WebRTC-based Peer Assisted Framework for HTTP Live Streaming

Midhul Varma, Hema Kumar Yarnagula and Venkatesh Tamarapalli
Department of Computer Science and Engineering, IIT Guwahati, Guwahati, India
Email: {midhul, h.yarnagula, t.venkat}@iitg.ernet.in

***Abstract*—HTTP based adaptive streaming technologies have become the primary vehicle for delivering live video content on the Internet. The WebRTC stack has opened up the possibility of creating a purely browser based and plugin free mechanism of utilizing peer to peer upload bandwidth to assist HTTP based streaming systems and hence make them easier to scale. Accordingly, we have implemented a WebRTC-based peer assisted framework for HTTP live streaming. We have further experimentally evaluated its performance with varying peer density. Our experimental results reveals that our framework reduces more than three fourth of the server load by serving the segment request from the neighbouring peers.**

I. Introduction

The exploding volume of Internet traffic has made it increasingly important to design efficient and scalable systems, especially for delivering video content to end users. Of late, using HTTP to stream video content has become the norm. The main reasons for the triumph of HTTP over traditional streaming protocols are due to NAT/Firewall friendliness and ability to scale using Content Delivery Networks (CDN). Furthermore, the video content streamed using HTTP can be consumed by HTML5 players without the need of any additional plug-ins or software. Web Real Time Communication, WebRTC for short, is a collection of protocols and APIs whose aim is to enable better RTC on the web. WebRTC's DataChannel API enables peer to peer transfer of arbitrary data between web browsers, using just JavaScript code, running on them. The set up of the peer to peer connection requires a signaling server for exchange of necessary information like IP addresses and port numbers. During the setup phase WebRTC uses sophisticated techniques to traverse any NAT/Firewalls in the connection path. Once the setup phase is successful, all of the data flows directly between peer to peer.

On the other hand, technologies like Dynamic Adaptive Streaming over HTTP (DASH) [1] making it possible to automatically adjust the bitrate of a video stream based on the network conditions. HTTP Live Streaming (HLS) is an HTTP-based media streaming communications protocol related to DASH, supporting the delivery of both Video-on-demand (VoD) and live content. HLS works by encoding the stream at a variety of data rates and breaking the overall stream(s) into a sequence of small segments. The HLS client downloads the segments from any of the encoded streams as HTTP-based file downloads and loading to an overall potentially unbounded transport stream.

Though substantial amount of HLS implementations based on client-server architecture were reported in the recent research literature [2]. WebRTC's DataChannel API opens up the possibility of adding peer assistance to HTTP based streaming systems. In such a system, viewers simultaneously watching a stream can make peer to peer DataChannel connection(s) with each other and utilize them to share fetched video segments among each other. Thus, it will not only reduce the load on the streaming server but also lessen the delay in fetching the segment. Moreover, the most attractive feature of these kinds of systems is their ease of deployment. Peers participating in the streaming service do not require any additional plug-ins or software installation in order to enable peer assistance. In fact very less work has been done on the design and analysis of these kinds of systems in literature. Accordingly, in this work, we have implemented a WebRTC-based peer assisted framework for HTTP live streaming. We have also implemented a P2P controller in the clients to support the peer assistance functionality. We have conducted preliminary experimentation with varying peer density for measuring the peer contribution during the streaming. Our experimental results reveal that our framework can reduces more than three fourth portion of the segment request load from the streaming server, without degrading the stream quality.

II. Implementation Details

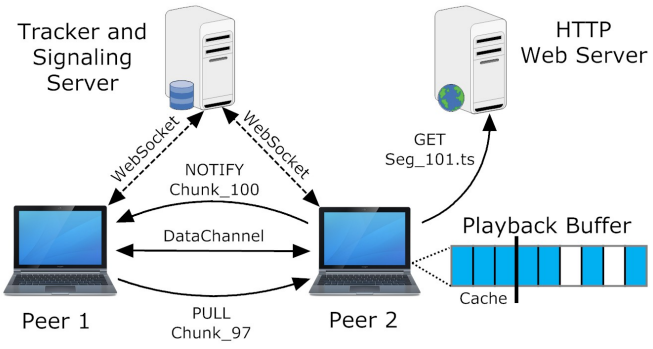

Fig. 1. Pictorial illustration of WebRTC-based peer assisted framework

In this section, we briefly discuss the implementation aspects of WebRTC-based peer assisted framework along with details of our P2P controller. Fig. 1 provides a pictorial illustration of our WebRTC-based peer assisted framework with two peers. We have chosen HLS as our adaptive bitrate streaming technology due to the simplicity in its specification and format in the context of live videos. The streaming server maintains a sliding window containing a fixed number of video segments. New video segments are published periodically at a fixed interval (around 2-4 secs). Whenever a new video

segment gets published, the window maintained by the server slides forward by one segment duration. Information about the sliding window state and the segment URLs present in the window are stored in a metadata file of M3U8 format. There is an master playlist file as well, containing details about different bitrate streams and URLs of their corresponding M3U8 descriptors.

