Hyper-linked Communications: WebRTC enabled asynchronous collaboration

Michael Shell
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: http://www.michaelshell.org/contact.html

Homer Simpson
Twentieth Century Fox
Springfield, USA
Email: homer@thesimpsons.com

James Kirk
and Montgomery Scott
Starfleet Academy
San Francisco, California 96678-2391
Telephone: (800) 555–1212
Fax: (888) 555–1212

Abstract

The Hyper-linked communications concept applies much of the hypermedia concepts, widely used on Web content. This paradigm allows to synchronize, structure and navigate communication content integrated into voice and video calls.

Voice and image together can express emotions like no other medium can. With hypermedia concepts, we can add more value to conference calls.

WebRTC *technology allows real-time communications between web browsers without the need to install additional software. The nature of web browser applications already follows the hypermedia concept, which makes* WebRTC *the ideal technology to apply the hyper-linked communications concepts. The web browser platform provides an abstraction layer that makes it possible to create applications that run independently from the operating system. The native support for* WebRTC *in operating systems extends its usage to outside the web browser, allowing for the exploration of functionalities for which web browsers provide poor support, such as video recording and massive information storage.*

Our goal was the development of an application targeted to the web platform, resorting to WebRTC*, that leveraged the hyper-linked communications by providing a conference environment enriched with multiple media types, collaborative text editors, time annotations, instant messaging and a mechanism to superimpose hyper-content to video.*

In this document, we present the current State Of The Art in hyper-linked communications and related technologies, propose and implement an architecture for an hyper-linked communication application based on WebRTC*. This work was evaluated by users, who reported that they liked to use it and thought it to be extremely innovative.*

1. Introduction

1.1. Background

As communications technologies appeared, we adapted the way we communicate. The purpose of this project is not the replacement of the current video and audio communications, but to enrich them with hypermedia content and make them a more natural and easy to learn process.

With the advent of WebRTC and its successive integration with web browsers, it became possible to develop video conference web applications without plugins, this presents a range of possibilities on what can be implemented using already existing web technologies.

Furthermore, real-time communication applications can make a significant difference on business, education and health sectors by providing tools for devel-

oping teaching and learning online, teamworking and socializing web applications.

1.2. Proposed Solution

Our goal in this project is to develop an application targeted to the web platform, resorting to Web Real-Time Communication (WebRTC), that leverages the hyper-linked communications by providing a video conference environment enriched with interactive and non-interactive discrete media types such as images, subtitles, forms and all types of content that can be added using HyperText Markup Language (HTML)5, Cascading Style Sheets (CSS)3 and *JavaScript* including continuous media types such as video, music and animations.

One of the key features of this project is the ability to navigate in time in order to reproduce the conversation again or introduce hyper-content to it such as time annotations, interactive lists of topics and subtitles. In this context we also provide a simpler method for creating and synchronizing hyper-content using *QR codes*.

In addition to this conference environment, which provides different functionalities than traditional conference environments such as *Skype* and *Google Hangouts*, we also enable a collaborative text editor and a chat that supports sending time hyper-links and files to conference participants.

Furthermore, another relevant feature is the possibility to compose multiple video streams into a single one, which enables adding more users to conference rooms without impacting on clients performance. Users can change to individual streams on demand or automatically to the talking users.

1.3. Thesis Contribution

Making it clear, this project aims to complement current audio, text and video communications in order to create rich and collaborative interfaces with the ability to add more content on a future time (e.g. creating time annotations for improving content search) in order to increase its value. It is also important to highlight another goal of this project which is the ability to navigate in time by rewinding communications, fast-forward and jump to certain points.

We have presented an architecture that can meet our goals, implemented the respective prototype and tested it with real users and performance benchmarks.

According to Martin Geddes, the quality of the interaction worsens as the number of users increase [**?**]. In our testing phases we will quantify and qualify the impact of increasing users on the interface and performance of our prototype.

All the problems faced during the development and limitations were reported on the thesis so that a future project better then ours can be easily and better developed.

