Hyper-linked Communications: WebRTC enabled asynchronous collaboration

Henrique Duarte Lopes Rocha
email: henrique.rocha@tecnico.ulisboa.pt
student number: 68621

Master Degree in Telecommunications and Informatics Engineering
Instituto Superior Técnico
Universidade de Lisboa

Abstract. The Hyper-linked communications concept applies much of the hypermedia concepts, widely used on Web content, to communication and collaboration services. This paradigm allows to synchronize, structure and navigate communication content, providing a way to integrate social media content and collaborative tools into voice and video calls. Voice and image together can express emotions and expose creativity like no other medium can. With hypermedia concepts, we can add more value to voice and video communications.
Web Real-Time Communication (WebRTC) technology allows real-time communications between web browsers without the need to install and use additional applications or plug-ins. The nature of web browser applications already follow 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 possible to create applications that run indenpendently from the operating system. The native support of WebRTC in operating systems extends its usage to outside the web browser, allowing to explore functionalities that web browsers are poor to support such as video recording and massive information storage.
Our goal in this project is to develop an application that leverages the Hyper-linked communications concept to enable collaborative real-time audio and video communication, enriched with other media types, that can be accessed live or accessed later. This application will target the web browser platform, resorting to WebRTC.
In this document, we present the current State Of The Art in hyper-linked communications and related technologies, propose an architecture to implement an hyper-linked communication application based on WebRTC and provide a plan for its implementation and evaluation.

Keywords: WebRTC, asynchronous, communications, collaboration

1 Introduction

1.1 Context

Since the early days of Human History, we tried to communicate over far locations, from smoke signals to letters delivered by messengers. Real-time communications were

limited or even nonexistent. Despite all the efforts made to improve communications, written communication could never replace face to face communication. With the advent of the telephone network, communications have taken a very important step for us to feel more connected with whom we communicate. Still, only the human voice was not enough, and the invention of cameras and consequent video digitization were a huge step for real-time communications.

In the past, handwritten documents were limited to a writer per page at a time. Writing a book collaboratively was a difficult task due to synchronism between writers. Today, we can achieve more, it is possible to write a document collaboratively, correct spelling mistakes without wasting paper, restructure text at any moment, add a video to a newspaper article and more. Although much of what was said seems banal nowadays, none of this was possible before the computer's invention.

As Martin Geddes states[14], "No computer in our lifetimes will ever rival a human voice's capacity to conveying rich and complex social and emotional meaning" , although nothing replaces the physical contact with a person while we communicate, we are at a time when we can do more than just a visual and verbal communication, hypermedia can be added to video and voice in order to extend its value. The concept of structured voice and video synchronized with hypermedia is called hypervoice[14].

1.2 Problem Statement

As communications technologies appeared, we adapted the way we communicate. This project doesn't aims to replace the current video and audio communications, but to enrich them with hyper-media content and make them a more natural and easy to learn process.

With the advent of WebRTC, it became possible to develop video conference web apllications without plugins, this presents a range of possibilities on what can be implemented using already existing web technologies.

Real-time communication applications can make a difference on business, education and health sectors by providing tools for teaching and learning online, teamworking and socializing.

For multiple reasons, we often need to repeat or postpone some of our tasks, some people tend to forget what they ear or see. A real-time system is a huge source of information that requires much attention from its users. An application that provides a way to remember our past communications would be a strong tool for not only to catch what we lost but also to enhance our knowledge.

1.3 Thesis Goals

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 annotations for improving content search) in order to increase its value. Another important goal of this project is the ability to navigate in time by rewinding communications, fast-forward and jump to certain points.

A web application with an easy to learn user interface will be developed to accomplish solving our problem. Our application, unnamed yet, is targeted at web browsers

that are compatible with only standard technologies like JavaScript, WebRTC, HyperText Markup Language (HTML)5 and Cascading Style Sheets (CSS)3. Any additional plug-in is avoidable, *JavaScript* libraries will be preferred as they can be downloaded on the fly.

Although we propose an initial solution to solve our problem, we will make a continuous effort to survey the systems that solve our problem either completely or just a part of it, not excluding solutions that may appear during the implementation of our project.

