Adaptive Push-based Media Streaming in the Web

Luigi Lo Iacono and Silvia Santano Guillén
Cologne University of Applied Sciences, Cologne, Germany

Keywords: Adaptive Media Streaming, Push-based Streaming, Web, WebSocket.

Abstract: Online media consumption is the main driving force for the recent growth of the Web. As especially real-time media is becoming more and more accessible from a wide range of devices, with contrasting screen resolutions, processing resources and network connectivity, a necessary requirement is providing users with a seamless multimedia experience at the best possible quality, henceforth being able to adapt to the specific device and network conditions.
This paper introduces a novel approach for adaptive media streaming in the Web. Despite the pervasive pull-based designs based on HTTP, this paper builds upon a Web-native push-based approach by which both the communication and processing overheads are reduced significantly in comparison to the pull-based counterparts. In order to maintain these properties when enhancing the scheme by adaptation features, a server-side monitoring and control needs to be developed as a consequence. Such an adaptive push-based media streaming approach is introduced as main contribution of this work. Moreover, the obtained evaluation results provide the evidence that with an adaptive push-based media delivery, on the one hand, an equivalent quality of experience can be provided at lower costs than by adopting pull-based media streaming. On the other hand, an improved responsiveness in switching between quality levels can be obtained at no extra costs.

1 INTRODUCTION

In the early days of the Internet, video technologies have been using specific streaming protocols such as Real-Time Protocol (RTP) (Audio-Video Transport Working Group, 1996) or Real-Time Streaming Protocol (RTSP) (Schulzrinne et al., 1998). A current trend is to deliver video content using HTTP-based adaptive streaming instead, including Microsoft Smooth Streaming (MSS) (Microsoft Corporation, 2015), Apple HTTP Live Streaming (HLS) (Pantos, 2015), Adobe HTTP Dynamic Streaming (HDS) (Adobe, 2013), and MPEG Dynamic Adaptive Streaming over HTTP (DASH) (ISO/IEC Moving Picture Experts Group (MPEG), 2014). Reasons for this broad adoption of HTTP as foundation for media streaming include the reachable amount of potential consumers through the Web, the cost efficient deployment on standard HTTP servers and the exploitation of available scalability infrastructures such as Content Delivery Networks (CDN) (Held, 2010).

HTTP-based streaming protocols require that the client periodically fetches sequential file-based media chunks in order to retrieve and reconstruct the content. This is especially true for live streaming settings. This pull-based media distribution approach has strengths in respect to scalability and fault-tolerance, since the client is able to request chunks from different media servers. On the other hand, it comes with system-inherent costs due to the continuous request of media chunks. Moreover, the pull-based nature of media streaming over HTTP affects the way the adaptive change to media delivery conditions is performed. The decision what quality level to choose for the next media chunks needs to be taken on the client-side. Thus, the media player application requires measuring media delivery and rendering parameters for determining e.g. the present network conditions. The client then uses these observations for selecting a suitable quality level for the next media chunk to pull, i. e. the best quality that still ensures a seamless and smooth media playback. With regard to the adaptation a common procedure is to prepare the media content in distinct quality levels by encoding the source media at different bitrates. The client is then able to switch between these variations of the content. This is commonly denoted as stream-switching. Although having multiple versions of the same media content implies much higher storage demands on the server, an advantage is that the adaption does not require further processing during the transmission and that this does not rely on any particular properties of

Iacono, L. and Guillén, S.
Adaptive Push-based Media Streaming in the Web.
In *Proceedings of the 12th International Conference on Web Information Systems and Technologies (WEBIST 2016) - Volume 1*, pages 121-129
ISBN: 978-989-758-186-1

the employed codec, thus being codec-agnostic. Another approach is based on scalable codecs such as H264/MPEG-4 AVC (ISO/IEC Moving Picture Experts Group (MPEG), 2012). Such codecs support spatial and temporal scalability enabling the adaptation of resolution and frame rate. The difference to stream-switching is that the raw source media is encoded only once. However, the drawback is that it imposes some restrictions, such as the need for specialized servers and the limited amount of available codecs. Transcoding-based approaches adapt the video content to a specific bitrate on-the-fly by transcoding the raw content. Since this transcoding needs to be performed for each client, the processing load increases significantly providing a limited scalability.

