Low Delay MPEG DASH Streaming over the WebRTC Data Channel

Shuai Zhao, Zhu Li, Deep Medhi

[1]Computer Science & Electrical Engineering Department,
University of Missouri–Kansas City, USA
{szb53, lizhu, dmedhi}@mail.umkc.edu

***Abstract*—Dynamic Adaptive Streaming over HTTP (MPEG-DASH) is becoming the de-facto data format and streaming solution for over-the-top (OTT) video on-demand streaming. The underlying HTTP transport has its limitations and known penalties in slow start, under-utilization, and other inefficiency. An alternate approach to DASH is the new transport schemes like HTTP/2.0 and WebSocket. In this work, we explore WebRTC as a streaming transport to carry DASH data and demonstrate that it is competitive in serving low delay streaming applications with fast channel switching, with proper video data coding and streaming signaling/control solutions.**

***Index Terms*—DASH, WebRTC, Low Delay, Video Streaming**

I. Introduction

Along with the increasing network traffic for video on-demand streaming services, the demand for an efficient video transfer standard has been raised over the years. Traditional video streaming protocols such as Real-Time Streaming Protocol (RTSP) and HTTP progressive downloading are not bitrate adaptive and network bandwidth is not fully unitized under erroneous user request.

MPEG-DASH or DASH [1] is one of the most popular MPEG standards and is designed to improve media transfer efficiency over the HTTP. In DASH, media such as video and audio is stored in various bitrate segments. A media presentation description (MPD) file is used to represent the media meta information. Inside of an MPD file, each media segment is associated with an HTTP-URL. DASH clients use the MPD file to fetch proper media segments based on the current network condition. It always tries to provide the best user experience with the best bitrate it can fetch from the media server. While DASH improves the media streaming quality in various network environments, it bears the disadvantage from the underneath transport protocol of HTTP. Since each DASH client segment request is an HTTP-GET request. Each communication between a DASH client and media server has an HTTP meta data overload. It does not perform well in certain media streaming applications. The current implementation of DASH is over HTTP/1.1. Each DASH client can only request one HTTP-URL at a time and does not support bi-directional communication.

The current work-in-progress protocol of DASH over HTTP/2.0 is still under experimentation. It defines two core experiments (CE): the DASH over Full Duplex HTTP-compatible Protocols (FDH) [2] and Server And Network Assisted DASH [3] (SAND). In this definition, server *push* is enabled instead of just pulling from the client compared with DASH over HTTP/1.1. The DASH client and media server can communicate through a WebSocket server. MPD files are delivered over a WebSocket sub-protocol. The *push* signals' messages are carried in the WebSocket channel. While the new protocol can theoretically improve the DASH segment transfer rate, it uses TCP for media transfers. The well-known TCP issues, such as slow-start and window collapse, are still limit the DASH media transfer rate.

A popular browser-based web real-time communication protocol (WebRTC) has provided a set of API [4] that is defined by the World Wide Web Consortium (W3C). It provides a peer-to-peer communication channel between two browsers. The WebRTC protocol is defined by Internet Engineering Task Force (IETF) [5]. It supports real time media including video and audio media as well as data transfer for binary data using its *DataChannel* [6]. The advantage of WebRTC data transfer protocol is that it can use either TCP or UDP as a transport layer protocol depending on your current network setup and application character. It only uses UDP when web applications use WebRTC for data transfer.

In this work, we explore the low delay WebRTC data channel as a transport vehicle for carrying DASH video sessions, and deploy our own sender side pacing solution for low delay DASH streaming. Simulation over a relay network using the NS-3 platform, demonstrates the significant improvement in end-to-end QoE delivery and reduction of start up and channel switching delays.

The paper is organized in to the following sections, in section II discuss the related work. In Section III, we discuss the current issues and on-going work with the MPEG-DASH group addressing low delay streaming. In Section IV, we introduce the WebRTC transport infrastructure and our DASH over WebRTC system setup. In Section V, the simulation setup and results are discussed. Finally, we discuss the performance and outline future work.