We will present an architecture that can meet our goals, implement the respective prototype and test it with real users, unitary tests and benchmarks.

According to Martin Geddes, the quality of the interaction worsens as the number of users increase[14]. 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 will be reported on the thesis so that a future project better then ours can be easily and better developed.

1.4 Document Structure

This document is structured as follows. Section 2 presents the related work. In Section 4, we propose an architecture for an Web Application that fulfills the goals of this thesis, including all the need infrastructure and software. Section 5 describes our work methodology and our plan for implementing and validating our proposed architecture. Section 6 presents the summary and conclusions of our work.

2 Related Work

This section is structured as follows. Section 2.1 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 2.2 describes the WebRTC technology and the protocols needed to implement our project. Section 2.3 addresses the signaling component of chat applications, which is not defined on WebRTC specifications. Section 2.4 presents the evolution of multimedia content until the hypermedia, its capabilities, synchronization mechanisms and interactivity. Section 2.5 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 number of future users was unpredictable, lead to some protocols not being suitable for the huge growth of users that followed. IPv4 limits the number of public addresses in such a way that today they are scarce [20]. One way to overcome this problem was the development of a mechanism 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).

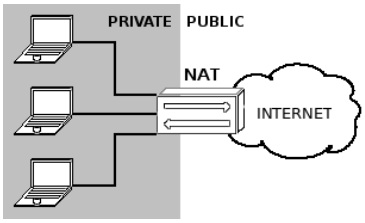


Fig. 1: Network Address Translation

Initially NAT offered an alternative to address exhaustion and a minimal sensation of security, although its current wide usage, NATs are exposing their weaknesses to the application layer, namely on applications that require direct communications between two private networks. There are four types of NAT implementations[26]: *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.

Applications based on multimedia and peer-to-peer file sharing have been one of the most strained by NAT. Those kind of applications require real time communication in order to achieve the best performance.

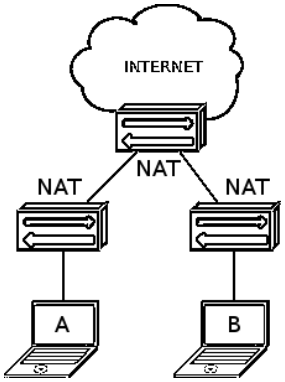


Fig. 2: Multiple level NATs

Session Traversal Utilities for NAT (STUN) and Traversal Using Relays around NAT (TURN) [12] servers are a possible solution to overcome NAT. None of those can establish direct connections on multiple level NATs.

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 in other connection connections. That is why Symmetric NAT represents a problem for peer-to-peer communications.

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.

Most of client-server applications aren't affected by NAT when the servers are public, but they're inadequate for real time communication between two private endpoints. Clearly TURN requires a more expensive relaying infrastructure and, in most cases, more network usage, leading to a worse quality of service. The requirements of real-time video communication makes this kind of model unsuitable.

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.

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

<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>

Table 1: WebRTC protocol Stack

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[23], 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. The main difference is the inclusion of a sequence number per packet that is used for packet re-ordering on reception and protects from duplicated packets. If a packet sequence number is less than the expected sequence number the packet is discarded. If a packet sequence number is greater than the expected sequence number the packet may be enqueued or discarded. By knowing the sequence of messages that are sent and received in DTLS, timers are used for packet retransmission avoiding acknowledgment messages.

	TCP	**UDP**	**SCTP**
Reliability	reliable	unreliable	configurable
Delivery	ordered	unordered	configurable
Transmission	byte-oriented	message-oriented	message-oriented
Flow control	yes	no	yes
Congestion control	yes	no	yes

Table 2: Overview of transport protocols

WebRTC's *DataChannel* is built on top of SCTP, which is encapsulated by DTLS. DTLS encapsulation provides confidentiality, authentication and integrity to the transfered data. A *Data Channel* has one incoming stream and one outgoing stream, providing bidirectional communication. Each data channel direction can be configured for reliable or unreliable transmission, the same can be done for order delivery and priority. which can also be defined for improving the quality of service of a particular stream over the others.

