Peer-to-Peer Video Streaming in HTML5 with WebTorrent

István Koren(✉) and Ralf Klamma

Advanced Community Information Systems (ACIS) Group, RWTH Aachen University, Ahornstr. 55, 52056 Aachen, Germany
{koren,klamma}@dbis.rwth-aachen.de
http://dbis.rwth-aachen.de

Abstract. Multimedia-related businesses dealing with movie streaming and video-based short messages have increased the global Internet video traffic substantially in the last couple of years. At the same time, multimedia on the Web has been standardized in terms of codecs and browser-based JavaScript APIs. However, the technological challenges concerning the distribution of large video files are today mainly tackled by scaling up capacities in cloud data centers, or relying on content delivery networks. Both approaches favor financially strong, large companies, while independent video providers with highly demanded videos are disadvantaged. In this article, we conceptualize methods to offload video delivery from centralized clouds to clients. In particular, we implemented and evaluated a system that streams videos peer-to-peer via WebTorrent in HTML5. The resulting library is available open source.

Keywords: Web multimedia · Peer-to-peer · WebTorrent

1 Introduction

The global Internet video traffic has risen substantially in the last couple of years. The continuous increase of connection speed has paved the way for the success of multimedia-related businesses dealing with movie streaming and video-based short messages. In 2014, 64% of all consumer Internet traffic regarding bits transferred was video traffic [2]. According to Cisco's *Visual Networking Index*, this number will most likely rise to 80% in 2019. Meanwhile, a shift of multimedia technologies on the Web from proprietary plugins like Adobe Flash to a standardized set of HTML5-related standards has taken place. Necessary preconditions for cross-device multimedia on the Web were tackled, including licensing issues around media codecs such as WebM and MPEG H.264. However, technological challenges in distributing large video files are mainly addressed by relying on central cloud providers with large storage capacity, so that the de facto video distribution model still follows the client-server architecture. Video files are first uploaded from client devices to a central cloud solution, before being downloaded or streamed to clients. This entails a number of drawbacks. First,

T. Mikkonen et al. (Eds.): ICWE 2018, LNCS 10845, pp. 404–419, 2018.
https://doi.org/10.1007/978-3-319-91662-0_33

live streaming takes a temporal indirection when routed over a server. Second, it generates a high load on central points of the network infrastructure. Last but not least, video creators must accept the terms and conditions of the cloud provider to serve the video. Although *Content Delivery Networks (CDNs)* solve the first problem, they introduce a further monetary burden, making it hard for small content providers to reach the quality and availability of big providers. Thus, a possible solution lies in offloading the load *from* the cloud *back* to the clients, following principles of Edge computing [11]. Peer-to-peer video streaming is a promising principle to meet these challenges. Although browser plugins like *Adobe Flash* and *Akamai Netsession* have tried to solve this several years go, they have failed due to usability and security shortcomings. Today, two recent group of Web standards are here to solve the challenges natively in the browser. On the one hand, the Media Source Extensions control feeding `<audio>` and `<video>` tags with multimedia data. On the other hand, WebRTC as a group of protocols and APIs solve issues around cross-browser peer-to-peer data streaming.

In this paper, we describe our conceptual considerations and prototype implementation of a WebTorrent-based framework for peer-to-peer video streaming in native HTML5. The results are available as open source solutions, enabling the further development with the help of the open source community.

1.1 Motivation

Online videos have become a huge market over the last years. Large Internet companies like *Facebook* and *Twitter* entered the video business, trying to gain a share of the consumer market dominated by *YouTube*. There is also a movement towards live video on social networking sites. As a social phenomenon, *digital natives* prefer new kinds of social networks like *Snapchat* and *Instagram*, whose business model includes short video "stories" that vanish after a user-defined time period. Besides entertainment and social networking aspects, there is also huge potential in further multimedia-based application cases like distant, collaborative workplaces. Users get more acquainted with local multi-device streaming of videos from mobile computers to the big screens of smart TVs. Simultaneously, video bitrates increase steadily. 4K with a horizontal screen display resolution of around 4,000 pixels is now state-of-the-art in home entertainment, but 8 K is already on the horizon with the recent HDMI specification [13]. Streaming video files needs to tackle a number of challenges:

- Fast initial startup time: There should be no significant lag when starting a stream.
- Random access: It should be possible to switch the position in the timeline.
- Complying with device and network limitations: The resolution of the target device and the bandwidth need to be respected.

This paper is organized as follows. After presenting related work, we evaluate different concepts for peer-to-peer multimedia streaming on the Web. We then elaborate on the conceptual architecture of our solution relying on a tracking

server. Concretely, we decided to use an approach relying on *WebTorrent*. We then present the *OakStreaming* library that enables setting up browser-based peer-to-peer video streams. The library is configurable and extendable. Amongst other parameters, developers may specify a limit on the amount of data each peer is allowed to upload to other peers, to keep a balance amongst the peers.

