P2P Live Video Streaming in WebRTC

Florian Rhinow, Pablo Porto Veloso, Carlos Puyelo, Stephen Barrett, Eamonn O Nuallain
School of Computer Scienceand Statistics,
Trinity College Dublin

***Abstract*—In this paper, we analyse the feasibility of implementing live video streaming protocols into web applications with the use of WebRTC. As a result of demand the distribution of video content requires ever increasing bandwidth. Although, specialised programs exist to distribute video content efficiently, web pages have up until recently not been able to leverage these technologies. WebRTC could serve as a solution by enabling peer-to- peer communication directly between browsers without any need for a server as an intermediary. The feasibility analysisis accompanied by a practical implementation of a peer-to-peer streaming protocol in WebRTC that runs natively in all browsers and an identification of optimal settings for such a protocol. Our work highlights current limitations and future challenges when implementing sophisticated peer-to-peer solutions using a technology that is still in its infancy. Finally, we provide preliminary experimental data on WebRTC which measures the performance of such a system in a laboratory environment.**

I. Introduction

With the increased bandwidth that is available to end consumers, online video streaming has experienced a huge increase in recent years. It has been recently predicted that soon ninety percent of the Internet will be video [1]. This estimate is not only based on increasing demand, but video streaming also consumes significantly greater bandwidth than other traffic. This results in high costs for the providers of online video. It is estimated that video platforms such as YouTube pay up to a million dollars per day for their bandwidth consumption [2]. To reduce costs and increase performance, video content providers often deploy Content Distribution Networks (CDNs) is which provide a shared distribution infrastructure in multiple locations close to end-users. Like many other service providers, CDNs strive to provide greater bandwidth efficiency and performance. Currently, there are 28 commercial CDN providers that use increasingly powerful servers in this competitive market. This shows the increase in importance of distributing video over the Internet.

To reduce bandwidth required for internet video streaming, it was initially proposed to use the IP-Multicast protocol. However, IP-Multicast was never deployed because it was considered to be impractical [3]. As a result, research has shifted its focus to peer-to-peer multicast protocols like Coolstreaming which have been successfully deployed for large-scale video streaming events. However, many popular web applications such as Youtube or Netflix do not leverage these protocols to reduce their server loads. The main reason for this is that up until recently web applications were not able to instruct clients to send data in a peer-to-peer fashion without the use of external applications such as browser plug-ins. With the introduction of Web Real-Time Communication [4] by the World Wide Web Consortium (W3C), this may change. WebRTC is a standard of the W3C that allows for real-time communication between Internet browsers. Establishing a communication channel between browsers is divided into two steps:

- Multiple browsers visit a website that makes use of WebRTC;
- The website server sends the browser IDs to the visitor;
- Using this ID, browsers can then connect to a specific peer browser.

As a result of this the overlay network does not need to know any IP Address with which to establish connections. Instead, the WebRTC API is used for connection management. According to the WebRTC draft, this API is built into browser applications that can make use of real-time communication by calling the appropriate JavaScript functions in the API. Browsers are expected to implement the standard once finished. However, due to the draft status, just a handful browsers have only partially implemented the standard to date. These include Google Chrome and Mozilla Firefox. Besides browserto- browser communication functionalities, WebRTC offers API calls such as *getUserMedia*. These allow access to the microphone and camera of the client. Additionally, most WebRTC demos and recent WebRTC start-ups focus on video telephony and communication. Hence, it is clear that simple peer-topeer applications between two or three parties can be easily implemented with WebRTC. This paper examines how well WebRTC is suited for more complex peer-to-peer algorithms.

This paper is divided into four parts. First, we present the related work of various streaming protocols and the WebRTC standard itself. Second, we outline the implementation of our live video streaming application using WebRTC as well as its limitations. Finally, we evaluate our implementation with a set of experiments and discuss the feasibility of large-scale peerto-peer video streaming application using WebRTC.

II. Related Work

In the following, related work of both video distribution algorithms and WebRTC are presented. Comparatively little research has been conducted on WebRTC due to its novelty whereas solutions for distributing large file streams date back to 1985.