1.4. Outline

This rest of this document is structured as follows:

- **Chapter ??** describes the previous work in the field.
- **Chapter ??** describes the system requirements and the architecture for an Web Application that fulfills the goals of this thesis.
- **Chapter ??** describes the implementation of our Web Application and the technologies chosen.
- **Chapter ??** presents the evaluation tests performed and the corresponding results.
- **Chapter ??** summarizes the work developed and proposes future work.

2. Related Work

This section is structured as follows. Section **??** describes the problems that real-time communications face on nowadays internet, namely the Internet Protocol Version 4 (IPv4) address exhaustion and the client server model constraints. Section **??** describes the WebRTC technology and the protocols needed to implement our project. Section **??** addresses the signaling component of chat applications, which is not defined on WebRTC specifications. Section **??** presents the evolution of multimedia content until the hypermedia, its capabilities, synchronization mechanisms and interactivity. Section **??** explores streaming protocols for non-interactive multimedia and how to introduce the interactive component, another important aspects are the ability to control the time flux of a stream and collaborative application development.

2.1. Early days of the Internet and its remaining flaws

The need to build a global communications network in an age when almost nobody had access to that technology and the unpredictability of the number of future users, lead to some protocols not being suitable for the explosive growth and proliferation of users that followed. IPv4 limits the number of public addresses in such a way that today they are scarce [**?**].

To this end, one way to overcome the IPv4 address scarcity problem was the development of a mechanism

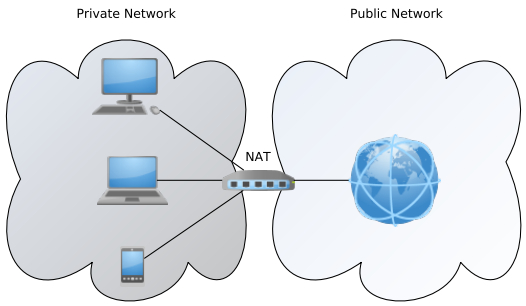

Figure 1. Network Address Translation

that groups multiple address into a single one, the machine that is assigned that address is then responsible for redirecting messages to members of its group using their private addresses, each connection in the private network is identified publicly by the same Internet Protocol (IP) address with a different port.

This technique is known as Network Address Translation (NAT) (figure **??**).

Initially NAT offered an alternative to address exhaustion and a minimal sensation of security. However, due to their current wide usage, NATs weaknesses are being exposing at the application layer, namely impacting applications that require direct communications between two private networks.

There are four types of NAT implementations [**?**]: *Full Cone NAT*, *Restricted Cone NAT*, *Port Restricted Cone NAT*, *Symmetric NAT*.

Full Cone NAT maps each public IP address and port to a private IP address and port. Any external host can communicate with private hosts through their mapped public address and port. This represents the least restrictive type of NAT and, as we will see later, the unique type of NAT that enables real time communications from point to point.

Restricted Cone NAT requires that a private client must first send a message to an external host before it can receive messages from the same host. With this type of NAT, the private client can be contacted from any port of the same external host.

Port Restricted Cone NAT works in the same way as Restricted Cone NAT, but it only allows communications from the same external host's IP address and port, ignoring all messages from other applications within the same external host.

Symmetric NAT maps different ports for each connection. As we will see later, this type of NAT represents a problem on real time communications.

Non-Symmetric NATs became the common configurations on the Internet. As a direct result, problems started to appear: the amount of ports that IP makes available is also small compared to our current needs; worse than that, NAT also difficults end-to-end communication, forcing most applications that follow this model to be implemented ineffectively.

Unless the router that performs as NAT has forwarding rules to every desired ports of each user, applications behind a NAT are prevented from receiving incoming connections from the public network, which forces them to behave as a client of a client-server model.

Session Traversal Utilities for NAT (STUN) and Traversal Using Relays around NAT (TURN) [**?**] servers are a possible solution to overcome NAT.