2 Related Work

This section introduces related work from academia and industry that tackles the challenges of delivering video content through peer-to-peer technologies.

2.1 Content Delivery Networks

High server load and world-wide latency challenges are usually tackled with Content Delivery Networks (CDNs). A CDN consists of a world-wide network of caches that replicate content of a main server. Requests to the central identity are then redirected to the nearest cache [1]. A commercial CDN provider usually delivers content for many different content providers. For the Web user, CDNs are transparent; the content provider, however, needs to pay fees.

Coolstreaming and Akamai NetSession are two representatives of systems that relay the caching to the peers. Lin et al. described and evaluated the peer-assisted CDN Akamai NetSession [8]. The peers of the network need to run instances of the NetSession Interface application. By including a library into third-party applications like download managers, they can also benefit from the network. Running as part of a download manager, a NetSession Interface instance first tries to download the fragments of a file from other peers of the NetSession network. Fragments that cannot be received fast enough are downloaded from dedicated servers of the CDN. Additionally, a running NetSession Interface uploads already downloaded fragments to other NetSession Interfaces.

2.2 WebRTC-Based Prototypes

Högqvist et al. describe and evaluate Hive.js [10], a WebRTC-based, peer-assisted video streaming solution. A central tracking server keeps track which player instances are currently connected. Their software then builds a random graph between all viewers of a video. Periodically, each peer selects one of his WebRTC connections and terminates it. Immediately after that, the peer connects to a randomly chosen peer from a list of potential new neighbors. This list is provided by the tracking server. An evaluation has shown that their system performs well for peer-to-peer streaming with 30 or less peers. Hive.js does not check data integrity of downloaded video fragments.

Gomes Soares et al. describe an implementation of a WebRTC-based, peer-assisted video streaming solution [6]. The system uses an ISP and geolocation awareness concept. Each peer in their system belongs to a WebRTC cluster. Peers who belong to the same provider's network and who are geographically

close are preferably assigned to the same cluster. All peers in the same cluster are directly connected to each other through WebRTC connections. If a peer needs a media fragment, it first tries to receive it via broadcasting the request within its cluster. If the peer is not able to receive a desired media fragment from the peer-to-peer network in time, it requests it from a CDN. The authors' experiments demonstrate that their implementation leads to high fluctuations regarding the percentage of fragments that are delivered through the peer-to-peer network.

Antony J.R. Meyn described another WebRTC-based, peer-assisted video streaming system [5]. Due to the fact that at the time of conceptualization no browser vendor had implemented support for the WebRTC data channel, no prototype was created. Nurminen et al. describe a WebRTC-based, peer-to-peer video streaming system [7]. Similarly, no working WebRTC data channel was available on mainstream browsers. Instead, the authors evaluated the load and the MD5 hashing algorithm to validate data integrity of media fragments that were delivered by peers.

3 Requirements for Peer-to-Peer Video Delivery

This section discusses various models of delivering video streams to viewers. We then show how these can be developed using common Web technologies. Based on these considerations, we list our functional and non-functional requirements.

3.1 Delivery Models

Video content delivery as it is currently implemented on various popular websites like YouTube, Facebook and Vimeo follows a *client-server* model. Users upload the video to a central cloud repository. When other users retrieve the video's website, the provider embeds the URL of the video in the returned HTML page, typically within a `<video>` tag. The browser then requests the video from the server over the specified URL. After that, the server initializes the video stream to the client. In this scenario, the video is always delivered from the server to the client. No data is transferred between any two clients. Although Web caches such as CDNs can reduce the load on the main server, even if a video is watched at the same time on two different client devices side by side, two separate connections to the server are established and the data is downloaded twice. The limitation of the number of requests a server can answer is a significant bottleneck of the client-server architecture. However, traditional client-server systems can easily ensure content integrity and reliable accounting.

The advantage of *peer-to-peer* systems compared to client-server models is that the load is decentralized onto the clients. Peers are connected to each other and retrieve video fragments from neighbor peers ideally. The disadvantages of peer-to-peer systems compared to sufficiently equipped client-server systems are longer initial start-up times as the video source has to be negotiated initially, and more unwanted stuttering or stalling of video play due to the disappearance of other peers during the playback [4]. In a peer-to-peer network, the time span

between connecting and receiving the first byte of a media fragment is relatively large; getting the first fragment from a Web server or CDN is significantly faster.