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. *Skype*[1] is an application that allows video, voice, instant messaging and multi-party communication over proprietary protocols, its main strength are the amount of users that are using it nowadays and the ability to perform voice calls to the Public Switched Telephone Network (PSTN). But compared to *Skype*, WebRTC applications don't need to be pre-installed.

[1] `http://www.skype.com/`(accessed June 1, 2015).

Google Hangouts[2] is another popular video multi-party conference web application. In the past, in order to use *hangouts* on a web browser a plug-in had to be installed, nowadays hangouts is using WebRTC. *Google Hangouts* supports viewing videos on *youtube* synchronously, drawing collaboratively, creating music and playing multi-player games. These applications are implemented with *Adobe Flash*.

Jitsi Meet[3] is a WebRTC collaborative application that uses *Jitsi Videobridge* for high quality and scalable video conferences and supports shared document editing. *Jitsi Meet* allows a great amount of users in the same conversation by identifying the current most active participant users and, by consequence, reducing the video and audio quality for all the other users. *Jitsi Videobridge* is a server that enables multi-party video calls.

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. WebRTC doesn't implement signaling, as different applications may require different protocols, and there is no single answer that fits all problems. Multiple options are available for signaling, which can be performed using Session Initiation Protocol (SIP), Extensible Messaging and Presence Protocol (XMPP), *WebSockets*, *Socket.io* or by implementing a custom protocol.

WebRTC uses Session Description Protocol (SDP) [9] 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 over Hypertext Transfer Protocol (HTTP). 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 polling.

Polling consists on sending periodic messages to which the server responds immediately with empty content or fresh information. Text and presence messages are unpredictable, if the time between periodic requests is short, most of the time the server will return empty results wasting network bandwidth and energy. On the other hand, if the time between periodic requests is large, newer messages may arrive too late.

A technique called long polling 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, especially for short messages.

The WebSocket protocol allows bidirectional communications over a full-duplex socket channel [7]. WebSocket handshake phase specifies a HTTP header in order to upgrade to *WebSocket* type of communication, but the remainder messages are exchanged without HTTP headers, which leads to much smaller messages and better network usage. WebSockets may not be available on every web browser, frameworks like

[2] `http://plus.google.com/hangouts`(accessed June 1, 2015).
[3] `http://jitsi.org/Projects/JitsiMeet`(accessed June 1, 2015).

socket.io[4] and *SockJS*[5] fall back to using HTTP when there is no support for WebSockets.

Bidirectional-streams Over Synchronous HTTP (BOSH)[22] is a technique based on long polling that uses two socket connections and allows sending client messages to the server while a previous request is held. The BOSH specification assumes that a connection manager is implemented to handle HTTP connections. This connection manager is basically a translator from HTTP to raw message so that the server may be implemented as if this communication is performed over TCP. When the connection manager holds for a response for too long, it responds with an empty body, this technique prevents an HTTP session from expiring when the client is waiting for a response, thus expanding the session time. Expiring sessions can be expensive due to the overhead of establishing new connections, which is even worse when HTTP is used over Secure Sockets Layer (SSL).

If the server is holding a request, it maintains a second connection to receive more requests from the same client. The request on hold returns immediately with a possible empty body leaving its socket free, while the second connection serves the polling loop. The exchange of roles of those two connections allow to pull data from multiple contexts instead of being locked in just one.

SIP [25] is a protocol used for negotiation, creation, modification and finalization of communication sessions between users. 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 call's applications over IP networks.

The working group SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)[6] proposed the creation of SIP extensions, namely presence information [13] and instant messaging [4].

SIP is used in Voice Over IP (VoIP) applications due to its compatibility with the PSTN. Service providers are making their SIP infrastructures available through WebSockets. Frameworks like *jsSIP*[7], *QoffeeSIP*[8] and *sipML5*[9] 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.