STUN servers are quite simple. They receive requests from NATed clients, with the source address of a request being the public address that NAT mapped to the client. STUN servers will then reply to the client, providing the mapped public address, so it knows its associated public IP address and port. Symmetric NAT changes IP port for each different connection, for that reason, when the STUN servers reply with the IP address and port of their connection, it will be useless for clients to use them on other application connections. That is why Symmetric NAT represents a problem for peer-to-peer communications.

On the other hand, TURN uses public servers to relay traffic between private endpoints. It may use a Peer-to-peer (P2P) network relay to find the best peer, but after that, the behavior is much like client-server. Direct communication is only achieved by STUN when NAT is a type *full cone*. Interactive Connectivity Establishment (ICE) is a technique that uses STUN when direct communications are possible and TURN while a direct communication isn't possible.

When connection is established, either in a direct or indirect way (via TURN servers), WebRTC came to simplify how audio and video are transmitted through web browsers.

2.2. Real time communications

WebRTC is an open source technology that defines a collection of standard protocols and JavaScript Application Programming Interface (API)s for web browser based real time communications without installing any additional application or plug-in. Table **??** shows the protocols that WebRTC rely on.

Some operating systems such as *Android*, *iOS*, *Linux*, *OSX* and *Windows* implement native WebRTC libraries, extending the usage of WebRTC to applications outside the web browser. This native support can

Table 1. WebRTC protocol Stack

<table>
<tr><td colspan="3"></td><td>MediaStream</td><td>DataChannel</td></tr>
<tr><td>XHR</td><td>SSE</td><td>WebSocket</td><td>SRTP</td><td>SCTP</td></tr>
<tr><td colspan="3">HTTP 1.X/2.0</td><td colspan="2">Session (DTLS)</td></tr>
<tr><td colspan="3">Session (TLS) - optional</td><td colspan="2">ICE, STUN, TURN</td></tr>
<tr><td colspan="3">Transport (TCP)</td><td colspan="2">Transport (UDP)</td></tr>
<tr><td colspan="5">Network (IP)</td></tr>
</table>

help to implement applications that record video and audio streams for further playback.

WebRTC defines three main APIs: MediaStream, PeerConnection and DataChannel.

- **MediaStream** allows the browser to access the camera, microphone and the device's screen.
- **PeerConnection** acquires connection data and negotiates with peers.
- **DataChannel** provides a channel for exchanging arbitrary data with other peers.

WebRTC uses User Datagram Protocol (UDP) for transporting data, which provides lower latencies than Transmission Control Protocol (TCP), but is not reliable and does not assure packet order and integrity. Stream Control Transmission Protocol (SCTP) and Secure Real-time Transport Protocol (SRTP) are used for streaming data, providing a mechanism for congestion control and partial reliable delivery over UDP. All transferred audio, data and video must be encrypted with Datagram Transport Layer Security (DTLS) symmetric keys. DTLS provides the same security guarantees as Transport Layer Security (TLS).

TLS doesn't support independent packet decryption [**?**], for that it requires a reliable transport channel, typically TCP. The decryption of a packet depends on the previous packet, which for unreliable transport protocols like UDP may represent a problem, either due to packet loss or different reception order.

DTLS is similar to TLS, but is used on top of UDP.

WebRTC's *MediaStream* is built on top of SRTP, which requires an external mechanism for key exchange. DTLS keys are negotiated on handshake in order to achieve a secure connection. The new keys derived from DTLS handshake are seized for SRTP encryption, the remaining SRTP communications are done through UDP without using DTLS.

WebRTC aims to provide a standard platform for real-time audio and video on the Web. It arrives at a time when several proprietary products are well established.

2.3. Signaling: meet and get to know

Signaling is the process by which applications exchange connection information about peers and servers, their capabilities and meta-data. In particular, WebRTC does not implement signaling, as different applications may require different protocols and there is no single answer that fits all problems. As a consequence, multiple options are available for filling the missing WebRTC's signaling component, which can be performed using Session Initiation Protocol (SIP), Extensible Messaging and Presence Protocol (XMPP), *WebSockets*, *Socket.io*[1] or by implementing a custom protocol.