Peer-assisted video streaming is another delivery method. It is a hybrid model that combines the advantages of client-server like high availability with the load distribution of pure peer-to-peer system [4,6,8]. In a peer-assisted streaming system, if a peer cannot receive a desired media fragment from the peer-to-peer network in time, it downloads it from the source of the media stream, e.g. from a server [10]. The short initial start-up times are possible because in this kind of hybrid solution the first media fragments of media content can be downloaded from the CDN or Web server to enable the start of the video playback as soon as possible. This fallback solution guarantees that all media fragments are received by each peer early enough to avoid stuttering or stalling of video playback.

3.2 Technology

After discussing delivery models of Web videos above, in this section we discuss related Web Technologies used by our system.

WebRTC. WebRTC is a HTML5 API that enables establishing direct peer-to-peer connections between two or more Web browsers [3]. The WebRTC data channel is the communication channel type of WebRTC which enables to transfer arbitrary binary data between Web browsers. In order to use the WebRTC data channel, a website does not need the explicit agreement from the visitor, however the Web application needs to be delivered over a secure HTTPS channel.

To establish a WebRTC connection, signaling data has to be exchanged between the peers [3]. This includes the public IP addresses of each peer and the local IP addresses, if the peers are located in the same LAN. The data is typically exchanged with the help of a signaling server that is reachable from both ends of the connection. When using this approach, both peers connect to the signaling server and use it as a relay for transferring their signaling data to the other peer. The signaling data can also be transferred by other means. For instance, the peer-to-peer WebRTC connection can be established by manually typing in signaling data into an HTML form. Because of firewalls and the functioning of network address translation (NAT), direct IP-based communication between peers is normally not possible on the Web. Therefore, if the two peers are located in different local networks and at least one peer is behind a NAT-enabled router, a so-called STUN (Session Traversal Utilities for NAT) server is necessary to traverse the router so that the WebRTC connection can be established.

Media Source Extensions. The W3C Media Source Extensions specification enables JavaScript to send byte streams to media codecs [14]. Web browsers that support this specification enable JavaScript code to "feed" a HTML5 `<video>` or `<audio>` element with a video/audio stream piecewise. It enables the implementation of client-side prefetching, buffering and adaptive bit rate for streaming media in JavaScript.

3.3 Requirements

Considering the available body of research literature and the capabilities of commercial software described in the previous section, we present a number of functional and non-functional requirements for our system in the following. As of today, all browsers including the initially hesitant Apple Safari support the WebRTC or similar RTC group of standards for peer-to-peer connections on the Web. The Edge browser by Microsoft currently exposes the ORTC interface with similar capabilities; thus the findings of our system should be easily transferable.

Generally, we want to enable streaming videos in a peer-to-peer manner between instances of different browsers. Random access of playback positions should be allowed to enable users to jump freely during the stream. There are different strategies for distributing the video content amongst the peers. *Rarest piece selection* means that those fragments of a video get requested first, that are estimated as rarest in the network. Offloading streams from the cloud to a peer-to-peer network naturally shifts the load onto the clients. We therefore envision having a fair distribution of the load amongst the peers. A ratio defining the proportion of download and upload should be configurable in addition. For example, it is more reasonable to stream from a device that is connected to a stable wired connection than it is from a battery-powered mobile device in a slow mobile network. The respective configuration parameter is the *peer upload limit*. Other parameters include the buffer size, which is the length of seconds of video playback we cache to enable a seamless playback of the video file. Finally, the system to be developed should generate and expose statistical values for further tuning the system. For example, the amount of video data downloaded from other peers and the amount of uploaded data could be logged. The connection to other peers should either be mediated by a central server or be established via manual connection.

Finally, non-functional requirements include the ability to cope with high fluctuations of the available bandwidth. Also, the system should be quite resistant to continuous arrival and departure of peers. This means that the video stream must not slow down or stop completely if a peer leaves the network. We particularly stress the importance of a comprehensible and easy-to-understand documentation of the source code, as the results should be available as open source solution to foster further development through the wider open source community.

4 Conceptual Design

To create a peer-to-peer solution for distributing video data, we evaluated three different architectures: a synchronized look-up table, a distributed hash table and a tracking server.

4.1 Synchronized Look-Up Table

The goal of a synchronized look-up table concept is to collect all available significant information about the network state at each peer in the network. In this

approach, peers publish which fragments they have cached already. Peers use this information to update their own look-up tables where the keep track of the current overall video distribution state. To retrieve needed fragments, peers send out messages to all other peers together with an urgency indicator calculated out of the temporal distance of the currently played fragment. Other peers then react to the message and offer the needed fragment. As a consequence, there is a large number of messages that need to be broadcasted to all participants, thus any naive algorithm would not be scalable to a large number of peers. A solution would be to multicast messages to a subset of peers.