The contribution of this paper is the introduction of a novel adaptive media streaming approach for the Web. In opposite to the currently available technologies and standards, the proposed scheme provides an adaptive push-based media delivery. By pushing media segments from the server to the client, the overheads inherent to the pull-based counterparts can be omitted completely (Lo Iacono and Santano Guillén, 2014). As a consequence, the monitoring of quality of experience parameters and according decision-making logic need to be present on the server-side. This paper extends the approach introduced in (Lo Iacono and Santano Guillén, 2014) by adaptation capabilities and evaluates various metrics in comparison to the pull-based approaches. One major aspect that has been investigated is the time required for switching between distinct quality levels. With the introduced adaptive push-based media delivery approach the usual trade-off between media segment length and data expansion is mitigated. The remainder of this paper is organized as follows: Section 2 gives a brief review on the available related work in both commercial products and algorithms proposed in the literature. Section 3 introduces the adaptive push-based media streaming approach and describes a reference implementation. Based on this functional prototype the contributed approach is evaluated in Section 4. The paper concludes in Section 5 by summarizing the obtained findings and by discussing future work.

2 RELATED WORK

Related work can roughly be classified in solutions used in commercial products and research work from scientific publications. Microsoft Smooth Streaming (MSS) (Microsoft Corporation, 2015) is an extension of the Microsoft HTTP server IIS (Internet Information Server) (Kenneth Schaefer, Jeff Cochran, Scott Forsyth, Dennis Glendenning, Benjamin Perkins, 2012) that enables HTTP-based media streaming of H.264 (ISO/IEC Moving Picture Experts Group (MPEG), 2012) video and AAC (ISO/IEC Moving Picture Experts Group (MPEG), 2004) audio to Silverlight and other clients. The video content is segmented into small chunks that are delivered over HTTP. As transport format of the chunks, MSS uses fragmented ISO MPEG-4 (ISO/IEC Moving Picture Experts Group (MPEG), 2012) files. To address the unique chunks MSS uses time codes in the requests and thus the client does not need to repeatedly download a meta file containing the file names of the chunks. Apple's HTTP Live Streaming (HLS) (Pantos, 2015) solution is currently an Internet Draft at the Internet Engineering Task Force (IETF). As MSS, HLS also enables the adaptive media streaming of H.264 video and AAC audio. The HLS client downloads a playlist containing the meta data for the available media streams, which use MPEG-2 TS (Transport Stream) (ISO/IEC Moving Picture Experts Group (MPEG), 2013) as wire format. The media content is embedded into a Web page using the HTML5 video element (Berjon et al., 2014), whose source is the m3u8 manifest file, so that both the parsing of the manifest and the download of the chunks are handled by the browser. With less success regarding the adoption in the market, HTTP Dynamic Streaming (HDS) (Adobe, 2013) is a similar solution from Adobe. Like MSS and HLS, HDS breaks up video content into small chunks and delivers them over HTTP. The client downloads a manifest file in binary format, the Flash Media Manifest (F4M), at the beginning of the session and periodically during its lifetime. As in MSS, segments are encoded as fragmented MP4 files that contain both audio and video information in one file. It differs, however, from MSS with respect to the use a single meta data file from which the MPEG file container fragments are determined and then delivered. In this respect, HDS follows the principle used in HLS instead, which requests and transmits individual chunks via a unique name.

A current trend, in which consumers move from desktop computers to smartphones, tablets, and other specialized devices to consume real-time multimedia leads to the situation that these devices present huge differences in terms of compatibility. Thus, the delivery of media to any of these platforms requires the support of a large number of streaming protocols as well as media formats and codecs. To address this issue, the MPEG standardization group has developed a general standard for the delivery of adaptive streaming media over HTTP with the aim of harmonizing the various proprietary technologies. The so called Dynamic Adaptive Streaming over HTTP (DASH) (ISO/IEC Moving Picture Experts Group (MPEG), 2014) allows standard-based clients to retrieve content from any standard-based Web server. Its principle is to provide formats that enable efficient and high-quality delivery of streaming services over the Web to provide very high user experience by providing a low start-up, no re-buffering and trick modes. To accomplish this, it proposes the reuse of existing technologies in relation to containers, codecs, DRM, etc. and the deployment on top of Content Distribution Networks (CDN). It specifies the use of either MPEG-4 or MPEG-2 TS chunks and an XML manifest file that lists the available chunks, the so-called Media Presentation Description (MPD). As an intermediate result it can be noted, that all major commercial HTTP-based adaptive media streaming solutions as well as the MPEG-DASH standard make use of stream-switching together with the pull-based delivery model.