2.3.1. WebRTC. WebRTC uses Session Description Protocol (SDP) [**?**] to define peer connection properties such as types of supported media, codecs, protocols used and network information. An SDP offer describes to other peers the expected type of communication and its details, such as used transport protocols, codecs, security and other.

One of WebRTC signaling's requisites is bi-directional communication.

Hypertext Transfer Protocol (HTTP) uses a request-response paradigm, where a request is sent by the client, followed by a server response.

Sometimes it is required that some information be obtained in real time, but we saw, some NAT's do not support callbacks from servers, preventing them from notifying clients as soon as an event occures. One technique to overcome this problem is long polling which consists on making the server hold the request until there is fresh information or expiring it after some time. As soon as it receives the reply, the client makes another request. Long polling technique results on a better network usage and a faster server response, but both simple polling and long polling requests are sent with HTTP headers, which add data overhead and can be noticed especially for short sized messages.

The *WebSocket* protocol allows bi-directional communications over a full-duplex socket channel [**?**], by other words it supports sending and receiving data simultaneously.

2.3.2. Session Initiation Protocol. SIP [**?**] is a protocol used for negotiation, creation, modification and finalization of communication sessions between users.

1. http://socket.io/(accessed June 1, 2015).

SIP follows a client/server architecture with HTTP like messages and it can be used as a signaling protocol. The advantage of SIP is the ability to make video and voice calls between the telephone network and applications over IP networks.

SIP is used in Voice Over IP (VoIP) applications due to its compatibility with the Public Switched Telephone Network (PSTN).

Frameworks like *jsSIP*[2], *QoffeeSIP*[3] and *sipML5*[4] are used on the client side to parse and encode SIP messages, making SIP accessible to web based applications.

SIP with *WebSockets* can be used as a signaling method for WebRTC applications, it allows web browsers to have audio, video and Short Message Service (SMS) capabilities like mobile phones. For instance, it's possible to inter-operate web communications with SIP networks, mobile and fixed phones.

2.3.3. Extensible Messaging and Presence Protocol. XMPP was initially developed for instant messaging and presence (Jabber[5]). It is nowadays an open technology for standardized, decentralized, secure and extensible real-time communications.

Today, multiple XMPP server implementations exists, such as: *ejabberd*[6], *Metronome*[7], *Openfire*[8] and *Prosody*[9]. *Ejabberd* is the server that implements more Request For Comments (RFC) specifications and XMPP Extensions (XEP)s[10].

The connection to the XMPP server can be done through the same web server as the user is connected, but this web server would handle a lot of connections and that would have a great impact on the overall system performance. On the other hand, users can directly connect directly to XMPP servers from web applications using the *JavaScript* library *strophe.js* [11]

Typically a web application consists in multiple web pages which are navigated by users. Each time a web page is accessed, either from a new context or a transition from a previous page, its context is cleared except for local storage and cookies. The *JavaScript* context is cleared, including the XMPP connections performed by *strophe.js*. As such, an automatic mechanism is required for avoiding the implicit user reauthentication. One solution for reauthentication can be achieved by storing the user's JabberID and password on local storage and every time a page is accessed the authentication is performed without the users knowledge. Clearly this solution represents security flaws as the local storage can be easily accessed locally or by performing a cross site scripting attack to reveal all the local storage. The same problem arises if the credentials are stored on cookies.

An alternative solution is known as Session Attachment [12], which requires that a *session identifier*(SID) together with a *initial request identifier (RID)* be passed to *strophe.js* in order to re-connect to the same stream on XMPP server. Either SID and RID are unpredictable and, particularly, *RID* changes on every request making it worthless if a user maintains more than one tab opened, for example for multiple conversations at the same time.

2.4. Hypermedia: more than words, more than images

Since the early days of video technology, one of the problems raised consisted on how to add more information onto video without generating multiple versions. This section examines technologies that allows different ways to present multimedia content in such a way that it change based on synchronization amongst other multimedia elements or user interaction.