4.2 Distributed Hash Table

The main aim of a distributed hash table (DHT) concept is to make finding a peer which can deliver the desired media fragment possible, reliable and efficient without a need for a central coordination node. Distributed hash tables are decentralized distributed systems which provide a look-up service for key-value pairs [9]. DHTs have a network structure consisting of nodes and connections between these nodes. The look-up functionality of a DHT gets as input an arbitrary key out of a defined key space, and outputs the corresponding value. Each key is (temporarily) assigned to a network node and any participant can retrieve the corresponding value by sending a query message into the DHT network which then gets routed to a node that got assigned this key. Responsibility for the key space is distributed among all nodes of the network in such a way that the quality of service provided by the DHT is robust against continual node arrivals, departures and failures. Moreover, DHTs do not have a single-point of failure, which enables peer-to-peer video streaming systems that use a DHT to be fairly safe against total failures. A popular DHT algorithm and protocol is Chord [12]. Using the key-value storage functionality of DHTs, it can be dynamically stored and retrieved which peer can deliver which video fragments. The mentioned properties of Chord make finding a peer which can deliver a desired media fragment efficient and scalable. Most DHT concepts have similar properties like Chord, making them attractive as a basis for peer-to-peer video streaming systems.

4.3 Tracking Server

Another possible solution to implement a peer-to-peer video streaming system is to connect every peer to a central tracking server that stays in contact with each peer to keep an overview of the network state. No actual video data is transferred from or to a tracking server. For performance reasons, the tracking server could observe, (a) which peer needs which media fragment in the next few seconds, and (b) how reliable each peer has been in the last seconds. On the one hand, a tracking server may calculate which peer should best send which media fragments to which other peer and then send the corresponding orders to the peers. On the other hand, it may also just organize which peer connects to which peer and let the peers send requests for media fragments to other

peers. This approach highly reduces organization efforts of the tracking server and therefore significantly improves the scalability. In [6] as well as in [10], this tracking server approach proved successful. Peer-to-peer video streaming concepts based on tracking servers are tried and tested, which is shown by the fact that every WebRTC-based, peer-to-peer video streaming implementation presented in Sect. 2.2 uses a tracking server. An obvious disadvantage of the tracking server concept compared to the two aforementioned main concepts is that the tracking server is a single point of failure. When the tracking server stops working, no peer can find new peers to connect to.

Torrent Tracker. A torrent tracker is a special kind of tracking server. The torrent concept works with torrent files, which group other files into fragments and identify them by cryptographic hash values. Furthermore, it lists the size (in bytes) of the fragments. Moreover, a torrent file optionally contains additional information such as one or more URLs to torrent trackers. Everyone who knows the hash value of the complete torrent file can request from tracking servers direct peer-to-peer connections to peers which want to exchange fragments of the respective file. Torrent trackers, additionally, enable to build up peer-to-peer connections between two peers that are interested in the same torrent file, taking the role of a signaling server. The set of peers with whom a peer shares a direct peer-to-peer connection are called the swarm instance of the peer. Over the newly established peer-to-peer connections, peers can then request fragments from each other. The protocol defines how peers request file fragments from each other. Peers only request file fragments from peers with whom they have already established a peer-to-peer connection. A torrent file can be tracked by different tracking servers. With the hash value contained in the torrent file, any peer can check the integrity of received fragments.

4.4 Discussion

The preceding discussion of possible concepts for the planned peer-to-peer system revealed that the concept of a synchronized look-up table scales significantly worse than DHT or a tracking sever concept. The existing related work shows that tracking server concepts are decently good studied and work well. DHTs, on the other hand, do not have a single point of failure and new peers only need one connection to any node of the DHT to enter the DHT network. Therefore, using a DHT seems to be a viable solution. Unfortunately, we were not able to find a solid DHT implementation that runs without a plugin in the browser; developing a new one was out of the scope of this work. In the following, we therefore present our solution based on a torrent tracker, named *OakStreaming*.

5 OakStreaming Peer-to-Peer Video Streaming Library

In this section, the architecture and implementation of the planned video streaming system is presented. The system consists of a torrent tracker, a Web server