A multitude of adaptation methods for determining network conditions and adjusting to them based on various parameters and proposing different architectures can be found in the literature. Most of the mechanisms are either client-based or require feedback from the client. (Kim et al., 2013), (Liu et al., 2011), (Douga et al., 2014) and (Jiang et al., 2012) belong to the first category and present e.g. client-based adaptation approaches based on the change between measured throughput over time, the throughput measured based on the segment fetch time, the client-side buffered video time or the harmonic mean of the bandwidth estimation over the last 20 chunks respectively. To the second category belong e.g. (Bouras and Gkamas, 2005), (Jammeh et al., 2009), presenting an architecture such that measurements are performed by the receiver, which returns a feedback message to the server. The first one describes a network monitoring module, which uses metrics such as packet loss rate and delay jitter while the second proposes a closed loop congestion controller based on fuzzy logic utilizing packet inter-arrival times (packet dispersion) and its rate of change as inputs. Another approach founded on the same principle of requiring some sort of feedback from the client is followed by (Balk et al., 2003) and (Papadimitriou and Tsaoussidis, 2007), which propose own developed protocols on top of UDP and make use of the Round Trip Times (RTT) of empty ACK messages from the client as metric to estimate the network conditions.

Less common but more related to the present work are approaches, which perform media stream adaptation on the server-side solely. Such approaches can be found e.g. in (De Cicco et al., 2011) and (Kuschnig et al., 2010). Both adaptation mechanisms are completely server-side, i.e. measuring, estimation and actuation are carried out at the server and thus the control loop does not require any explicit feedback from the client. (De Cicco et al., 2011) presents a quality adaptation controller which takes as input the queue length of the sender buffer placed at the server to select the video level. (Kuschnig et al., 2010) describes an evaluation of different rate-adaptive streaming algorithms for TCP. In this approach, the proposed server-side adaptive streaming system makes use of different metrics to estimate the network conditions, such as the time spent for the transmission of a specific media block, the current congestion window and the estimated RTT of the TCP connection.

3 ADAPTIVE PUSH-BASED MEDIA STREAMING

The architecture of the proposed adaptive push-based media streaming approach is shown in Figure 1. As for the pull-based streaming protocols, it is based on the provisioning of short media segments produced in various quality levels. The distinguishing point of the present contribution is that the stream-switching decision is conducted on the server-side and that the push-based media distribution model is deployed as foundational concept. The main reason for this decision is to introduce a media streaming approach for the Web with minimal overhead.

The proposed scheme is capable of detecting the current data throughput rate for each established client connection and delivering the best-most media quality chosen from a set of provided quality levels. The measurements as well as the adaptation process occur on the server-side, reducing the amount of control and meta data to be exchanged between client and server to a minimum. Actually this approach does not require any feedback from the client to ascertain the current available data throughput rate. Moreover, the proposed approach is based on the media push model. Thus, the server sends the available video segments to all connected clients in playback order, without having the client request each of them.

The low-overhead application-level WebSockets protocol opens the path for implementing the push-based approach for the Web (Lo Iacono and Santano Guillén, 2014). A WebSocket is a permanent bidirectional channel between a WebSocket client—most commonly a Web browser—and a WebSocket server through which the media segments can be

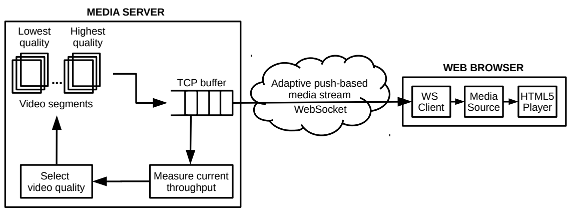

Figure 1: Adaptive push-based media delivery architecture.

pushed.

3.1 Server-side Components

As Figure 1 depicts, the media delivery process on the server is carried out in distinct phases. The process includes the in sequence retrieval of the segments, the decision on how to schedule the content feeding to the output buffer, the feedback from the own process about the current throughput and the selection of the appropriate quality level for the next segment to be sent. The strategy followed by the server in terms of chunk scheduling is immediate-sending, meaning that the next chunk is sent as soon as it is available for delivery as long as the output buffer is not full.

To the best of the authors' knowledge, and following the encapsulation principles of the OSI reference model, there are no WebSocket libraries that provide the mechanisms to provide higher-level layers with information about the underlying TCP connection that allow recognizing at application level the current status of the channel, such as the TCP buffer state. They merely provide the means necessary to handle the connection between both endpoints at a very high level by simple asynchronous read and write operations. In consequence, the server has been developed using plain TCP sockets as base and implementing an own version of the WebSocket protocol on top of it as communication means. The implementation in C# programming language is based on blocking TCP methods that the classes `TcpListener` and `TcpClient` .NET `System.Net.Sockets` namespace provide. By utilizing blocking methods, opposed to the majority of other implementations, the buffer state can be estimated measuring sending times based merely on TCP ACK feedback instead of on explicit messages from the client. This allows for calculating the current media throughput. The average of this metric over the last sent segments is a good indicator to assess how quickly the packets leave the TCP buffer and thus is used to select next segment's quality level. The selection algorithm employed relies on the essential requirement of ensuring correct playback at the client, which implies taking the measured value as threshold and selecting the highest quality media bitrate below it.