978-1-4799-3351-8/14/$31.00

A. Video Distribution Algorithms

IP Multicast was initially proposed to lower costs and increase available bandwidth for services such as video streaming. IP Multicast is designed to allow servers to send data to multiple destinations. In the standard, routers analyse the destinations of an IP multicast packet and duplicate the packet if the destination needs to be routed to different physical port. Although IP Multicast was proposed back in 1985 it is still not widely deployed. A study [3] suggests that there are multiple reasons for this: unclear implementation strategies, insufficient commercial requirement and the complexity of the design. Since IP Multicast will probably never be widely deployed, tree-based peer-to-peer infrastructures were proposed as a solution [5], [6]. These overlay networks relay the data among nodes in a peer-to-peer fashion, the data being spread from the root to its leafs. It has beeen shown that the performance of tree-based architectures can reach theoretically optimal metrics [7]. However, they also have features that make them unsuitable for use in peer-to-peer video streaming solutions:

- Tree-based protocols are highly vulnerable to *churn*. Given that one of the upper nodes leaves the network, the tree has to be restructured. Due to the highly dynamic nature of the Internet such a system is an inappropriate solution. solution.
- Given that the upload capacity of nodes is much lower than the capacity of the server, there exists an unfair distribution of the available bandwidth for each node.

Since 2005, new categories of pull-based streaming protocols emerged such as Coolstreaming [8], [9] and GridMedia [10], [11], which were similar to systems like BitTorrent [12]. These protocols can be seen as the first successful Internet peer-to-peer video streaming protocols [9]. Coolstreaming was commercially used by software like PPlive [13] and Sopcast [14]. Initially, Coolstreaming and GridMedia were implemented as a pull-based protocol. Both overlay networks compensated for the shortcomings of tree-architectures by splitting video streams into chunks with peers requesting each of these chunks in a pull-based manner [8]. Pull-based protocols are divided into two parts: an initialisation phase and a data transfer phase [10]. In the initialisation phase, the client connects to a server and is supplied with a list of randomly chosen nodes. The client then connects to these nodes, which in turn become its neighbours. Thus, a pseudo random mesh is generated in which every node has a random set of neighbours. In the transfer phase, two types of packets are exchanged: control packets and streaming packets. Streaming packets contain a chunk of the video stream and the sequence number of that chunk. Streaming packets are interchanged on request with request packets.

Request packets and buffer map packets are both control packets. Request packets contain all sequence numbers that a node wishes to obtain from a specific neighbour, while buffer map packets are sent periodically and contain all the sequence numbers a node has in its buffer. Nodes generate request packets, by comparing their buffer to their received buffer maps. Then, the node generates a request packet automatically for all missing streaming chunks and addresses them to the neighbour which sent the buffer map packet. If two neighbours offer the same streaming packet, analogous to the FIFO principle, the browser requests the streaming packet from the neighbour that first informed the browser of the presence of the packet. A packet is requested again, if the browser does not receive within a certain threshold [10].

Both systems Coolstreaming and GridMedia have become increasingly widespread [15]. With their rise in popularity their architecture underwent several revisions. Currently both technologies leverage hybrid push-pull-based protocols which divide streaming chunks into different streaming groups. If a peer successfully pulls a chunk from a neighbour, it automatically subscribes to the entire sub-stream that consists of all chunks of a streaming group. In this manner only one chunk is requested from a neighbour and every subsequent chunk of the same group is pushed by the neighbour to the peer. Similar to the basic protocol, peers know about the availability of various chunks through notifications, so called buffer maps, which are sent periodically by all peers to their neighbours [8], [9].

Although initially Coolstreaming and GridMedia begin with a mesh structure, peers organise themselves into highly dynamic parallel trees. Thus, when observing individual streaming chunks/streaming groups an ordered tree-based structures is generated. Similar to the performance of the above mentioned pure tree-based architectures, Coolstreaming and GridMedia have also been shown to allow high performance and upload utilisation of peers [16]. They also offer a high resistance to churns which make them strong candidate for online video streaming applications. Apart from streaming pushpull- based protocols, which, from a data stream perspective eventually form multiple tree-based structures, there has also been a debate about whether pure mesh-based structures are able to achieve a similar performance with a high resistance to churn. However, recently Picconi *et al*. [15] have shown that solely mesh-based overlay such as the DP/LU algorithm, can achieve near optimal performance metrics as well.

Our system uses a simplified pull-based protocol, which consequently shares many of the basic properties of Coolstreaming [8], [9] and GridMedia [10], [11]. However, while this previous work used stand-alone clients for peer-to-peer video streaming, we suggest the use of communication directly between browsers by using WebRTC to allow video-providing websites to leverage these peer-to-peer technologies. Apart from showing how pull-based video streaming protocols can be implemented in web pages, we also provide preliminary experimental data, while highlighting current limitations and future challenges of this new scenario.