II. Related Work

DASH is becoming the de-facto data format and streaming solution for the video on-demand streaming applications. Video streaming traffic over a web browser is continuously growing. High-quality user experience is an increasing trend. This paper explores a broad range of recent studies including DASH implementation, improvement, and WebRTC use cases. The standard contributions for MPEG-DASH can be found in [7]. The standard video segmentation in DASH has been explored in [8] and [9]

The number of DASH implementation use cases [10] [11] [12] have been shown the benefit of DASH as a video services method. The exiting DASH improvement recommendation has been studied with the standard DASH definitions. Suggestions for improvement of DASH user experiences have been raised by various media playback rate adaption algorithms [13] [14]. A dynamic HTTP streaming adaption algorithm is proposed in [15]. In our work, we proposes a new DASH segment switchover using a sub-representation segmentation description.

Our approach of utilizing WebRTC as a transport layer protocol is focusing on its *DataChannel*. A use case such as WebRTC P2P application is studied in [16]. Performance of WebRTC has been explored by many studies in web browsers [17] [18] as well as for mobile users [18]. In this work,we propose to use WebRTC to transfer DASH segment over the network and explore the benefits of a WebRTC *datachannel* compared with a traditional DASH segment transfer.

Further more, to improve the granularity of QoE/QoS operating points, spatio-temporal quality layer signaling has been proposed in [19] for the MPEG Multimedia Transport (MMT) solution, which is the next generation media transport replacing MPEG TS for broadcasting and IP network delivery. In this work, new fine granular quality layering and signaling are introduced, capitalizing on the new spatio-temporal quality metrics supported by the ISOBMFF quality metric work [20]. More fine granular operating points improve the robustness of the streaming, and offer more choices at the time of congestion. They also allow for a lower delay streaming start and channel switching.

The studies mentioned above mainly discuss the DASH and WebRTC performance problems by various usage cases. There are other studies that focus on WebRTC congest control algorithm design [17] [21] [22] [23] that aggressively explore new measurements and new congestion models to improve the utilization and delay performance. The streaming applications like DASH on top of the WebRTC data channel, can take advantage of this new quality. They also introduce APIs to expose more congested state information to pace the quality layer scheduling and transmission of the data segments.

III. Low Delay DASH Streaming over HTTP

By offering multiple representations of the same media content, DASH clients can independently choose a proper media segment based on current network throughput observed by itself and mostly from the measurement of content round trip time (RTT) between DASH clients and the content server. In a wide network area environment, media content servers are most likely to be hosted at content delivery networks (CDN). DASH content is replicated on the CDN edge servers to minimize the RTT to the clients they are serving. Sophisticated HTTP Caching and CDN traffic optimization and localization are employed to improve overall QoS. However, due to client pulled operation and inherent slow start of TCP, the DASH streaming solution (in general) requires a much longer initial delay than the traditional broadcasting and linear IPTV experience.

The standard DASH allows clients to pull media content by requesting pre-defined media meta information files called Media Description Presentation (MPD) files. Any communication between DASH clients and servers is HTTP Assisted; meaning it uses the HTTP protocol for data transfer control. DASH over HTTP/1.1 supports client side pulling. This client-driven approach allows clients to make decisions for the next segment of downloading. It adapts to continual change of network throughput, which in turn provides a better user experience. However, it brings new challenges, such as: (1) how to make sure of the media server's reliability; (2) how to improve content delivery efficiency; (3) how to control advertisement placement during video playback, etc.

In order to address these issues, two core experiments on standard DASH have been introduced (a) SAND–Server and Network Assisted DASH and (b) DASH over Full Duplex HTTP-compatible protocols (FDH). Both SAND and FDH are still HTTP-based; they introduce server side push methods to assist standard DASH.

SAND introduces network middle boxes that allow DASH clients to send feedback and coordination messages to accelerate the DASH sessions. It defines DASH Aware Network Elements (DANE) as network entities capable of sending and/or receiving SAND messages. During the SAND Core Experiment, actual experiments were conducted to bring evidence of the usefulness of network assistance. It enables fast convergence after a DASH session starts. The fast network bandwidth estimation is around 15 times faster compared with the traditional DASH based on [24].