To enable a maximum performance based on the blocking TCP functions, it is mandatory to adjust the buffer size on the sender side accordingly. The buffer size might have a considerable effect on the quality of the transmission when selected inappropriately. It is enforced that this does not result too small in comparison with the average size of a segment, since this would cause the application to react with a large latency to network condition changes. On the other side, the buffer size should also not be too large as setting it larger than the segment size would cause the content to be sent through too quickly leaving the buffer very soon and having as a consequence that the metrics would not be reliable.

3.2 Client-side Components

Although this adaptive push-based media streaming approach follows in many aspects the MPEG-DASH standard, yet standard DASH players are not adequate to be used on the client side. The reason lies in the fundamentally different delivery approach. The player has been developed using the JavaScript WebSocket API (Hickson, 2012), the HTML5 Video element for video rendering and the MediaSource Extensions (Colwell et al., 2015) to feed the Video element with the segmented content.

When new content is received, it is appended to a processing queue from which they will be chained to a playing buffer. In order to enable the video player to perform a switch between streams of distinct quality, the server provides the client with the initialization segment i.e. meta data describing the media of the new representation each time a switch occurs. This meta data file is chained into the stream the same way as every other segment. As the proposed push-based media streaming approach is controlled completely by the server, there is no need for a manifest file as in other implementations, which have the purpose of informing the client of the available content.

4 EVALUATION

An experimental evaluation has been conducted in order to derive the properties of the proposed adaptive push-based media streaming approach. The main objective is to evaluate how precisely and fast the system is able to adapt to network changes as well as how efficiently this can utilize the available connection. In addition, several different segment durations have been investigated to assess what the segment duration when generating adaptive streaming content should be, on account of how the duration affects the viewing quality in various aspects. This decision often comprises a trade-off between two characteristics: the flexibility when adapting to network changes and the encoding efficiency, both desirable and unfortunately going in opposite directions.

Previous work on this topic concludes that a good recommendation for HTTP Adaptive Streaming is to use a segment length between 2 and 3 or between 5 and 8 seconds, depending on whether persistent connections are established or not. Well-known solutions like Microsoft's Smooth Streaming use 2 seconds as default segment duration, since connection details are updated every 2 seconds (Microsoft Corporation, 2015), while Apple recommends up to 10 seconds on their HTTP Live Streaming solution (Apple Inc., 2014). The expectations are, however, that when it comes to a low-overhead protocol such as the adaptive push-based approach introduced in this paper, a good compromise on the segment length can be achieved and that this falls even lower as for those over HTTP, leveraging its reduced header overhead and its push abilities to get higher flexibility on quality switching.

Having a long segment length, although favorable to obtain higher encoding efficiency, may cause playback stalls when the Internet connection is changing over time. There is another important disadvantage of long segment lengths with major effects on live streaming with regard to start-up latency, since the longer the segment length is, the longer is the time users need to wait for the first picture to be received. On the other hand, if the proposed approach were used to send short segments the switches between qualities would be faster and smoother. Thus, having removed the downside of the overhead increase HTTP protocol would produce when increasing the number of requests using shorter chunks, the initial assumption is that the proposed push-based approach would benefit of having a short segment duration.

4.1 Content Preparation

The video used for this experiment is the 10 minute open movie *Elephants Dream* (Blender Foundation, 2006). Each video-set is composed of five different quality levels, with the characteristics shown in Table 1. In total ten video-sets have been evaluated, packetized in chunks of different segment lengths varying between 0.5 and 20 seconds.

FFmpeg [1] has been used for transcoding and mp4box [2] for MPEG-DASH packetizing. A key parameter to consider in the transcoding process is the GOP (Group Of Pictures), which governs how the different types of video frames are chained. An I-frame is a picture that is coded independently of all other pictures. P- and B-frames contain difference information relative to previously decoded pictures. The GOP has an influence on many aspects including the media structure and playback. A very significant one is related to media segmentation. A decoder or player is not be able to play back a segment, which does not start with an I-frame. Thus, it is not possible to split the content into different segments at a point other than an I-frame. The duration of all segments must be equal in addition. Another relevant characteristic is the seamless quality level switching. Since the player can switch to the playback of a new representation only from an I-frame, this type of frames must be located on the same position in all of them to avoid time jumps in the playback. The GOP length is henceforth strongly related to the segment duration.