B. WebRTC

WebRTC is currently under development by the W3C [4] and the IETF [17], but parts of the draft specification have already been partially implemented in Google Chrome and Mozilla Firefox.

Whereas, previous W3C web standards allowed communication only between a browser and a server, WebRTC allows direct communication between browsers, without any server in the middle. This theoretically allows for the implementation of peer-to-peer algorithms such as Coolstreaming or GridMedia.

The WebRTC API currently by Google Chrome and Mozilla Firefox consists of the following data communication functions [4]:

- RTCPeerConnection: The RTCPeerConnection interface provides a connection between peers to exchange data.
- MediaStream: The MediaStream interface represents a stream of audio or video data.
- RTCDataChannel: The RTCDataChannel interface describes a full-duplex data connection between two nodes.

Because of the novelty of the specification, implementations of WebRTC are largely experimental and as a consequence little research has been done on WebRTC. Eriksson *et al.* [18] and Loreto *et al.* [19] however summarised the current state of the specification and evaluated it. Nurminen *et al.* [20] investigated the feasibility of peer-to-peer BitTorrent system using WebRTC. The system was intended to be used to upload and download online videos. Nurminen *et al.* were not able to implement the prototype because parts of the WebRTC specification were not available in browsers at that time. They concluded that in theory, such a system can be implemented but that to date too many limitations exist to efficiently construct such a system.

While Nurminen *et al.* proposed to use WebRTC in BitTorrent systems, we focus on evaluating WebRTC for live video streaming by using a pull-based protocol. As a result, instead of eliminating the need for servers by storing the data in the network, we aim to reduce the server load. Additionally, we present a working implementation and preliminary data.

III. Design

WebRTC is the first API that can enable peer-to-peer distribution of video content natively in browsers. Given the variety of available protocols outline in Section II-A, in the following we want to discuss which protocols may be most appropriate for video streaming using WebRTC.

The goal for the chosen protocol is primarily to enable video streaming without interruption. As a result, protocols which focus on maximising peer upload capacity and handling of dynamic nodes are much more suitable for online video streaming than protocols which, for example, focus on latency. Hence the implementation was completed using a simple pull-based protocol which is based on the description given by Zhang *et al.* [10]. This choice was made based on the following:

- State-of-the-art video streaming protocols are in essence based on a simple pull-based protocol [10].
- It has been shown that pull-based protocols with carefully adjusted system parameters can achieve nearly optimal peer upload utilisation and short initial buffering delays [10], [16].
- With regard to the limitations mentioned in Section IV, pull-based protocols offer a simple implementation that is well suited a prototype implementation in a novel technology such as WebRTC.

IV. Limitations of WebRTC

Currently WebRTC suffers from a number of limitations which are outlined in the following. Nurminen *et al.* [20] already discussed several limitations which hindered them. However, while we were able to implement a prototype there were still multiple restrictions involved:

- There is currently a interoperability issue between browser. This led to implementation difficulties among browsers. The library PeerJS for example currently supports the Google Chrome browser only, because WebRTC is implemented in Mozilla Firefox differently.
- The browser implementations are currently in alpha or beta status and as a result have a number of bugs and may terminate unexpectedly
- The WebRTC API does not yet offer functions for connection management and establishment. Instead, a second communication channel is necessary to establish a connection. We were using XmlHttpRequests and WebSockets to overcome this limitation.

V. Implementation

In the following, we will outline how we implemented WebRTC for live video streaming. WebRTC is built into browsers as an API and JavaScript needs to be used to call functions of API. The most suitable function for our implementation was the RTCDataChannel interface, because it allows us to send arbitrary data. Additionally, our implementation uses a modified version of the experimental PeerJS JavaScript library [21]. This library wraps many of the native WebRTC functions into custom JavaScript functions and provides functions to establish a WebRTC connection between peers.

To resemble most of the properties of pull-based protocols discussed in Section II-A our implementation is divided into two phases: Initilisation and Data Transfer.

A. Initilisation

The purpose of our *WebRTC network* is mainly to distribute streams of the same video content. This video content is distributed by a main server, to which we refer to as *source publisher* in the following. We use a *WebRTC coordinator* is used to maintain the *WebRTC network* in which client support the *source publisher* in content distribution. In our implementation the *WebRTC coordinator* and *source publisher* are the same host.