FDH exploits the new HTTP/2.0 transport protocol that supports bi-directional traffic by minimizing the number of DASH clients requests. It is investigating the problem of delivering media segments with shorter delay and/or reducing the request processing at the HTTP server. The technologies considered are HTTP/2 and WebSockets. The main idea is that once a client is interested in streaming a particular content (e.g., a live channel), it may be beneficial not to send an individual GET request for each segment to the network. The network may keep sending segments once they become available to the client for a predetermined amount of time or until the client tells the server to stop. This is known as the K-Push operation mode in DASH FDH. It minimizes the number of client content GET messaging overhead, and prevents the TCP window collapse resulting in a slow start recovery.

The current proposed SAND message carrier implementa-

tionand FDH is still using HTTP that still bares the HTTP payload but does not perform well in time-sensitive streaming situations and other existing drawbacks such as no support of bidirectional communication for SAND between the client and server.

IV. LOW DELAY DASH STREAMING OVER WEBRTC

In our work, the *datachannel* function of WebRTC is utilized as a media pipeline for DASH traffic. The WebRTC $datachannel$ can be used as a peer-to-peer data transfer channel. The way of finding a peer is through a signaling process. Signaling can be done by a signal gateway like WebSocket, Socket.io and XMPP to exchange peer network information such as session initiation protocol (SIP), session control messages (SDP), network configurations as ICE candidates, and other control messages. The WebRTC protocol itself can also solve issues caused by NAT or firewalls using various signaling server designs such as WebSocket.

The goal of WebRTC is to simplify the development of real time data transfer and communication over networks by providing standard web APIs, which are defined by IETF and W3C. The main target of WebRTC is for peer-to-peer multimedia streaming, including video-on-demand (VOD) and living videos. It supports client-server data transfer paradigms as well as a multiplexing peer-to-peer data channels. The web server such as WebSocket/socket.io acts as a rendezvous server for signaling purpose to build a peer-to-peer communication channel. Because the WebRTC architecture does not mandate any particular protocol/server types between peers, it leaves much large freedom when deploying WebRTC over a WAN network.

Our approach of utilizing WebRTC as a transport protocol for DASH segment transfers focuses on WebRTC's *datachannel* function. It is created between peers using the RTCPeerConnection, a starter function to build a communication channel with a particular peer and using an underlying transport built as the Stream Control Transmission Protocol (SCTP) over Datagram Transport Layer Security (DTLS) over UDP.

Congestion control over WebRTC is still not finialized. The WebRTC *datachannel* provides the best effort real time data transfer services. Any congestion control algorithm implemnetation needs to work well with SCTP to prevent SCTP starving [25]. It also needs to make sure to keep a low delay, packet loss, and reasonable fairness between flows. Google GCC [23] is the only built-in congestion control algorithm for WebRTC. The basic concept of the GCC is to have two controllers located at sender (A_s) and client side (A_r). The A_s probes the available bandwidth and A_r computes the "Client Estimated Maximum Bitrate" in order to limit the sending rate from A_s. The GCC improves WebRTC real time data transfer rate by fine tuning the server and client's sending and receiving rates. The server will push data to the client based on the computed rate.