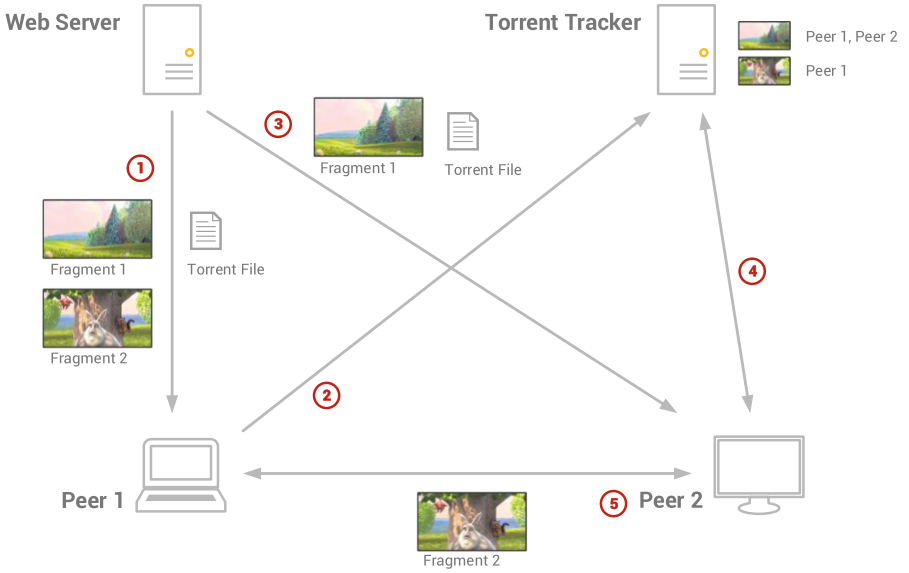

Fig. 1. Overview of OakStreaming architecture and video fragment exchange

and OakStreaming instances. The OakStreaming instances run on the devices of the viewers. In the first subsection, a system overview will be given.

Figure 1 shows the three main parts of our system: the Web server delivering the initial Web application including the OakStreaming client-side library, the torrent tracker responsible for sharing video fragment locations, and the peers interested in watching the video. First, peer 1 retrieves the Web application and the initial video fragments from the Web server (1). The server also keeps a torrent file which includes information about the fragments of the video. The client then announces the availability of fragments to the torrent tracker (2). If a second peer connects to the system by retrieving the Web application (3), subsequent fragments are already available on the first peer. Therefore, the torrent tracker announces the availability of fragment 2 to the second peer (4), which then starts a peer-to-peer connection to the first peer (5). The torrent tracker is also responsible for negotiating the direct peer-to-peer data channel between the peers.

The example above showed the simplest case for retrieving video from the peer-to-peer network. We additionally included several strategies for efficient and effective data transfer. For instance, multiple parameters can be set to limit the amount of uploaded data from a peer; in the example above, the second peer could have also retrieved the first fragment already from the peer-to-peer network. Alternatively, the video file can also be sourced by a participating peer. All parameters are explained in the next section.

5.1 Implementation

In this section, we describe the implementation of the system for peer-to-peer video streaming on the Web. The implementation is based on the WebTorrent JavaScript library[1]. It is an adaptation of the BitTorrent protocol for the Web, using WebRTC connections for exchanging data fragments. We extended its functionality significantly by introducing additional parameters targeting video streams. The OakStreaming library has been developed as a Node.js module which is turned into a Web browser compatible version via the Browserify[2] bundler. The Node.js module only exports a single object which is the constructor to create OakStraming instances; it is global to the browser window's namespace. An OakStreaming instance provides several public methods for the library user; no properties are exposed.

Initiating a Stream. The `createStream(callback, videoFile, options)` method expects one required and two optional parameters. The required parameter is a callback function which gets called with a previously instantiated `StreamTicket` object that contains all the streaming properties. The first optional parameter is the video file which is handed over as a W3C File object. The second optional parameter contains options for the streaming process. If a video file gets handed over to the `createStream` method, it seeds this video file to the WebTorrent network. In this case, additional streaming information is entered through the `options` argument.

Receiving a Stream. Most logic of our OakStreaming library is hidden behind the `receiveStream(streamTicket, callback, stopUploadWhenDownloaded)` method. The required `streamTicket` argument expects a `StreamTicket` object containing peer and Web server connection information. An OakStreaming client can download a video from a Web server and from peers of the WebTorrent network in parallel. The `callback` function gets called upon successful connection. The `stopUploadWhenDownloaded` parameter defines whether upload to other peers should get stopped once the video is fully downloaded locally. To be able to play back all common video formats, each peer repacks received media fragments on-the-fly by using a Node.js module called videostream[3]. A videostream object repacks media fragments and puts them into a source buffer to be played back by a HTML5 `<video>` element.

As soon as the WebTorrent instance has processed a torrent file and found peers, it starts downloading the video in the background according to the *rarest-piece-selection* strategy. The torrent tracker also exchanges the signaling data amongst the peers.

Byte range requests can be conducted by calling the `createReadStream()` method of the WebTorrent API. This method returns a readable stream object

[1] https://webtorrent.io/.
[2] http://browserify.org/.
[3] https://github.com/jhiesey/videostream.

which can be used to (partially) read the requested byte range even if it has not been downloaded completely yet. If the byte range request should not span the entire file, the range can be specified through an argument. Byte range requests which were created from calls to `createReadStream` are fetched as fast as possible and in sequential order from the WebTorrent network. The rarest-piece-selection fragment downloading, as initialized by the creation of a WebTorrent instance, is suspended as long as a stream returned by `createReadStream` has not yet received every byte out of its byte range request.