In addition to the mentioned server, we us a *PeerJS server* to support the *WebRTC coordinator* in establishing a connections between to peers. WebRTC does not provide native API handlers to coordinate the communication, such as establishing and terminating the connection. However, there are JavaScript libraries available, which automate the connection management. In our implementation we use PeerJS 0.2.8 [21].

The connection management itself is quite straightforward Every client that wants to join the peer network has to register at the PeerJS server and must provide an unique ID. Other peers that register can then connect to previously registered peers by using their ID. To establish a connection, the PeerJS server then uses XmlHttpRequests and WebSockets. No further connection to the PeerJS server is required, after a connection is established between peers, instead they directly communicate by using the RTCDataChannel of WebRTC.

A peer generates a user ID and registers at the PeerJS server and the *WebRTC coordinator*. The WebRTC coordinator selects pseudo randomly two peers that are already part of the *WebRTC network* and sends these information to the connection establishing peer. This peer then opens a RTCDataChannel to these two peers and thus is part of the WebRTC network. As a result, the first peer only subscribes to the *source publisher*. The second peer subscribes to the *source publisher* and the first peer, while the third peer selects randomly two peers from the first, second or *source publisher*. Thus, a random mesh is created with every new peer which joins the network.

In essence, while any peer can publish to multiple subscribers, every peer (except for the first peer) subscribes to exactly two other peers. Consequentially, peers, which have joint the *WebRTC network* first are closer to the *source publisher* and have naturally more subscribers. Likewise, peers which stay in the network for an extended periods, also gain more subscribers.

This behavior ensures, that the peers with most subscriber are also more likely to be more stable and closes to the *source publisher*.

B. Data Transfer

To capture the essence of pull-based protocols as described by Zhang *et al.*. [10], our pull-based protocol uses two types of packages: *BufferMap*and *PacketRequest*. Both types pf packets are are exchanged periodically among publishers and subscribers. A BufferMap contains the sequence IDs of all data chunks which the peer has in its buffer. Peers which are also *publishers* periodically send BufferMap packets to all their subscribers. When a subscriber receives a BufferMap package, it compares the BufferMap's content to its own buffer. Request packets are sent automatically by subscribers to the appropriate publisher for all missing streaming chunks that a peer discovers in its own buffer map.

Peers communicate with each other by using the RTCDataChannel, which allows to send arbitrary data. We decided to use the JSON format to encode our messages, because it is easily parsed by JavaScript. Because high packet loss can severely affect viewing experience, we are simulating a reliable data transfer of all packets by using TCP in our experiments. This can in future experiments be complemented with distortion-buffer optimized TCP as described by Sehgal *et al.*. [22] or replaced with a custom UDP protocol implementation.

VI. Evaluation

In the following, we present three experiments that we conducted to measure the performance of our prototype with different parameters.The metrics that we have used for our experiment were the following:

- Total packets sent: All data sent by each node on average in bytes per second.
- Control packets received: Control packet received by each node on average in bytes per second. Control packets are either buffer map packets or request packets.
- Redundant packets received: Redundant packets received by each node on average in bytes per second. Redundant packets are streaming packets that the node has already received.
- Delivered packets received: All received streaming packets that are not redundant by each node on average in bytes per second.

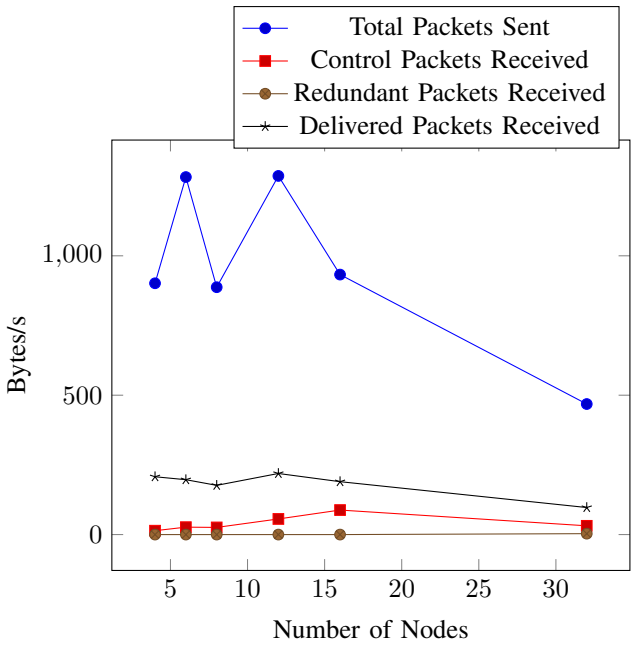

Fig. 1. Transmission properties with a different sets of participating nodes

Figure 1 shows the results of our first experiment. The graph shows how our metrics are changing with an increasing number of participating nodes. We examined the following number of nodes: 2, 3, 4, 6, 8, 12, 16, and 32 nodes. In the experiment, we were using a file with the file size 50 Kilobytes. The metrics were determined by measuring the time that it took each node to receive the packet and the total number of received packages of each type.