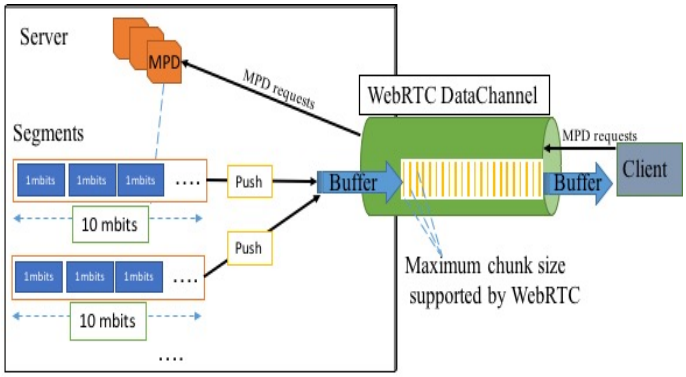

Fig. 1. DASH Over WebRTC

Traditional DASH is a pull based video streaming solution over the HTTP, which suffers from inefficiency and under-utilization of the underlying TCP transport. Periodic pull based streaming further exacerbates the TCP slow start problem. In this work, we investigate the WebRTC real time data transfer mechanism and propose a sender pacing technique to further improve DASH segment transfer rate and enable low delay DASH streaming.

With WebRTC based solution, which is RTP based, we are not suffering from TCP under-utilization. However this comes with a cost because the sender needs to take over the congestion avoidance logic and pacing the transmission. We use a sender self-timing transmission scheme, with underlying webRTC congestion signals exposed via WebRTC APIs to help avoid congestion. Fig. 1 shows our system architecture. A server serves DASH MPD files and segment data. A DASH client communicates with the server through a WebRTC *datachannel*. How WebSocket creates a WebRTC *datachannel* among all peers will be explained in Section V-B.

WebRTC has a concept called the maximum chunk size that represents the maximum data chunk it supports for each data transfer. When a client requests an MPD file, the server locates the correct media content and pushes each DASH segment to the client using the WebRTC *datachannel*. For example, when a client requests an MPD file that describes a segment file of 10 seconds. Based on the supported maximum chunk size (ex. 1 KB), then our sender pacing method will send segments by 100 frames every 10 seconds. The server's sending rate (r) is controlled by both the sender rate controller A_s and receiver rate controller A_r to achieve an optimal DASH segment transfer rate. User experience can then be further improved.

Section V shows our experiments and comparisons between standard DASH and our proposed WebRTC as a transport layer for standard DASH.

V. EXPERIMENTATION AND PRELIMINARY RESULT

In this work, we conduct our experiments on both DASH and WebRTC using an NS-3 simulated network. In order to run the simulated network with anexisting DASH.js player and WebRTC applications, we deploy two Linux containers using

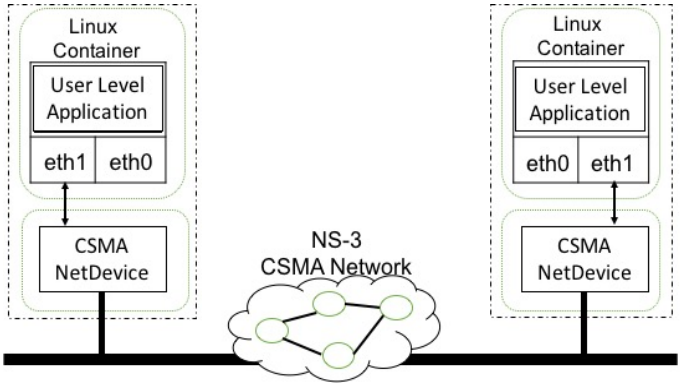

Fig. 2. Experiment Setup

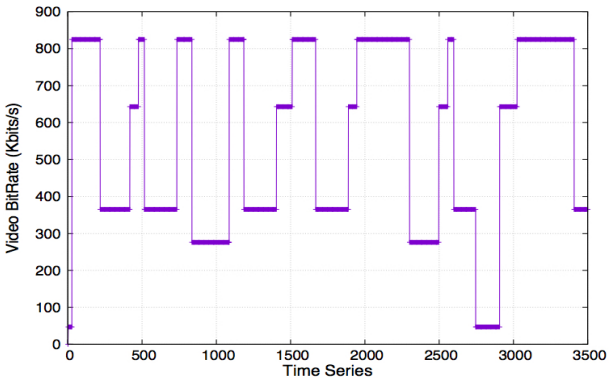

Fig. 4. DASH Client Play Bitrate

a Linux LXC library and Chrome as our test web browser. The DASH.js player and WebRTC are deployed as user level applications running on top of two different Linux containers that are connected by tap devices using an NS-3 simulated CSMA network. Figure 2 shows the topology setup using NS-3. The NS-3 network link bandwidth is set to 1 Mbps. The bandwidth of the simulated network is tested by the iperf network tool. The average UDP link bandwidth was found to be around 1 Mbps and the average TCP link bandwidth was around 0.8 Mbps.