Hypermedia concept brings the possibility to organize and overlay multimedia elements into a nonlinear linear structure holding the promise of future technology and features.

Hyper-video is a kind of video that contains links to any kind of hypermedia, including links to skip part of it. An example of hypermedia application could be a search engine over hypermedia content, like subtitles, in order to jump to a specific time in a video or audio track. *HyperCafe* [**?**] was an experimental project to expose hyper-video concepts that consisted of an interactive film that enabled switching between different conversations taking place inside a cafe.

In order to navigate through a dynamic video, we must be aware of time synchronization and the multiple time flows, it ils important that all time, causality and behavior rules are well defined.

HyVAL [**?**] is an eXtensible Markup Language (XML) based language that was proposed for modeling composition, synchronization and interaction of

2. http://jssip.net/(accessed June 1, 2015).
3. http://qoffeesip.quobis.com/(accessed June 1, 2015).
4. http://sipml5.org/(accessed June 1, 2015).
5. http://jabber.org/(accessed June 1, 2015).
6. http://jabberd.im/(accessed June 1, 2015).
7. http://lightwitch.org/metronome(accessed June 1, 2015).
8. http://igniterealtime.org/projects/openfire/(accessed June 1, 2015).
9. http://prosody.im/(accessed June 1, 2015).
10. http://en.wikipedia.org/wiki/Comparison_of_XMPP_server_software(accessed June 1, 2015).
11. http://strophe.im/strophejs/ (accessed May 6, 2016).
12. https://metajack.im/2008/10/03/getting-attached-to-strophe/ (accessed May 6, 2016).

hypermedia. HyVAL defines defines video structure, internal video and external media objects.

Synchronized Multimedia Integration Language (SMIL) [**?**] was introduced to describe temporal behavior of multimedia content, in particular, it could be used to overlay subtitles on films. With SMIL it is possible to synchronize multiple videos, either in parallel or in sequence, reproduce a different audio track, overlay user interface elements with hyper-links, among multiple other features.

The Document Object Model (DOM) is a standard API that allows easy management of documents that are organized in a tree structure, by providing Create, Read, Update and Delete (CRUD) operations over its elements and their attributes. DOM makes it easy to inter-operate between imperative and declarative programming languages [**?**].

Like DOM, SMIL DOM is an API for SMIL documents. Allowing CRUD operations over SMIL documents is an important feature for extending SMIL capabilities, for example for creating non-linear animations and triggering external events like *JavaScript* functions.

SMIL fits our goals for creating a multimedia rich hyper-call, but it lacks on browser compatibility. Ambulant [**?**] was one of the SMIL players that were developed for browsers, although this player implements most of SMIL 3.0 [**?**] specifications, it needs to be installed on browsers as a plug-in.

JavaScript is an imperative object-oriented language based on *ECMAScript*. It is used mainly on client-side and executed by a web browser. *JavaScript* has its own implementation of DOM and one of its advantages is the ability to download and execute code on the fly without the need of pre-installed plug-ins.

JavaScript has compatibility issues among the different web browsers, leading to different behaviors. To solve that problem, there are libraries written in *JavasSript*, namely *jQuery*, that implements the same functionality for multiple browsers, masking most of the incompatibility issues.

Scalable Vector Graphics (SVG) is an XML based format that incorporates the animation module of SMIL. Currently, SVG allows adding movement and animating attributes of elements. When embedded on HTML5, it allows dynamic changes to inner content in real-time through the DOM API. Besides that, it also allows calling *JavaScript* functions on events such as animation end, mouse over and mouse click.

Using technologies that relies only on web standards, like CSS, HTML5, *JavaScript* and SVG, will make possible to develop an application that solves our problem with the advantage of being compatible with a greater amount of web browsers.