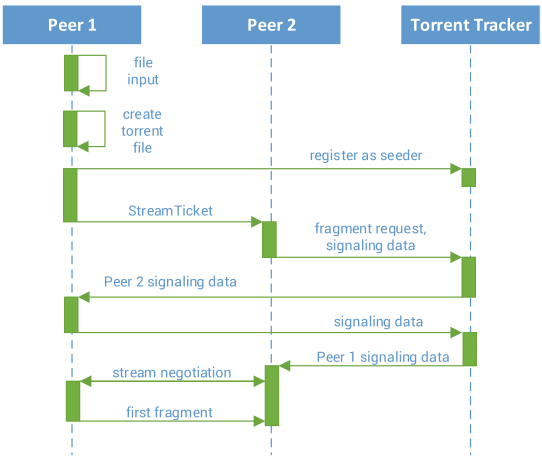

Fig. 2. Establishing an OakStreaming peer-to-peer session

Figure 2 shows the sequence of messages sent around between OakStreaming peers and the torrent tracker. First, a torrent file is created on the first peer by calling the library with a video file. The library then registers the peer as seeder of the video file. To notify the second peer about the availability of the file, a `StreamTicket` including the torrent file is sent to the second peer. The second peer then queries the tracker together with its signaling data. It is forwarded to the first peer, asking for its signaling data in turn. After the first peer's signaling data is presented to peer 2, both start a WebRTC stream negotiation process. Finally, the first fragment of the video is transferred from peer 1 to peer 2.

6 Evaluation

The evaluation of our library is divided into a technical and a user evaluation. In the technical evaluation, we measured how the three implemented features added to the Web Torrent library affected the video streaming. In the user evaluation, we asked seven Web developers to use the OakStreaming API and give feedback regarding the usability of the library, its documentation and peer-to-peer video streaming in general.

6.1 Technical Evaluation

In order to test how well the implemented system is able to organize peer-to-peer streaming, several tests have been conducted with up to eight peers. Each test was conducted with one seeding OakStreaming instance and 2, 4 or 8 OakStreaming instances which should emulate viewers of the video. The video used was a three minute high definition (HD) video that comprised 106 Megabyte. At the start of each test, each OakStreaming instance established a WebSocket connection to a WebTorrent tracker. As implemented in the WebTorrent library, the WebTorrent tracker automatically initializes the WebRTC connection establishment between OakStreaming instances. We have found that the process of querying a neighbor for video fragments could take a significant amount of time, even if the peer-to-peer connection had already been established. Moreover, a peer can only receive file data from its neighbors in fragments whose size is specified during creation of the torrent. This circumstances increase the time from the moment a peer receives a chunk of video data to the moment it serves the received chunk to the viewer. The OakStreaming library uses the default value calculation algorithm of the WebTorrent library for the fragment size. Additionally, after a WebRTC connection between two peers has been established, the WebTorrent protocol needs time to initialize the neighborhood conditions and exchange information which fragments which peer can deliver. Because these factors can cause a delay, the emulated viewers were started with a random time offset. First, an interval from 0 to 10 s was chosen. Using this interval, the average amount of video data that the viewers delivered to each other was relatively low. Therefore, several interval sizes were tested. Besides 0 to 10, the checked interval sizes were 0 to 20, 0 to 30 and 0 to 60. When using 0 to 60 as time offset interval, most data was transferred between emulated viewers compared to the other three; it was therefore chosen as the offset value for all graph-related test runs.

As a result, our tests showed that a lower value for the seconds of video playback to be buffered before the OakStreaming client switches from sequential-piece-selection to rarest-piece-selection leads to a higher overall download time. This correlation was expected and confirms the usefulness of this setting.

We also tested pure peer-to-peer delivery versus peer-assisted delivery regarding the average time the playback was stalled. As expected, peer-assisted delivery significantly reduces stalling time; here, video playback was only interrupted during initial start-up.

To summarize, the results of the technical evaluation clearly indicate that peer-assisted delivery and automatic switching between sequential-piece-selection and rarest-piece-selection enhance the quality of service of the peer-to-peer video streaming system. When the peer-to-peer network consisted out of two peers, the average start-up time for pure peer-to-peer streaming was 1086 milliseconds. In case of peer-assisted streaming this number went down to 667 ms. Measurements with four peers in a pure peer-to-peer environment resulted in an average start-up time of 936 ms. In case of peer-assisted streaming this number went down to 695 ms.

