneck.

There are two types of bottlenecks - a client side and a server side bottleneck. The difference is that the client side bottleneck cannot be dealt with with a faster server, i.e., the server has the capacity to serve higher bitrate segments, but the client link is not fast enough to receive them in time. The obvious solution is to throw money at the problem and upgrade the client link. In our previous work we looked at how to deal with the client bottleneck using multiple links the client might have [8].

If the bottleneck is on the server side, the server capacity needs to be upgraded to deal with the bottleneck. This can be either done by using multiple servers or by increasing the capacity of the server already deployed. We looked at how to increase the capacity of a server without upgrading its physical link, i.e., by using different client request strategy, changing the TCP congestion control etc., in [11, 12].

As ongoing work, we are investigating new functionality provided to the user in order to further use the data after the large drop at 19:00 UTC in Figure 13. One idea is to allow the users to search for events and make their own video playlists on-the-fly [10] which again could be shared in social media etc. Such interactions will change the video access patterns, and new analysis will be needed.

Acknowledgment

This work is supported by the iAD (Information Access Disruptions) center for Research-based Innovation funded by Norwegian Research Council, project number 174867.

8. REFERENCES

[1] Move Networks. http://www.movenetworks.com, 2009.
[2] Coding of audio-visual objects: ISO base media file format, ISO/IEC 14496-12:2005. *ITU ISO/IEC*, 2010.
[3] IIS Smooth streaming extension. http://www.iis.net/downloads/microsoft/smooth-streaming, 2013.
[4] Squid. www.squid-cache.org/, 2013.
[5] Adobe Systems. HTTP dynamic streaming on the Adobe Flash platform. http://www.adobe.com/-products/httpdynamicstreaming/-pdfs/httpdynamicstreaming_wp_ue.pdf, 2010.
[6] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proc. of ACM MMSys*, pages 157–168, Feb. 2011.
[7] Comoyo. http://www.comoyo.com, 2012.
[8] K. R. Evensen, T. Kupka, D. Kaspar, P. Halvorsen, and C. Griwodz. Quality-adaptive scheduling for live streaming over multiple access networks. In *Proc. of NOSSDAV*, pages 21–26, 2010.
[9] R. Houdaille and S. Gouache. Shaping HTTP adaptive streams for a better user experience. In *Proc. of MMSys*, pages 1–9, 2012.
[10] 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 *Proc. of ACM MM*, pages 989–990, 2009.

[11] T. Kupka, P. Halvorsen, and C. Griwodz. An evaluation of live adaptive HTTP segment streaming request strategies. In *Proc. of IEEE LCN*, pages 604–612, 2011.

[12] T. Kupka, P. Halvorsen, and C. Griwodz. Performance of On-Off Traffic Stemming From Live Adaptive Segment HTTP Video Streaming. In *Proc. of IEEE LCN*, pages 405–413, 2012.

[13] T. Lohmar, T. Einarsson, P. Frojdh, F. Gabin, and M. Kampmann. Dynamic adaptive HTTP streaming of live content. In *Proc. of WoWMoM*, pages 1–8, 2011.

[14] Move Networks. Internet television: Challenges and opportunities. Technical report, November 2008.

[15] C. Müller, S. Lederer, and C. Timmerer. An evaluation of dynamic adaptive streaming over http in vehicular environments. In *Proc. of MoVid*, pages 37–42, 2012.

[16] D. Nelson. Internet Information Services. http://www.iis.net/learn/media/live-smooth-streaming, 2012.

[17] R. Pantos (ed). HTTP Live Streaming. http://www.ietf.org/internet-drafts/draft-pantos-http-live-streaming-10.txt, 2013.

[18] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz. A comparison of quality scheduling in commercial adaptive http streaming solutions on a 3g network. In *Proc. of MoVid*, pages 25–30, 2012.

[19] T. Stockhammer. Dynamic adaptive streaming over HTTP - standards and design principles. In *Proc. of MMSys*, pages 133–144, 2011.

[20] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia streaming via TCP: an analytic performance study. In *Proc. of ACM MM*, pages 16:1–16:22, 2004.

[21] YouTube statistics. http://youtube.com/t/press_statistics, 2012.

[22] A. Zambelli. Smooth streaming technical overview. http://learn.iis.net/page.aspx/626/smooth-streaming-technical-overview/, 2009.