Table 1: Quality levels of video content for evaluation.

Quality Level	Resolution	Bitrate [kbps]
QL0	426x240	800
QL1	640x360	1200
QL2	854x480	1600
QL3	1280x720	2600
QL4	1920x1080	4000

With these requirements in mind, the GoP has been set to be self-contained (closed), meaning that it starts with an I-frame, it is independent from previous I-frames and it does not refer frames from other GOPs. Moreover, the GOP length as well as the key-frame interval has been set in way that just one GOP is included in every segment, according to:

Frame rate [fps] * Segment duration [s]

The framerate has been set to 24 fps in every case and the resolution is the only factor which differ from one another, starting from 426x240 in the lowest quality (QL0) to 1920x1080 in the highest one (QL4), which gives as a result a video overall bitrate of approximately 800, 1200, 1600, 2600 and 4000 kbps respectively, as listed in Table 1. Afterwards, mp4box has been used to segment the video in chunks of small duration and according to DASH AVC 264 (DASH Industry Forum, 2015) for ten different segment lengths (see Table 2).

[1]https://www.ffmpeg.org
[2]https://gpac.wp.mines-telecom.fr/mp4box/

4.2 Evaluation Results

The experimental evaluation results demonstrate how the proposed approach performs compared to pull-based counterparts based on HTTP. In addition, the experiments support the initial assumptions regarding the trade-off between the encoding efficiency and the adaptation flexibility, showing effects of the segment length. The file size on hard disk each set requires is used as metric to assess the encoding efficiency. The size is affected because of how media content is structured, using different types of frames, i.e. I-, P- and B-frames. The higher the amount of I-frames there is, the larger the size will be, and this is very much related to the segment length. For the investigated content, this comparison is represented in form of a bar graph in Figure 2 for each video set and the same quality level, in this case the lowest quality (QL0) for all. When reducing the segment length the size increases considerably, resulting in a remarkable size difference between the highest and the lowest length, with around 1.3 GB and 1.53 GB, respectively. On the other hand, the variation is negligible between a segment length of 5 and one of 20 seconds.

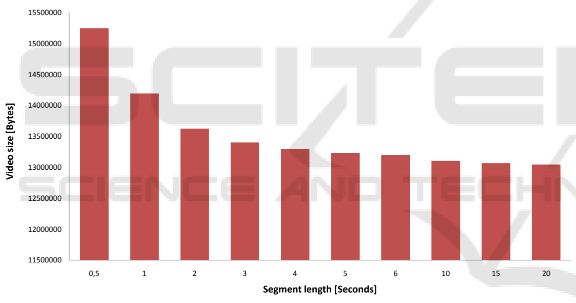

Figure 2: Total size of the video for each set of the same representation.

The application level protocol used for the transmission also introduces an overhead that adds to the media size. It also depends on the number of segments. As mentioned previously, the use of HTTP as application level transfer protocol produces a large overhead for short segment durations, much higher than that from WebSocket protocol for the same case. The table in the appendix shows an approximation of the average overhead that would be produced for each video set and for each of the quality levels in case of using HTTP, regarding the overhead generated by updating the manifest file and the intrinsic overhead of using HTTP for the media transmission in contrast to the overhead of this push-based approach. The results present very different results for each implementation, which may have a significant impact on resources needed and ultimately on the final costs for the transmission. For the worst case, i.e. using very small chunks of 0.5 seconds, the sum of HTTP overhead and the manifest overhead can be as high as 4.82 % of the total size for the lowest quality level, compared to as slow as 0,0086 %, practically negligible, for a WebSocket implementation. This difference lies in the amount of meta data that needs to be sent, including request and response headers as well as the manifest file necessary on HTTP-based adaptive streaming mechanisms, which in many implementations is updated every time before requesting a new chunk for live streaming. The size of this file varies for each stream. However, its average size is of several KB. The percentage of overhead in relation to the video size is always higher for lower qualities due to the fact that higher quality videos occupy more size in memory. Therefore, for the same segment length, 0.5 seconds, the overall HTTP and manifest overhead would add up to 1.15 % of the total size for the best quality in HD.

The architecture of the experimental setup consists merely on the evaluation client and the media server, which are connected through a local network, to ensure that the bandwidth when no specific settings apply is high, and a bandwidth shaper between them. The bandwidth shaper controls the maximum achievable bandwidth with the Linux traffic control system *tc*. The tests have been carried out creating specific queuing disciplines with *qdisc* and a hierarchical token bucket *htb* applied to the specific interface and port to limit the client download rate. The video stream is sent over this link whose maximum available throughput changes following the function depicted in Figure 3, with a minimum value of 1000 kbps and maximum value of 4000 kbps with random step variations that occur at multiples of t=50 seconds.