We used *hls.js*[1], Dailymotion's open source HTTP Live Streaming client. It relies on HTML5 video elements and Media Source Extensions (MSE) for playback. In the first place, the client player downloads the master playlist file and parses the same to obtain information about available bitrate streams and segment URLs. It then selects an initial bitrate and downloads the corresponding M3U8 descriptors file. As soon as the descriptors file gets downloaded, it starts fetching the oldest segment present in the window. To keep track of the sliding window and obtain information about newly published segments, client player periodically re-fetches the M3U8 descriptors file. When the player switches to a different bitrate it downloads the corresponding M3U8 file and follows the aforesaid process.

We have also implemented a special P2P controller which takes care of the peer assistance functionality in the framework. We have commented the default *hls.js* segment fetch logic and the functionality will be taken care by our P2P controller implementation. Our framework also contains two key components namely, tracker and signaling server. The tracker keeps track of the peers who are currently watching the live stream. The signaling server acts as the medium for the setup of WebRTC peer connections, which requires exchange of network, session and encryption information between peers. Once the connection is setup between the peers, a DataChannel is established and video data can be transferred directly between them. Both of these components (i.e., tracker and signaling server) communicate with peers using bidirectional WebSocket channels (as shown in Fig. 1). Although the tracker and signaling server, and the streaming server are shown as a separate components in the Fig. 1, they could very well be hosted on the same physical machine.

The peer assistance mechanism for our implementation is very similar to mesh based P2P streaming approaches reported in [3], which have proven to be robust and scalable. When a new peer joins the live stream, the P2P controller initiates a connection with the tracker and sends a bootstrap request. The tracker acknowledges this and sends back a list of randomly selected peers. The P2P controller then establishes a WebRTC peer connection with each of the peers in the list. It makes use of the signaling server for the same. Hence, the peers form a randomly connected unorganized overlay network. Once the peer connection is set up, DataChannels are established for transmission of video content and metadata information (refer Fig. 1). The unit of sharing in the P2P network is called *chunk*, whose size is smaller than that of an individual *video segment*(unit of video content published by the HLS streaming server). These segments can be further divided into smaller units called *chunks*, which are the fundamental units of sharing between peers. Thus, different parts of the same segment can be fetched from different peers. Whenever a peer receives a chunk, it notifies it's neighbours (list of peers to whom it is connected to) using a NOTIFY message over the DataChannel. Hence, every peer will have the up to date information about segments each of it's neighbours possesses. If a peer is in need of a chunk, currently available with one of it's neighbours, it sends a request using a PULL message on the DataChannel. Upon receiving a PULL request, peer transfers the requested chunk over the same DataChannel. As shown in the Fig. 1, the playback buffer at each peer can be logically divided into two parts. Firstly, The playback buffer consists of chunks from segments which are yet to be played. Chunks missing in this part of the buffer need to be fetched either from the server or from neighbour peers. Once a segment has been consumed by the player, it's corresponding chunks enter the cache area. They are cached for a certain additional duration to serve other peers. There is a scheduler running on each of the peers which is responsible for dispatching chunk requests to server/peers. However, for simplicity we have chosen the chunk size same as of the segment size for our current implementations.

[1]https://github.com/dailymotion/hls.js/

TABLE I
SEGMENT FETCHING % OBSERVED UNDER VARYING PEER DENSITY

Peer Density	% of segments fetched from Peers	% of segments fetched from Server
5	76.53	23.47
15	78.96	21.04
30	86.33	13.67

III. EXPERIMENTAL RESULTS & CONCLUSION

The experiments were conducted on our campus LAN with regular background traffic at different times of the day. The experiments are performed with varying peer density, i.e., 5, 15, and 30 respectively. Table I tabulates the average results obtained by multiple experiments under each peer density scenarios. We can observe that under all the three scenarios (i.e., with 5, 15 and 30 peers), the peers pull more then three fourth of their segments from their neighbouring peers instead of from the server. Hence, reducing the server load to a greater extent. Our results also demonstrate that the percentage of segments fetched from the neighbour peers increases linearly with increase in number of peers in the system. Motivated by these initial good results, our future work would focus on designing a bitrate adaptation algorithm to further improve the stream quality and large scale evaluation of the framework.

REFERENCES

[1] T. Stockhammer, "Dynamic adaptive streaming over HTTP - standards and design principles," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 133–144. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943572

[2] A. Fecheyr-Lippens, "A Review of HTTP Live Streaming," *Internet Citation*, pp. 1–37, Jan 2010.

[3] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live p2p streaming approaches," in *26th IEEE International Conference on Computer Communications (IEEE INFOCOM '07)*. IEEE, May 2007, pp. 1424–1432.