6.2 Developer Evaluation

The aim of the developer evaluation was to test the usability of the OakStreaming library by asking potential library users for their opinion about the design of the library and peer-to-peer streaming in general. The seven participants drawn out of the pool of student workers at our department had intermediary to advanced knowledge and experience in the areas of torrent-based peer-to-peer systems, peer-to-peer systems in general, video streaming/hosting and JavaScript. The lab experiment comprised three programming task and filling out the evaluation questionnaire. The programming tasks were designed to make the participants familiar with the API and functionality of the OakStreaming library. The questionnaire mainly aimed at collecting data about the usability and documentation of the OakStreaming library as well as general opinions regarding peer-to-peer video streaming.

Session Setup and Programming Tasks. Although the participants had to solve the same three tasks, they were asked to work on them individually. The setup was the same for all three programming tasks. The final Web application of each task should be tested in two to three Web browser windows. The participants could conduct these tests independent from each other on their own device.

The first programming task was to create a Web application which uploads and downloads a video to and from a Web server. The second and third programming tasks then both focused on completing the program code of a Web application which streams a video peer-to-peer. The peer-to-peer connections were established locally between the browser windows on the devices of the participants.

In task 2, the participants had to use the `streamVideo` method of the OakStreaming library to create a `StreamTicket` object from a video file. The object was then shared over the a synchronized data structure with the Yjs collaboration library[4]. The received object was then put into the `receiveStream` method of the peer instances.

Task 3 was very similar to task 2 but the participants had to use different parameters and parameter values when creating the `StreamTicket` object. In contrast to task 2, the video should be streamed in the peer-assisted mode and the participants had to set a value for the ratio of time downloaded from server versus peer-to-peer. The parameter values of task 2 and 3 were given by the task description. To solve task 3 it was necessary to read parts of the OakStreaming documentation. Finally, an evaluation form was filled out by the participants.

It asked the participants about their knowledge and experience in the areas of torrent-based peer-to-peer systems, peer-to-peer systems in general, video streaming/hosting and JavaScript. Moreover, the participants were asked to rate the usefulness of several features that the OakStreaming library implemented on-top of WebTorrent. Additionally, the participants had to rate the usability

[4] http://y-js.org.

of the OakStreaming API documentation. Furthermore, the participants were asked about their opinion regarding peer-to-peer video streaming in general.

Results. Five of the seven participants were able to solve all three tasks. Two participants were only able to complete after short clarification. The questionnaire revealed minor issues of the arguments, like putting together URL and port properties. While working on the tasks, several participants remarked that the hostname and the port of a Web server should not be separate parameters. After the evaluation the respective API was changed to a single URL property instead, which can now handle strings in several formats (e.g. http://example.com:42, http://example.com, example.com:42, etc.). Overall, all features of the OakStreaming library were considered easily understandable.

We were also interested in general opinions of our developers on peer-to-peer video streaming. Most of the participants rarely publish or share, but often consume Web videos. They saw many important advantages of peer-to-peer video streaming like benefits for small content providers with successful videos and the breaking of the monopoly of large content providers in terms of intellectual property rights. Additional remarks of the respondents covered legal issues if possibly illegal videos are streamed between peers.

7 Discussion and Future Work

This paper presented OakStreaming, a peer-to-peer video streaming system for the Web. The three main motivations for developing it were to reduce server load compared to state-of-the-art client-server-based video streaming systems; to avoid transfer of intellectual property to a third party; and to maintain a reasonable quality level for the viewer. Since all major browser manufacturers have implemented the W3C WebRTC or similar specifications, some commercial WebRTC-based peer-to-peer video streaming systems have been developed, while solutions by the academic community lack many desired features. We discussed the three alternative concepts synchronized look-up table, distributed hash table and torrent tracker. Finally we implemented the WebTorrent based peer-to-peer video streaming system called OakStreaming. Our evaluation has shown that the peer-assisted modus representing a hybrid scenario with peer-to-peer video delivery and a fallback server significantly reduces video playback start-up time. Peer-to-peer video streaming functionality has been implemented based on the content delivery functionality of the WebTorrent library. The OakStreaming library extends the WebTorrent library by various means:

- configurable limit on the amount of data each peer is allowed to upload
- configurable parameter specifying when the client switches from sequential-piece-selection to rarest-piece-selection
- dynamic combination of server and peer-to-peer streaming (peer-assisted delivery)
- possibility to easily add new client instances to an existing peer-to-peer network, without using a torrent tracker, by explicitly exchanging signaling data

The main challenge during the implementation was that in third-party libraries, many properties were only described implicitly in GitHub issues and not in the official API descriptions.