The benefit of using LXC in this work is two-fold: (1) it can take full advantage of virtualized network topology with various network conditions; (2) instead of rewriting any existing applications, we can deploy existing ones. It also is much more flexible compared with NS-3's Direct Code Execution (DCE) technique. Each LXC node has two network interfaces. They are used for data transfers and WebRTC signal control respectively. This allows us to use either a local signaling server as well as a remote one such as Firebase.

A. DASH Client Playback

DASH streaming bitrate is driven by the client side. Based on the available network bandwidth, the client requests the next proper bitrate segment. Once the media server receives the requests, it then sends the requested segment to the client. The advantage of this approach is that the client has the full control of the next segment needed. This client based, media segment, pulling approach can avoid extended playback delay and decrease jitter time in order to improve user experience by downloading the adaptive bitrate segment. The media server itself has no knowledge of network conditions; its main function is to provide the various bitrate segments. In our work, we propose to use WebRTC as a transport layer protocol for segment transfers. In order to show the advantage of our approach, we set up a DASH environment to show how the media playback bitrate changes and DASH segment pulling performance at the DASH client side. We deployed a DASH.js v2.1 player on one LXC container and a DASH segments' Apache Server on the other end of the network topology (see Fig. 3).

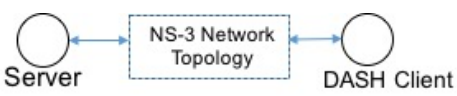

Fig. 3. DASH Topology

The simulated NS-3 network between the media server and DASH client is a Carrier Sense Multiple Access (CSMA) network with 1 ms delay between each simulated node. There is no background traffic running when we conduct our tests.

Fig 4 shows the video playback bitrate changing for the DASH client after a couple times of warmup running. The maximum playback rate is around 0.8 Mbps. The playback bitrate keeps changing and shows an ON/OFF pattern. The initial requests start from a low bitrate segment and then go up to the maximum bitrate segments. After a relative long period of constant high bitrate segments requests and download,s the network bandwidth is depleted and at the same time, the DASH client detects that the network is now low in bandwidth and decides to request a low bitrate segment in order to keep receiving media segments from the server without causing any playback delay and buffer.

A typical DASH client like DASH.js uses HTTP as the data transport layer protocol that runs on top of the TCP that shows a slow start pattern. Once the TCP window collapses, we can see the client side playback rate drop dramatically. That is a very obvious drawback of traditional DASH streaming using

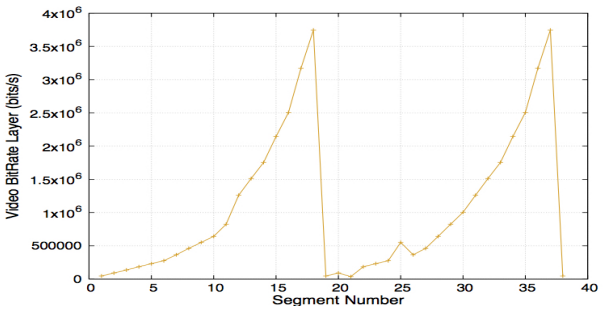

Fig. 5. DASH Segment SwitchOver Sequence

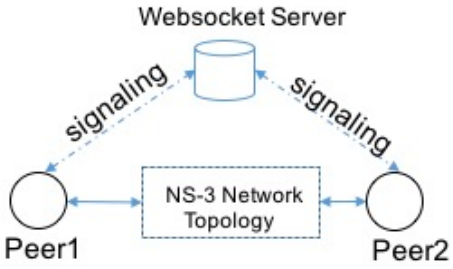

Fig. 6. WebRTC Topology

TCP protocol. In Fig. 4, one of the actual data throughputs pulled by the DASH.js client is illustrated. It shows the fluctuation of the channel utilization due to the client pull based control.