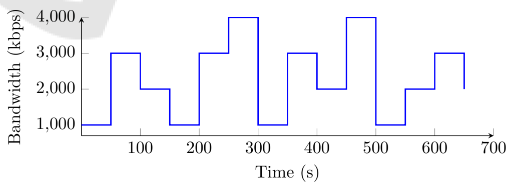

Figure 3: Data throughput shaping function.

Such scenarios with abrupt changes to the throughput are commonly used to evaluate dynamic system responses and observe how precise and how quickly the controller is able to adapt the video level to fast bandwidth variations. It has been employed in previous research as the mentioned in Section 2 to evaluate the key features of the response of adaptive streaming controller. The same bandwidth shaping function has been applied to all of the video-sets. These differ in the total media size, which explains why the total transmission of short segments video-sets takes more time than for that of longer segments. On the client side, the traffic has been captured using Wireshark[3] to extract the variations of the changing throughput. These measurements have been combined in a graph with the information the client obtains about the quality level of each of the segments received and the timestamp of the arrival. The contrast they show gives the pursued information to grasp the response of the adaptive controller. Figures 4, 5 and 6 are composed of the measured throughput in kbps for the application represented by a black line and the quality level of the segments, where each segment is marked as a red spot at the point in time it was received. Figure 4 shows how instant an implementation with short segments of 0.5 seconds each can adapt to match the available bandwidth, which is less than two seconds when the bandwidth increases and from 10 to 15 seconds when the bandwidth decreases. Figure 5 contains the results obtained by a scenario with segments of a length of 4 seconds each. It gets apparent that in comparison to the previous evaluation scenario this shows a poor adaptiveness especially when the bandwidth goes down, causing stalls in video playback as the bitrate of the quality level is well over the available bandwidth. Figure 6 shows an extreme case where the segment length is 10 seconds and the system is not able to react in time to match the available bandwidth. From a consumer viewpoint this would result in unpleasant consequences, leading to a very poor viewing experience caused by repetitive stalls in the playback.

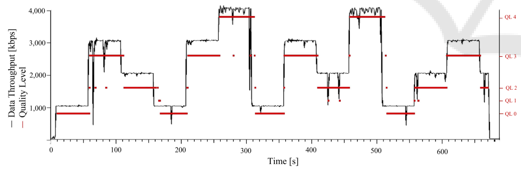

Figure 4: Stream-switching results for a segment length of 0.5 sec.

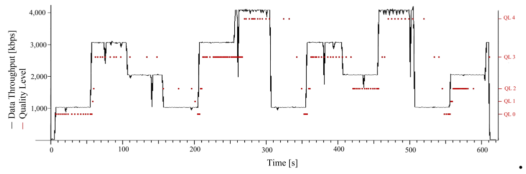

Figure 5: Stream-switching results for a segment length of 4 sec.

[3]https://www.wireshark.org/

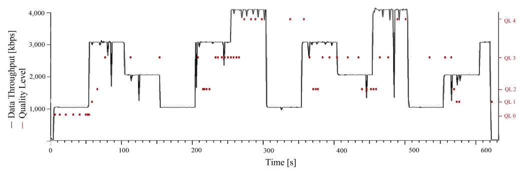

Figure 6: Stream-switching results for a segment length of 10 sec.

5 CONCLUSION

The achieved results from the conducted evaluations show very favorable improvements when implementing adaptive media streaming in the Web following a pushed-based delivery model. This contribution is based on the low-overhead WebSocket protocol leveraging its abilities for reducing communication overheads while eliminating the exchange of meta data in particular. To further reduce the overhead, all adaptation measures have been implemented on the server-side, obsoleting otherwise required synchronization communications. The performed evaluations also analyze the effect of the segment length on an adaptive implementation to ascertain which length best fits. According to the obtained results, a trade-off stemming out of the segment length. In respect to the introduced adaptive push-based media streaming approach this trade-off is lessened due to the generally low footprint of the protocol. The data expansion coming with small segment lengths can be partly compensated by the minimal protocol overhead. A compromise lies in a segment length of around 1 second, allowing for a responsive adaptation to network changes while still increase the data volume only moderately. Hence, when adopting the introduced approach, one gets an additional degree of freedom when designing adaptive Web-based media streaming services. The design can either optimize for minimal data expansion or for a maximal responsiveness to changing conditions.

REFERENCES

Adobe (2013). Adobe HTTP Dynamic Streaming Specification. http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/hds/pdfs/adobe-hds-specification.pdf.