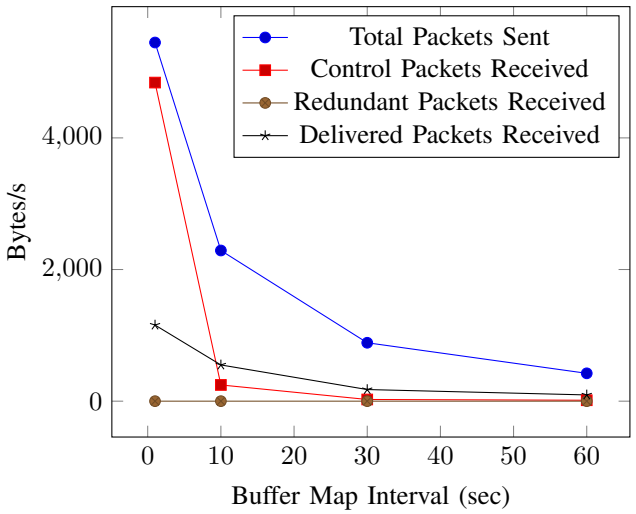

Fig. 2. Transmission properties with a different buffer map intervals

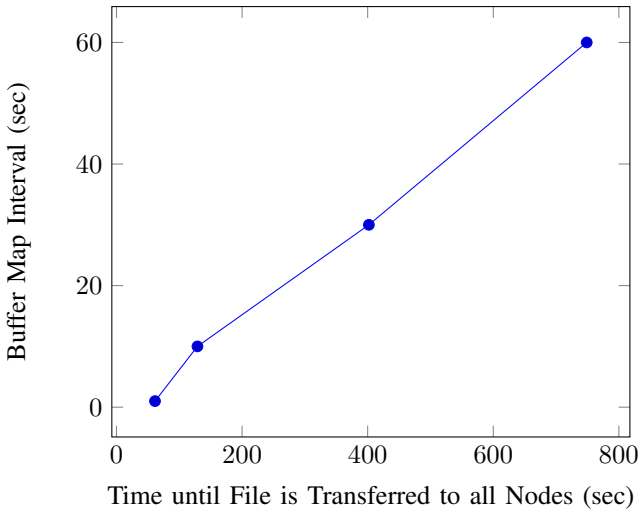

Fig. 3. The time it takes until all nodes successfully received a 50KB file with different buffer map intervals

Firstly, Figure 1 shows, that the combined value of control packets received, redundant packets received and delivered packets is much smaller than the value of total packets sent. This is mainly due to the high loss in the UDP data transfer. What might seem unusual is, that there are no redundant packets until we deploy 32 nodes. This is mainly due to our the long interval of 30 seconds in which we are sending buffer maps and because of the probability of two neighbours having both parents with the same list of packets. If that happens to be the case, the node requests the package twice with our simple protocol implementation.

Figure 2, and Figure 3, show the results of our second experiment. Figure 2, shows how our metrics change when using different intervals for sending the buffer map. While we were using an interval of 30 seconds in the previous experiment, this experiment examines our metrics using values of 1, 10, 30 and 60 seconds. For the experiment, we set the number of nodes to eight, while each node has two neighbours and the file transferred is 50 Kilobytes. Figure 3 shows that the files reaches all nodes much more quickly with a decreasing buffer map interval. However, Figure 2 shows that the overhead of control packets increases significantly with shorter buffer map intervals. This reveals a trade-off between overhead and delay that is regulated by the buffer map interval. Both graphs show that a 10 second buffer map interval has a delay that is only marginally greater than when using an interval of 1 second, yet it leads to a far smaller overhead. Another interesting observation that can be made is that if the buffer map interval is set to a value of 30 or 60 seconds, then only about 2-3% of all packets are control packets, while with a buffer maps interval of 1 second, 43% of all packets are control packets. 10 seconds seem to be an optimal parameter for our experimental setting.