In this work, we measured the downloaded segment changing pattern over a simulated network. Fig. 5 shows the DASH segment changing pattern. We can see bitrate changes at the client side by downloading different bitrate segments.

While we see how the DASH client can adaptively keep changing the segment bitrate and keep a high user experience, its drawback is also very clear, because the nature of the extra HTTP payload and TCP window collapse.

B. WebRTC Datachannel As a Transport Layer Protocol

Our approach of using WebRTC as a transport layer protocol for a DASH segment transfer can mitigate the issues of the current DASH implementation. We use the *datachannel* function of WebRTC. It uses the UDP protocol for any binary data transfers.

In this work, we implemented a WebSocket server for signaling purposes between two WebRTC *datachannel* applications that are deployed based on Fig. 6. Two WebRTC peers are connected by the simulated NS-3 network. The WebRTC running data transfer application using a Chrome browser.

Fig. 7 shows the workflow for the signaling process between peers and the WebSocket server. The first peer (Peer-1) initiates the *datachannel* creation process and sends the requests to the WebSocket server. The server approves them upon request and sends the created datachannel back to the initiator and then waits for any connection from the other peers. Peer-2 sends the join request to the previously created datachannel by peer-1. Upon receiving the joining requests, the server broadcasts that message to all the other joined peers. In our experiment, the other peer is peer-1. Peer-1 then starts to send SDP and ICE information to the WebSocket server and tries to make an agreement with the joining peer. Once both peers agree with each other's connection information, a bi-directional WebRTC datachannel is created.

When using WebRTC *datachannel* transfer media binary data, we implemented a push based, self-pacing transport control, which is different from the DASH.js client's pull based control logic. We used the default 16KB chunk size and the "bufferedamountlow" event for sender side flow control.

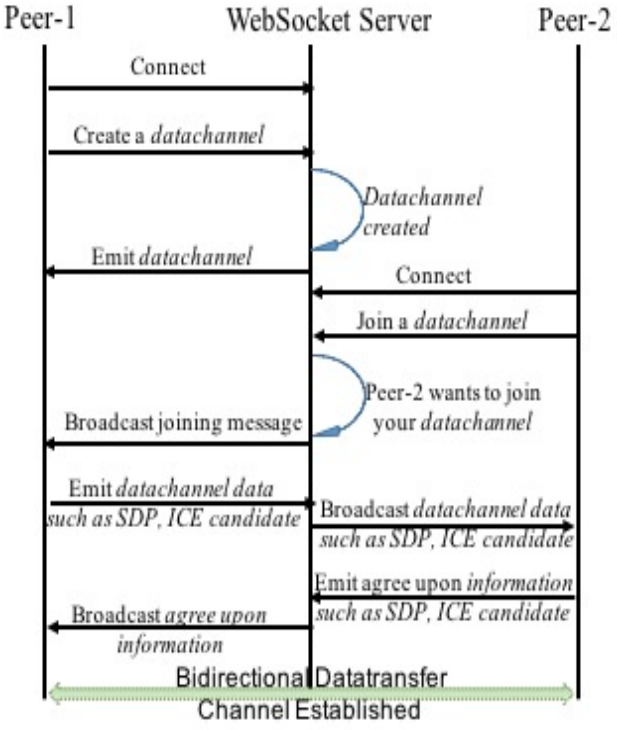

Fig. 7. WebRTC Datachannel Establish Flows

By calculating the DASH representation rates, the proper packet sending rates can be calculated and the self pacing sender pushing session can be executed. This results in a high utilization of the underlying link capacity. Fig. 8 shows the receiver keeping the maximum throughput, which is around 0.9 Mbps. The initial receiving time difference between the sender and receiver is 170 ms, which is not shown in the graph.