Balk, A., Maggiorini, D., Gerla, M., and Sanadidi, M. Y. (2003). Adaptive mpeg-4 video streaming with bandwidth estimation. In *Proceedings of QoS-IP*.

Berjon, R., Faulkner, S., Leithead, T., Navara, E. D., O'Connor, E., Pfeiffer, S., and Hickson, I. (2014). HTML5 Video Element. Working draft, W3C. http://www.w3.org/TR/2011/WD-html5-20110113/video.html#video.

Bouras, C. and Gkamas, A. (2005). Performance of adaptive multimedia transmission: The case of unicast technique. In *INC 2005, Proceedings of the fifth international Network Conference 2005*.

Colwell, A., Bateman, A., and Watson, M. (2015). Media source extensions. Editor's draft, W3C. dvcs.w3.org/hg/html-media/raw-file/tip/media-source/media-source.html.

De Cicco, L., Mascolo, S., and Palmisano, V. (2011). Feedback control for adaptive live video streaming. In *MMSys'11 Proceedings of the first annual ACM SIGMM conference on Multimedia systems*.

Douga, Y., Bourenanea, M., and Melloukb, A. (2014). Adaptive video streaming using tcp factors control with user parameters. In *Procedia Computer Science, Volume 34*, pages 526–531.

Held, G. (2010). *A Practical Guide to Content Delivery Networks, ISBN: 1439835888, 9781439835883*. CRC Press, Inc., Boca Raton, FL, USA, 2nd edition.

Hickson, I. (2012). The web sockets api. Candidate recommendation, W3C. www.w3.org/TR/websockets.

ISO/IEC Moving Picture Experts Group (MPEG) (2004). Advanced Audio coding. International standard iso/iec 13818-7, ISO/IEC.

Jammeh, E. A., Fleury, M., and Ghanbari, M. (2009). Rate-adaptive video streaming through packet dispersion feedback. In *IET Communications, Volume 3, Issue 1, Print ISSN 1751-8628, Online ISSN 1751-8636*, pages 25 – 37.

Jiang, J., Sekar, V., and Zhang, H. (2012). Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *CoNEXT'12, International Conference on emerging Networking EXperiments and Technologies*, pages 326–340.

Kim, Y., Shin, J., and Park, J. (2013). Design and implementation of a network-adaptive mechanism for http video streaming. In *ETRI Journal, Volume 35, Number 1*.

Kuschnig, R., Kofler, I., and Hellwagner, H. (2010). An evaluation of tcp-based rate-control algorithms for adaptive internet streaming of h.264/svc. In *MMSys'10, Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 157–168.

Liu, C., Bouazizi, I., and Gabbouj, M. (2011). Rate adaptation for adaptive http streaming. In *MMSys'11, Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 169–174.

Pantos, R. (2015). HTTP Live Streaming. Internet-Draft draft-pantos-http-live-streaming-18, Internet Engineering Task Force.

Papadimitriou, P. and Tsaoussidis, V. (2007). A rate control scheme for adaptive video streaming over the internet. In *ICC'07, IEEE International Conference on Communications*, pages 616 – 621.

Audio-Video Transport Working Group (1996). Rtp: A transport protocol for real-time applications. RFC 1889, IETF.

Blender Foundation (2006). Elephants dream. http://orange.blender.org/.

DASH Industry Forum (2015). guidelines for implementation: Dash-if interoperability points. http://dashif.org/wp-content/uploads/2015/10/DASH-IF-IOP-v3.1.pdf.

ISO/IEC Moving Picture Experts Group (MPEG) (2012). ISO MPEG-4. International standard iso/iec 14496-10 mpeg-4, ISO/IEC.

ISO/IEC Moving Picture Experts Group (MPEG) (2013). ISO MPEG-TS. International standard iso/iec 13818-1, ISO/IEC.

ISO/IEC Moving Picture Experts Group (MPEG) (2014). Dynamic Adaptive Streaming over HTTP. International standard iso/iec 23009-1:2014, ISO/IEC.

Lo Iacono, L. and Santano Guillén, S. (2014). Efficient and adaptive web-native live video streaming. In *International Journal on Advances in Internet Technology vol. 7, no. 3 - 4, ISSN: 1942-2652*, pages 232–242.

Apple Inc. (2014). HTTP Live Streaming Overview. https://developer.apple.com/library/ios/documentation /NetworkingInternet/Conceptual/StreamingMediaGuide/StreamingMediaGuide.pdf#page=31.

Kenneth Schaefer, Jeff Cochran, Scott Forsyth, Dennis Glendenning, Benjamin Perkins (2012). *Professional Microsoft IIS 8*. Wrox.