2.5. Extending collaboration tools with time manipulation

Real time collaboration applications have become a huge help on team tasks, providing a great boost on business, research and investigation velocity. Technologies like these are appearing along these days, but they were not be possible a few years ago because technology was limited or unavailable. Although today's technology is still limited on some aspects, progress is being done in order to improve the web ecosystem, by creating standards and migrating to newer technologies.

Section **??** presents the RTP protocol how it can be used to record media streams. Section **??** describes the media types and what types of media should be streamed. Section **??** addresses how an interactive media can be recorded. Section **??** presents the collaborative environment and libraries to synchronize distributed object among users.

2.5.1. Streaming and Recording.

Our first concern on real time collaboration applications, besides the communication itself, is the data storage and representation. Storing multimedia content is not a viable solution because most browsers recommend limiting local storage to at most five megabytes per origin.

In order to provide a way to record and playback streams, additional servers will be required to process and record the large amount of data generated by audio and video streaming.

Real-time Transport Protocol (RTP) [**?**] is used for streaming audio and video over IP. Multimedia content is transported on the payload of RTP messages, that contain headers for payload identification. RTP is independent from its payload type, allowing it to transport any kind of encoded multimedia. A sequence number is used for sorting received packets.

RTP allows to change its requirements and add extensions to it with profiles. One of the most used ones is the RTP profile for audio and video [**?**], which lists the payload encoding and compression algorithms. Another profile for RTP is defined by SRTP, which provides encryption, authentication and replay protection for RTP traffic. The analogous secure protocol for Real Time Control Protocol (RTCP) is Secure RTCP (SRTCP).

RTP recorders are independent of payload encoding, they don't decode RTP packets, they record packets

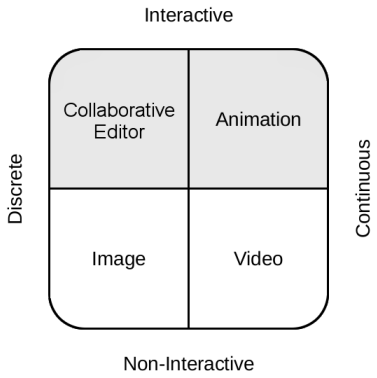

Figure 2. Media Types

instead, allowing to record all video and audio formats even if they're encrypted.

Kurento Media Server (KMS) supports streaming over *WebRTC*, HTTP and RTP endpoints. Another important component of KMS is *Kurento Repository*, which supports recording and playing directly from *MongoDB*. That is important for providing a scalable media storage. Unlike *Jitsi Video Bridge*[13], KMS does not enforce a specific signaling protocol.

2.5.2. Media Types.

Media Types can be distinguished by two criteria, the first one describes a media as discrete or continuous, the second one describes it as interactive or non-interactive. A discrete media is characterized by not depending from time, and continuous media as depending on it. Interactive media is characterized by its state being changed by external events such as user interactions.

For example, an image is non-interactive and discrete, while a video is continuous and non-interactive. A simple collaborative editor with just text is interactive and discrete. An animation that changes in function of user behavior is interactive and continuous.

2.5.3. Collaborative Environment.

Operational transformation (OT) technology was originally developed for consistency maintenance and concurrency control over distributed objects. OT algorithms are mainly used in collaborative applications such as distributed document edition.

Among mutiple OT platforms and libraries we present *ShareJS*[14], *TogetherJS*[15], *Goodow*[16], *Etherpad Lite*[17] and *otJS*[18] which we describe on Table **??**.

otJS is a *JavaScript* library that only implements operation transformations over plain text on the client side. An implication of implementing just the client side is the extra effort that is necessary to implement content's persistent storage. Besides this drawback, this library is very flexible because it's not tied to a specific database or server side technology.

Table 2. Comparision between Operational Transformation libraries

Library	Own Server	Own Storage	Operations
ShareJS	yes	yes	text+objects
TogetherJS	yes	no	text+objects
Goodow	yes	yes	text+objects
Etherpad Lite	yes	yes	extendable
OT.js	no	no	text

2.6. Chapter Summary

In order to implement an hyper-linked communication solution, several design decisions had to be made. The limitations imposed by the *Internet's* structure, its protocols and a browser's capabilities are key factors to consider when implementing a solution that allows bi-directional communications, interactive media and collaboration environment.