By comparing the behavior of the DASH.js client, which is the DASH-IF-Reference client implementation, and our WebRTC based DASH client side playback performance, WebRTC shows a much lower initial streaming start delay, and

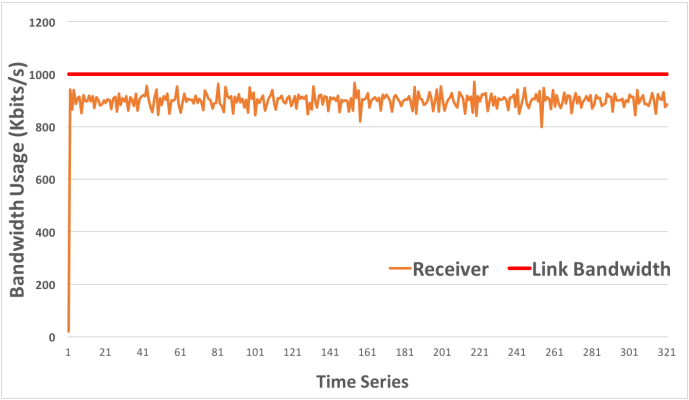

Fig. 8. WebRTC Datachannel Bandwidth Usage

a very high link capacity utilization, resulting in a significant constant high bitrate data segment transfer without showing an ON/OFF pattern like DASH; all very desirable features for supporting very low delay DASH streaming. It validates that use of WebRTC as a transport layer protocol can bring a noticeable advantage for the DASH segment data transfer. By properly exposing the underlying WebRTC congestion measure and control scheme, and integrating with the QoE metric signaling in DASH MPD, a much better QoE driven/optimal low delay DASH streaming solution can be developed.

VI. CONCLUSION & FUTURE WORK

In this work, we utilized the WebRTC data channel to develop a push based streaming solution for carrying low delay DASH content to clients. New protocols and signaling over WebSockets were utilized for the signal plane, while WebRTC data channel was used for content transport. Initial simulation demonstrated very high link utilization and a low delay for DASH over WebRTC as compared with a typical DASH.js client over the HTTP link. In the future, we will further invest and expose the underlying WebRTC congestion model to support better QoS adaptation and introduce new sptaio-temporal QoE metrics to MPD to facilitate better end-to-end QoE optimization. Also, it will be interesting to introduce a thin middleware layer plug-in that enables WebRTC based peer-to-peer DASH streaming.

Furthermore, due to the wide spread deployment of WebRTC in both Firefox and Chrome browsers, a de-facto real time end-to-end transport infrastructure is ripe for further development into a peer assisted video multicasting system, for which several leading industry players and standardization bodies have expressed interest. We will extend our experiments to provide more WebRTC related server and application designs.

REFERENCES

[1] *Information technology MPEG systems technologies Part6: Dynamic adaptive streaming over HTTP (DASH)*, ISO/IECFCD23001-6 ISO/IECJTC1/SC29/WG11 Std.

[2] Viswanathan, Swaminathan, K. Streeter, I. Bouazizi, and F. Denoual, "Working draft for 23009-6: Dash over full duplex http-compatible protocols (fdh)," iSO/IEC JTC1/SC29/WG11/ N15685 October 2015, Geneva, Switzerland.

[3] Server and network assisted DASH (SAND), "Information technology – dynamic adaptive streaming over http (dash) – part 5: Server and network assisted dash (sand)."

[4] A. Bergkvist, D. Burnett, and C. Jennings, "A. narayanan," webrtc 1.0: Real-time communication between browsers," *World Wide Web Consortium WD WD-webrtc-20120821*, 2012.

[5] H. Alvestrand, "Overview: Real time protocols for browser-based applications," 2015.

[6] R. Jesup, S. Loreto, and M. Tuexen, "Rtcweb data channels," *IETF ID: draft-ietf-rtcweb-data-channel-05 (work in progress)*, 2013.