Microsoft Corporation (2015). Smooth Streaming Specification. http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-SSTR].pdf.

Schulzrinne, H., Rao, A., and Lanphier, R. (1998). Real time streaming protocol (rtsp). RFC 2326, IETF. www.tools.ietf.org/html/rfc2326.

APPENDIX: MEASUREMENT RESULTS

Segment length [s]	Quality level	Adaptive pull-based streaming			Adaptive push-based streaming		
		Media data [%]	HTTP Ov. [%]	Manifest Ov. [%]	Media data [%]	WS Ov. [%]	Manifest Ov. [%]
0.5	q0	95.18	0.65	4.17	99.9914	0.0086	0.0
	q1	96.67	0.45	2.89	99.9940	0.0060	0.0
	q2	97.53	0.33	2.14	99.9956	0.0044	0.0
	q3	98.48	0.20	1.31	99.9973	0.0027	0.0
	q4	98.84	0.15	1.00	99.9979	0.0021	0.0
1	q0	97.41	0.35	2.25	99.9954	0.0046	0.0
	q1	98.17	0.25	1.59	99.9967	0.0033	0.0
	q2	98.62	0.18	1.19	99.9975	0.0025	0.0
	q3	99.15	0.11	0.74	99.9985	0.0015	0.0
	q4	99.39	0.08	0.53	99.9989	0.0011	0.0
2	q0	98.64	0.18	1.18	99.9976	0.0024	0.0
	q1	99.03	0.13	0.84	99.9983	0.0017	0.0
	q2	99.26	0.10	0.64	99.9987	0.0013	0.0
	q3	99.54	0.06	0.40	99.9992	0.0008	0.0
	q4	99.68	0.04	0.27	99.9994	0.0006	0.0
3	q0	99.07	0.12	0.80	99.9983	0.0017	0.0
	q1	99.33	0.09	0.58	99.9988	0.0012	0.0
	q2	99.49	0.07	0.44	99.9991	0.0009	0.0
	q3	99.68	0.04	0.28	99.9994	0.0006	0.0
	q4	99.79	0.03	0.19	99.9996	0.0004	0.0
4	q0	99.29	0.09	0.61	99.9987	0.0013	0.0
	q1	99.49	0.07	0.44	99.9991	0.0009	0.0
	q2	99.61	0.05	0.34	99.9993	0.0007	0.0
	q3	99.76	0.03	0.21	99.9996	0.0004	0.0
	q4	99.84	0.02	0.14	99.9997	0.0003	0.0
5	q0	99.43	0.08	0.49	99.9990	0.0010	0.0
	q1	99.59	0.06	0.36	99.9993	0.0007	0.0
	q2	99.68	0.04	0.27	99.9994	0.0006	0.0
	q3	99.80	0.03	0.17	99.9996	0.0004	0.0
	q4	99.87	0.02	0.11	99.9998	0.0002	0.0
6	q0	99.52	0.06	0.41	99.9991	0.0009	0.0
	q1	99.65	0.05	0.30	99.9994	0.0006	0.0
	q2	99.73	0.04	0.23	99.9995	0.0005	0.0
	q3	99.83	0.02	0.14	99.9997	0.0003	0.0
	q4	99.89	0.01	0.10	99.9998	0.0002	0.0
10	q0	99.70	0.04	0.26	99.9995	0.0005	0.0
	q1	99.78	0.03	0.19	99.9996	0.0004	0.0
	q2	99.83	0.02	0.14	99.9997	0.0003	0.0
	q3	99.90	0.01	0.09	99.9998	0.0002	0.0
	q4	99.93	0.01	0.06	99.9999	0.0001	0.0
15	q0	99.79	0.03	0.18	99.9996	0.0004	0.0
	q1	99.85	0.02	0.13	99.9997	0.0003	0.0
	q2	99.89	0.02	0.10	99.9998	0.0002	0.0
	q3	99.93	0.01	0.06	99.9999	0.0001	0.0
	q4	99.95	0.01	0.04	99.9999	0.0001	0.0
20	q0	99.84	0.02	0.14	99.9997	0.0003	0.0
	q1	99.88	0.02	0.10	99.9998	0.0002	0.0
	q2	99.91	0.01	0.08	99.9998	0.0002	0.0
	q3	99.94	0.01	0.05	99.9999	0.0001	0.0
	q4	99.96	0.00	0.04	99.9999	0.0001	0.0

Table 2: Adaptive media streaming protocol overheads: Comparing MPEG-DASH with the proposed adaptive push-based media streaming approach