P2P Media Streaming with HTML5 and WebRTC

Jukka K. Nurminen, Antony J. R. Meyn, Eetu Jalonen, Yrjo Raivio and Raúl García Marrero
Department of Computer Science and Engineering, Aalto University, Finland
Email: jukka.k.nurminen@aalto.fi, ajrmeyn@gmail.com, eetu.jalonen@aalto.fi, yrjo.raivio@gmail.com, serulo@gmail.com

***Abstract*—Video-on-demand (VoD) services, such as YouTube, generate most of the Internet traffic today, and the popularity of video services is growing. Service and CDN providers have to invest more and more in distribution networks, which creates pressure to innovate novel approaches. Peer-to-peer (P2P) streaming is a viable alternative that is scalable and can meet the increasing demand. The emerging HTML5 standard introduces APIs that give web browsers an ability to communicate directly with each other in real time. New standards also enable a setup, where browsers can act as P2P nodes. This paper reviews whether the new HTML5 and WebRTC standards are a fit for P2P video streaming, evaluates the performance challenges and proposes solutions. Preliminary analysis indicates that HTML5 can be applied to VoD, but there are concerns.**

I. Introduction

The interest towards HTML5 is strongly increasing. Besides new visualization possibilities HTML5 brings a number of new opportunities to run increasingly sophisticated web applications with novel features such as parallel processing, data storage and real-time communication (RTC), without any additional add-ons or plug-ins. These features are available through APIs, which browser applications can call from JavaScript. It is rather clear that simple standalone applications can be implemented with HTML5, but the situation is more challenging when we start to analyze diverse applications with high computation and communication needs.

The main use case we study is video-on-demand (VoD) using peer-to-peer (P2P) approach. Unlike a video conferencing use case, the P2P VoD is a good example of a communication intensive application that requires complex local operations to download and upload streaming content, to manage the download schedule, and to play the content. Similar operations are typical of many distributed applications and therefore the experiences gained have a wider applicability beyond the video streaming use case.

However, implementing P2P streaming with HTML5 has still challenges. Firstly, the standards are under preparation, and therefore browser implementations have issues with supported functions, stability and interoperability. Secondly, video streaming sets substantial requirements on CPU performance and bandwidth consumption. In mobile space these issues are even more critical. This research proposes an experimental system for P2P VoD service applying several new HTML5 APIs that have recently been standardized.

II. Background

There are various ways to implement streaming with P2P technologies. In this work we focus on how to apply BitTorrent and HTML5 for P2P streaming [1]. However, regular BitTorrent is focused on the P2P file sharing rather than the VoD streaming [2]. For that reason a few modifications are required to guarantee the order and timeliness of video pieces and furthermore, to ensure an uninterrupted viewing experience. BitTorrent functionality can be added to web browsers via a few different methods. Plug-ins and in-application web-servers are already available, but our proposed JavaScript based P2P/BitTorrent client is still to be realized. Having a platform independent P2P client, running on PCs and mobile devices and dealing with the issues related with the differing platforms is the goal.

HTML5 Web Real Time Communication (WebRTC) standard plays a key role in browser-to-browser communication. The WebRTC standard enables real-time communication and also introduces UDP based communication to web browsers to complement the normally used TCP communication with HTTP. The standard enables three modes and APIs for browser-to-browser communication: Peer Connection API offers connection establishment, Media Stream API video streaming functions and Data Channel API arbitrary data sharing operations [3].

III. Implementation

The design mirrors the BitTorrent architecture and the same entities Tracker, Seeder and Peers - are used. Figure 1 illustrates the functionality. Initially a Peer creates a torrent file containing the metadata of the video, hashes of each piece of the video, and uploads the torrent file to the Tracker (1). The actual video is uploaded to the Seeder (2). When another Peer requests the same video, the Peer first obtains the respective tracker file and starts downloading the video pieces from the Seeder or the original Peer or both (3). Eventually, as the video becomes popular, missing pieces are shared by Peers, which decreases the load on the original Peer and Seeder. Periodic access to the Tracker for requesting the latest Peer information is needed (4).

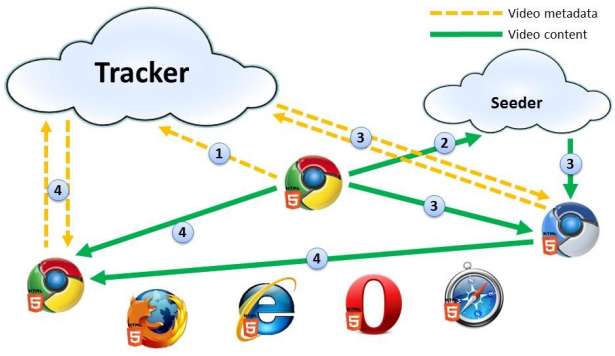

Fig. 1. Network Architecture for the P2P VoD service.