[7] I. 23008-1:2014, "Information technology – high efficiency coding and media delivery in heterogeneous environments – part 1: Mpeg media transport (mmt)." [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=62835

[8] J. P, T. V, and D. Medhi, "Sara: Segment aware rate adaptation algorithm for dynamic adaptive streaming over http," in *IEEE ICC workshop on QoE-FI, London, UK*, June 2015.

[9] ——, "Qoe management in dash systems using the segment aware rate adaptation algorithm," in *IEEE NOMS*, April 2016.

[10] Y. Liu, S. Dey, D. Gillies, F. Ulupinar, and M. Luby, "User experience modeling for dash video," in *Packet Video Workshop (PV), 2013 20th International*. IEEE, 2013, pp. 1–8.

[11] Y. Lim, K. Park, J. Y. Lee, S. Aoki, and G. Fernando, "Mmt: An emerging mpeg standard for multimedia delivery over the internet," *MultiMedia, IEEE*, vol. 20, no. 1, pp. 80–85, 2013.

[12] Z. Li and I. Bouazizi, "Light weight content fingerprinting for video playback verification in mpeg dash," in *Packet Video Workshop (PV), 2013 20th International*. IEEE, 2013, pp. 1–5.

[13] C. Liu, I. Bouazizia, M. M. Hannuksela, and M. Gabbouj, "Rate adaptation for dynamic adaptive streaming over http in content distribution network," in *Signal Processing: Image Communication*. IEEE, 2012, pp. vol. 27, no. 4, pp. 288311.

[14] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz, "Adaptation algorithm for adaptive streaming over http," in *19th International Packet Video Workshop*. IEEE, 2012, p. pp. 173178.

[15] M. K, Q. E, G. G, and W. A, "Atowards agile and smooth video adaptation in dynamic http streaming," in *8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2012)*. ACM, 2012, p. pp. 109120.

[16] C. Vogt, M. J. Werner, and T. C. Schmidt, "Leveraging webrtc for p2p content distribution in web browsers," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013, pp. 1–2.

[17] V. Singh, A. A. Lozano, and J. Ott, "Performance analysis of receive-side real-time congestion control for webrtc," in *Packet Video Workshop (PV), 2013 20th International*. IEEE, 2013, pp. 1–8.

[18] F. Fund, C. Wang, Y. Liu, T. Korakis, M. Zink, and S. S. Panwar, "Performance of dash and webrtc video services for mobile users," in *Packet Video Workshop (PV), 2013 20th International*. IEEE, 2013, pp. 1–8.

[19] I. B. Z. Li and K. Park, "Mmt amd1: Multiple qoe operating points signalling in mmt adc," Patent ISO/IEC JTC1/SC29/WG11/MPEG2014/m33 237.

[20] O. O. D. S. Y. Reznik, A. Giladi and S. Zhang, "Wd of iso/iec: Carriage of quality-related information in the iso based media file format," Patent ISO/IEC JTC1/SC29/WG11/MPEG2013/N13 992.

[21] T. T. Thai, N. Changuel, S. Kerboeuf, F. Faucheux, E. Lochin, and J. Lacan, "Q-aimd: A congestion aware video quality control mechanism," in *Packet Video Workshop (PV), 2013 20th International*. IEEE, 2013, pp. 1–8.

[22] V. Singh, S. McQuistin, M. Ellis, and C. Perkins, "Circuit breakers for multimedia congestion control," in *Packet Video Workshop (PV), 2013 20th International*. IEEE, 2013, pp. 1–8.

[23] L. De Cicco, G. Carlucci, and S. Mascolo, "Understanding the dynamic behaviour of the google congestion control for rtcweb," in *Packet Video Workshop (PV), 2013 20th International*. IEEE, 2013, pp. 1–8.

[24] E. Thomas, M. van Deventer, T. Stockhammer, A. C. Begen, and J. Famaey, *Enhancing MPEG DASH performance via server and network assistance*. Stevenhage: IET, 2015.

[25] C. Jennings, T. Hardie, and M. Westerlund, "Real-time communications for the web," *Communications Magazine, IEEE*, vol. 51, no. 4, pp. 20–26, 2013.