VII. Conclusion

Our implementation shows that peer-to-peer streaming is currently possible for more complex algorithms and larger sets of clients. However, the experiments and implementation have also shown, that with the current limitations of WebRTC such an implementation is currently not practical. Browser vendors need to undo their performance restrictions, so that sophisticated peer-to-peer protocols can be developed for WebRTC.

In future work, we intend to implement video streaming protocols similar to GridMedia and Coolstreaming to significantly increase performance. Our paper has shown that pullbased protocols can be implemented in WebRTC. Thus, it seems very likely that current hybrid push-pull protocols can be implemented as well due to the many similarities to simple pull-based protocol. The only remaining challenge is when building large-scale systems. However, although, no browser has completely implemented the current WebRTC draft, it is likely that it will be implemented in future releases. With the many peer-to-peer applications and solutions that exist to date, WebRTC could greatly empower the web applications of the future.

References

[1] H. Tsukayama. (2012, Jan.) Youtube: The future of entertainment is on the web. [Online]. Available: http://www.washingtonpost.com/business/technology/youtube-the-future-of-entertainment-is-on-the-web/2012/01/12/gIQADpdBuP_story.html

[2] Y.-W. Yen. (2008) Youtube looks for the money clip. [Online]. Available: http://tech.fortune.cnn.com/2008/03/25/youtube-looks-for-the-money-clip/

[3] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the ip multicast service and architecture," *Network, IEEE*, vol. 14, no. 1, pp. 78–88, 2000.

[4] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan. (2013) Webrtc 1.0: Real-time communication between browsers. [Online]. Available: http://dev.w3.org/2011/webrtc/editor/webrtc.html

[5] Y.-h. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1456–1471, 2002.

[6] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek *et al.*, "Overcast: reliable multicasting with on overlay network," in *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*. USENIX Association, 2000, pp. 14–14.

[7] E. Biersack, P. Rodriguez, and P. Felber, "Performance analysis of peer-to-peer networks for file distribution," *Quality of Service in the Emerging Networking Panorama*, pp. 1–10, 2004.

[8] X. Zhang, J. Liu, B. Li, and Y.-S. Yum, "Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3. IEEE, 2005, pp. 2102–2111.

[9] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the new coolstreaming: Principles, measurements and performance implications," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*. IEEE, 2008, pp. 1031–1039.

[10] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the power of pull-based streaming protocol: Can we do better?" *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 9, pp. 1678–1694, 2007.

[11] L. Zhao, J.-G. Luo, M. Zhang, W.-J. Fu, J. Luo, Y.-F. Zhang, and S.-Q. Yang, "Gridmedia: A practical peer-to-peer based live video streaming system," in *Multimedia Signal Processing, 2005 IEEE 7th Workshop on*. IEEE, 2005, pp. 1–4.

[12] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[13] SynaCast Media Tech Co.,Ltd. (2010) PPTV. [Online]. Available: http://download.pptv.com/en/about.html

[14] SopCast team. (2013) Sopcast - free p2p internet tv. [Online]. Available: http://www.sopcast.org

[15] F. Picconi and L. Massoulié, "Is there a future for mesh-based live video streaming?" in *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*. IEEE, 2008, pp. 289–298.

[16] C. Feng, B. Li, and B. Li, "Understanding the performance gap between pull-based mesh streaming protocols and fundamental limits," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 891–899.

[17] IETF. (2013) Rtcweb status pages. [Online]. Available: http://tools.ietf.org/wg/rtcweb/charters

[18] G. A. Eriksson and S. Håkansson, "Webrtc: enhancing the web with real-time communication capabilities," *On the brink of the Networked Society*, p. 4.

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

[20] J. K. Nurminen, A. J. Meyn, E. Jalonen, Y. Raivio, and R. G. Marrero, "P2P media streaming with HTML5 and WebRTC," in *IEEE International Conference on Computer Communications*. IEEE, 2013.

[21] M. Bu and E. Zhang. (2013) PeerJS - peer-to-peer data in the web browser. [Online]. Available: http://peerjs.com/

[22] A. Sehgal, O. Verscheure, and P. Frossard, "Distortion-buffer optimized tcp video streaming," in *Image Processing, 2004. ICIP'04. 2004 International Conference on*, vol. 3. IEEE, 2004, pp. 2083–2086.