Due to the use of NAT, bi-directional communications between clients have different needs from request-response based communications between clients and servers. This lead to the appearance of mechanisms such as STUN, TURN and ICE, in order to bypass the limits imposed by NAT.

WebRTC introduced an API for video, audio and data communications through web browsers without using plug-ins. However, WebRTC by itself does not define how users get to know each other nor how information flows between users. For this reason, we have studied the multiple ways we could implement this *get-to-know* mechanism which is known as signaling protocol.

With the communications establishment issue solved, we had to discuss the different types of media and what can be done with each kind in order to

13. http://jitsi.org/Projects/JitsiVideobridge(accessed June 2, 2015).

14. http://sharejs.org/(accessed June 2, 2015).

15. http://togetherjs.com/(accessed June 2, 2015).

16. http://realtimeplayground.goodow.com/(accessed June 2, 2015).

17. https://github.com/ether/etherpad-lite(Accessed 20 March 2016)

18. http://operational-transformation.github.io(accessed March 10, 2016)

increase the value of communications among users. In this context, we have studied solutions and libraries that allow us to implement our prototype with time manipulation features, collaborative text edition, record and playback interactive video.

3. Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thank...

References

[1] Martin, G.: Hypertext to Hypervoice - Linking what we say and what we do. (2012) 6

[2] Paper, A.I.U.W.: Next Generation Internet : IPv4 Address Exhaustion , Mitigation Strategies and Implications for the U. S. (2009) 1–26

[3] Rosenberg, J., Weinberger, J., Huitema, C., Mahy, R.: STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard) (March 2003) Obsoleted by RFC 5389.

[4] Lin, Y.D., Tseng, C.C., Ho, C.Y., Wu, Y.H.: How nat-compatible are voip applications? Communications Magazine, IEEE **48**(12) (December 2010) 58–65

[5] Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard) (January 2012) Updated by RFC 7507.

[6] Handley, M., Jacobson, V., Perkins, C.: SDP: Session Description Protocol. RFC 4566 (Proposed Standard) (July 2006)

[7] Fette, I., Melnikov, A.: The WebSocket Protocol. RFC 6455 (Proposed Standard) (December 2011)

[8] Paterson, I., Smith, D., Saint-Andre, P., Moffitt, J., Stout, L., Tilanus, W.: Bidirectional-streams Over Synchronous HTTP (BOSH). XEP-0124 (Draft) (April 2014)

[9] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard) (June 2002) Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141.

[10] Lonnfors, M., Costa-Requena, J., Leppanen, E., Khartabil, H.: Session Initiation Protocol (SIP) Extension for Partial Notification of Presence Information. RFC 5263 (Proposed Standard) (September 2008)

[11] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., Gurle, D.: Session Initiation Protocol (SIP) Extension for Instant Messaging. RFC 3428 (Proposed Standard) (December 2002)

[12] Muldowney, T., Miller, M., Eatmon, R., Saint-Andre, P.: SI File Transfer. XEP-0096 (Draft) (April 2004)

[13] Saint-Andre, P.: Multi-User Chat. XEP-0045 (Draft) (February 2012)

[14] Paterson, I., Saint-Andre, P., Stout, L., Tilanus, W.: XMPP Over BOSH. XEP-0206 (Draft) (April 2014)

[15] Stout, L., Moffitt, J., Cestari, E.: An Extensible Messaging and Presence Protocol (XMPP) Subprotocol for WebSocket. IETF RFC 7395 (October 2015)

[16] Eatmon, R., Hildebrand, J., Miller, J., Muldowney, T., Saint-Andre, P.: XEP-0004: Data Forms. (2007)

[17] Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard) (October 2004) Obsoleted by RFC 6120, updated by RFC 6122.