The video publishing process begins when a user starts a web application and selects a video to share. This phase involves the creation of the meta-data of the video file for the Tracker, and moving the file to a location accessible to the web application. Part of the meta-data creation is the generation of cryptographic MD5 hash values. The HTML5 File API is initially used to store the video file to be distributed, and once that process is complete, the file is moved to permanent storage using the IndexedDB API. Finally, once the file hashes have been generated, the newly created torrent file and the data file are sent to a Seeder for wider distribution.

When a user requests a video, the meta-data of the video file (torrent file) is downloaded to the user's browser along with a list of Peers who currently have the video available. Then the web application starts downloading the pieces of the video file from the Peers using the Data Channel API. When enough pieces for the beginning of the video have been downloaded and their MD5 checksum values have been successfully verified, the playback will start using HTML5's Video element. Offline Application Caching API can be used to load and execute a copy of the HTML5 JavaScript code stored on the client, even when the web browser is not connected to the Internet. This API allows playing the stored videos also when no connection to Internet is available.

IV. Feasibility Analysis

During the experiment and writing phases of the original paper, no Data Channel API implementation was available on the mainstream browsers to do testing on (Firefox Aurora releases since November 2012 have had support for WebRTC Data Channel API). Instead, the focus was on the then measurable parts of the project, in particular, the performance of MD5 on Javascript and how much load the video conferencing solution based on WebRTC generated. The calculation of MD5 hashes is computationally intensive, and using too much CPU consumption on this operation can adversely affect the speed of getting content to play and to the power consumption of the system, especially on mobile devices.

The options for improving hashing algorithm performance in browsers include providing a standardized, open API for the more popular hash functions that can be called by web pages. The others are trying to improve the overall performance of Javascript in browsers and lastly trying to improve the Javascript implementations of the hashing functions themselves. The site jsPerf [4] provides comparisons between different browser versions and implementations of the MD5 algorithm, and the tests there indicate that there can be over tenfold performance differences between different browser versions.

There are other considerations as well, such as network and file system performance, but the standard deviation of those is usually much smaller then compared to the more processor intensive operations such as file hashing. The video decoding and encoding processes evaluated with the WebRTC video conference experiment indicated that handling two streams, incoming and outgoing, is manageable with current desktop and near-future mobile devices. Increasing the load further with additional conference peers will severely degrade the frame rate of the incoming streams and will ultimately result in an unusable user experience. Hardware support for the more widely used codecs (H.264, WebM and OGG) can alleviate the situation somewhat. The assumption we can make from this is that a VoD type service implemented with HTML5, with a single incoming video stream from multiple sources, should not be an insurmountable task for even current mobile devices, at least, in regard to video decoding performance.

V. Conclusions

HTML5 can be used to perform P2P video streaming between browsers without any additional plug-ins. However, some limitations exist. Part of the challenges arise from limitations in current browser implementations, part of performance issues. The resources available through browsers are much more limited than for native applications. Further study would be needed to understand how much these limitations are fundamental in the browser technology or simply initial problems of premature implementations. The work does not end by implementing a P2P based solution for media streaming, but performance measurements over various platforms, including mobile devices, must be carried through. Mobile devices offer limited CPU and bandwidth capacity, and thus streaming applications must be well optimized for the scarce resources.

As we do not have a clear understanding of the load distribution between the different elements with an HTML5 based VoD service, the next step is to implement a working P2P VoD client on the latest Firefox versions, find out the relative importance of each part and locate the largest bottlenecks. The performance of the Data Channel API, file hashing and video decoding on the target browser are the main focus areas. The proposed P2P VoD design and implementation only works over a homogenous P2P network where all participants are using the HTML5 based solution, but for the end user it would be useful to get access to existing P2P networks such as BitTorrent. Allowing access to the vast number of existing content sharing and video streaming solutions is another avenue for future work.

Acknowledgment

The work is supported by Tekes (the Finnish Funding Agency for Technology and Innovation, www.tekes.fi) as a part of the Cloud Software Program (www.cloudsoftwareprogram.org) of Tivit (Strategic Centre for Science, Technology and Innovation in the Field of ICT, www.tivit.fi).

References

[1] A. Bakker, R. Petrocco, M. Dale, G. J., G. V., R. D., and P. J., "Online Video Using BitTorrent and HTML5 Applied to Wikipedia," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, Aug. 2010, pp. 1–2.

[2] J. Mol, A. Bakker, J. Pouwelse, D. Epema, and H. Sips, "The Design and Deployment of a BitTorrent Live Video Streaming Solution," in *Multimedia, 2009. ISM '09. 11th IEEE International Symposium on*, Dec. 2009, pp. 342–349.

[3] S. Loreto and S. Romano, "Real-time communications in the web: Issues, achievements, and ongoing standardization efforts," *Internet Computing, IEEE*, vol. 16, no. 5, pp. 68–73, Sept.-Oct. 2012.

[4] M. Bynens, "jsPerf – MD5 Shootout Revision 14," October 2012. [Online]. Available: http://jsperf.com/md5-shootout/14