The results of the technical evaluation clearly indicate that peer-assisted delivery and automatic switching between sequential-piece-selection and rarest-piece-selection enhance the overall Quality of Service (QoS) of the peer-to-peer video streaming system. In the developer evaluation, the usability and the design of the OakStreaming library were positively assessed by the participants. Moreover, the results of the questionnaire have shown that all participants agreed that peer-to-peer video streaming will become more important in the future. The results are helpful to set further goals regarding research in the area of Web-based peer-to-peer video streaming. Limitations of our work include further evaluation in larger developer groups. Privacy aspects in terms of sharing peer information were neglected by our research.

7.1 Future Work

We are planning to embed the library in various other Web application prototypes, to measure the long-term effects and behavior of the library. Today's Web videos are often streamed with an adaptive bitrate streaming technology, which downloads sections of the same video in different bitrate versions depending on the network conditions of the client. The DASH (Dynamic Adaptive Streaming over HTTP) technique is an international standard that enables adaptive bitrate streaming with conventional HTTP servers. Implementing adaptive bitrate streaming within a peer-to-peer network with a satisfying viewer experience is not trivial, as respective fragments have to be available on peers. To the best of our knowledge, there is no implementation which offers an adaptive bitrate in a WebRTC-based peer-to-peer video streaming system. Therefore, adding support for adaptive bitrate streaming to the OakStreaming library is a promising aim for future research and development.

Acknowledgements. We would like to thank our student Philipp Bartels for his contributions towards the implementation of the prototype and we are grateful for the feedback received in our evaluation and the review. The work has received funding from the European Commission's FP7 IP Learning Layers under grant agreement no. 318209 and from the European Research Council under the European Union's Horizon 2020 Programme through the project "WEKIT" (grant no. 687669).

References

1. Buyya, R., Pathan, M., Vakali, A.: Content Delivery Networks. Lecture Notes in Electrical Engineering, vol. 9. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77887-5
2. Cisco: Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019 White Paper (2015)

3. Hickson, I., Bergkvist, A., Burnett, D.C., Jennings, C., Narayanan, A., Aboba, B.: WebRTC 1.0: Real-time Communication Between Browsers: W3C Working Draft, 28 January 2016. https://www.w3.org/TR/2016/WD-webrtc-20160128/
4. Li, B., Xie, S., Qu, Y., Keung, G.Y., Lin, C., Liu, J., Zhang, X.: Inside the new coolstreaming: principles, measurements and performance implications. In: IEEE Conference on Computer Communications, pp. 1031–1039 (2008)
5. Meyn, A.J.: Browser to Browser Media Streaming with HTML5: Master's Thesis. Aalto University (2012)
6. Nogueira Barbosa, F.R., Gomes Soares, L.F.: Towards the application of WebRTC peer-to-peer to scale live video streaming over the internet. In: Simposio Brasileiro de Redes de Computadores (SBRC) (2014). http://sbrc2014.ufsc.br/anais/files/wp2p/ST4-1.pdf
7. Nurminen, J.K., Meyn, A.J.R., Jalonen, E., Raivio, Y., Garcıa Marrero, R.: P2P media streaming with HTML5 and WebRTC. In: IEEE Conference on Computer Communications Workshops (Infocom Workshops), pp. 63–64 (2013)
8. Papagiannaki, K., Gummadi, K., Partridge, C., Zhao, M., Aditya, P., Chen, A., Lin, Y., Haeberlen, A., Druschel, P., Maggs, B., Wishon, B., Ponec, M.: Peer-assisted content distribution in Akamai netsession. In: IMC 2013 Proceedings of the 2013 Conference on Internet Measurement, pp. 31–42 (2013)
9. Rescorla, E.: Introduction to Distributed Hash Tables (2006). https://www.ietf.org/proceedings/65/slides/plenaryt-2.pdf
10. Roverso, R., Hogqvist, M.: Hive.js: browser-based distributed caching for adaptive video streaming. In: IEEE International Symposium on Multimedia, pp. 143–146 (2014)
11. Satyanarayanan, M., Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., Hu, W., Amos, B.: Edge analytics in the internet of things. IEEE Pervasive Comput. **14**(2), 24–31 (2015)
12. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. **11**(1), 17–32 (2003)
13. Sugawara, M., Choi, S.Y., Wood, D.: Ultra-high-definition television (Rec. ITU-R BT.2020): a generational leap in the evolution of television [Standards in a Nutshell]. IEEE Signal Process. Mag. **31**(3), 170–174 (2014)
14. Wolenetz, M., Smith, J., Watson, M., Colwell, A., Bateman, A.: Media Source Extensions: W3C Candidate Recommendation 12 November 2015. https://www.w3.org/TR/2015/CR-media-source-20151112/