[18] Chainho, P., Haensge, K., Druesedow, S., Maruschke, M.: Signalling-on-the-fly: Sigofly. In: Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on. (Feb 2015) 1–8

[19] Sampaio Gradvohl, A.L., Iano, Y.: Matching interactive tv and hypervideo. Latin America Transactions, IEEE (Revista IEEE America Latina) **5**(8) (Dec 2007) 579–584

[20] Sawhney, N., Balcom, D., Smith, I.: Authoring and navigating video in space and time. MultiMedia, IEEE **4**(4) (Oct 1997) 30–39

[21] Shipman, F., Girgensohn, A., Wilcox, L.: Creating navigable multi-level video summaries. In: Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on. Volume 2. (July 2003) II–753–6 vol.2

[22] Zhou, T., Jin, J.: A structured document model for authoring video-based hypermedia. In: Multimedia Modelling Conference, 2005. MMM 2005. Proceedings of the 11th International. (Jan 2005) 421–426

[23] Bulterman, D.: Smil 2.0 part 1: overview, concepts, and structure. MultiMedia, IEEE **8**(4) (Oct 2001) 82–88

[24] Robie, J., Nicol, G., Wood, L., Wilson, C., Champion, M., Isaacson, S., Byrne, S.B., Jacobs, I., Sutor, R.S., Hors, A.L.: Document object model (DOM) level 1. W3C recommendation, W3C (October 1998) http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001.

[25] Bulterman, D.C.A., Jansen, A.J., Cesar Garcia, P.S.: Video On The Web: Experiences From SMIL And From The Ambulant Annotator. In Le Hégaret, P., ed.: Collected Position Papers, W3C Video on the Web Workshop, W3C (December 2007) –

[26] Bulterman, D., Jansen, J., Cesar, P., Mullender, S., DeMeglio, M., Quint, J., Vuorimaa, P., Cruz-Lara, S., Kawamura, H., Weck, D., Hyche, E., Pañeda, X.G., Melendi, D., Michel, T., Zucker, D.F.: Synchronized multimedia integration language (smil 3.0). World Wide Web Consortium, Recommendation REC-SMIL3-20081201 (December 2008)

[27] Gaggi, O., Danese, L.: A smil player for any web browser. In: DMS, Knowledge Systems Institute (2011) 114–119

[28] McAlarney, J., Haddad, R., McGarry, M.P.: Modeling network protocol overhead for video. In: Computer Modeling and Simulation (EMS), 2010 Fourth UKSim European Symposium on. (Nov 2010) 375–380

[29] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard) (July 2003) Updated by RFCs 5506, 5761, 6051, 6222.

[30] Schulzrinne, H., Casner, S.: RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551 (Standard) (July 2003) Updated by RFC 5761.

[31] Ivov, E., Marinov, L., Hancke, P.: COnferences with LIghtweight BRIdging (COLIBRI). XEP-0340 (Experimental) (January 2014)

[32] Mauve, M., Hilt, V., Kuhmunch, C., Effelsberg, W.: A general framework and communication protocol for the transmission of interactive media with real-time characteristics. In: Multimedia Computing and Systems, 1999. IEEE International Conference on. Volume 2. (Jul 1999) 641–646 vol.2

[33] Hilt, V., Mauve, M., Kuhmünch, C., Effelsberg, W.: A generic scheme for the recording of interactive media streams. In Diaz, M., Owezarski, P., Sénac, P., eds.: Interactive Distributed Multimedia Systems and Telecommunication Services. Volume 1718 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1999) 291–304

[34] Nichols, D.A., Curtis, P., Dixon, M., Lamping, J.: High-latency, low-bandwidth windowing in the jupiter collaboration system. In: Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology. UIST '95, New York, NY, USA, ACM (1995) 111–120

[35] Morris, R., Thompson, K.: Password security: A case history. Communications of the ACM **22**(11) (1979) 594–597

[36] Schütt, K., Kloft, M., Bikadorov, A., Rieck, K.: Early detection of malicious behavior in javascript code. In: Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence. AISec '12, New York, NY, USA, ACM (2012) 15–24