XMPP was initially developed for instant messaging and presence (Jabber[10]). It is nowadays an open technology for standardized, decentralized, secure and extensible real-time communications. XMPP messages are eXtensible Markup Language (XML) based, which is attractive for applications that need structured messages and rich hypermedia. Another advantage of XMPP is the addition of extensions, for ex-

[4] `http://socket.io/`(accessed June 1, 2015).
[5] `http://github.com/sockjs`(accessed June 1, 2015).
[6] `https://datatracker.ietf.org/wg/simple/documents/`(accessed June 1, 2015).
[7] `http://jssip.net/`(accessed June 1, 2015).
[8] `http://qoffeesip.quobis.com/`(accessed June 1, 2015).
[9] `http://sipml5.org/`(accessed June 1, 2015).
[10] `http://jabber.org/`(accessed June 1, 2015).

ample [18], which adds file transfer capabilities between two entities and [28] which enables multi-user chat. XMPP's bi-directional communication over HTTP is achieved through BOSH [21]. This kind of communication is also possible through WebSockets [34]. Today, multiple XMPP server implementations exists, such as: *ejabberd*[11], *Metronome*[12], *Openfire*[13] and *Prosody*[14]. *Ejabberd* is the server that implements more Request For Comments (RFC) specifications and XMPP Extensions (XEP)s[15].

One of the advantages that XMPP based applications offer is the possibility to send messages across different servers, an XMPP user is identified by its *Jabber ID* which is definined by the pair username and server name in the form *username@server*.

The user profile is represented in terms of Data Forms[6] protocol and managed (registration and update) through *Info/Query* (IQ) [27] requests to the XMPP server which responds with an IQ response containing information to retrieve the registration fields needed to fill its profile (eg username, password, telephone).

An important aspect of this model is the ability to define a user as a set of attributes that can change dinamically in order to fit the IM applications needs.

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.

Typically a web application consists in multiple web pages which are navigated by users. Each time a webpage 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*, an automatic mechanism is required for avoiding the user implicit reauthentication.

One solution for reauthentication can be achieved by storing the user's JabberID and password on local storage and everytime a page is accessed the authentication is performed without the users knlowledge. Clearly this solution represents security flaws, the local storage can be easily accessed (HOW? explain!) locally or by performing a cross site scripting attack and reveal all the local storage, the same problem arises if the credentials are stored on cookies.

An alternative solution is known as Session Attachment [16], which requires that a *session identifier*(SID) together with a *initial request identifier (RID)* is 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 which is worthless if a user mantains more than one tab opened, for example for multiple conversations at the same time.

Another alternative would be the development of our solution in a single web page, which would increse drastically the complexity of our web application.

---

[11] `http://jabberd.im/`(accessed June 1, 2015).
[12] `http://lightwitch.org/metronome`(accessed June 1, 2015).
[13] `http://igniterealtime.org/projects/openfire/`(accessed June 1, 2015).
[14] `http://prosody.im/`(accessed June 1, 2015).
[15] `http://en.wikipedia.org/wiki/Comparison_of_XMPP_server_software`(accessed June 1, 2015).
[16] `https://metajack.im/2008/10/03/getting-attached-to-strophe/`

We know that an XMPP based application could simplify our work just by using functionalities that XMPP already implemment, but from all the XMPP features, we don't need all of them, in fact we just need a subset of XMPP features, namely a simple way to register users, access and edit user's information, create chat rooms, send messages and access to presence information.

Another interesting approach for signaling would be Signaling-On-the-fly (SigOfly) which allows inter-domain real-time communications, while abstracting the protocol used [5]. SigOfly provides inter-domain communication by making use of the *Identity Providers* of each peer. The caller entity downloads a page with all the code needed, also known as messaging stub, to communicate with the called party. This code contains an implementation of the signaling protocol used in order to communicate to the called peer. If the called party domain is being overused, it is possible to switch the caller and called parties role, after that the called entity downloads the stub code from the caller domain instead. SigOfly is an approach very flexible because participants on a video call are not tied to just one type of signaling implementation. Another important aspect of SigOfly is the ability to perform multi-party conversations either through a *Mesh Topology* or a *Multipoint Control Unit*.

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.

Hypertext is a type of text that provides links to texts or other types of content, these links are known by *hyperlinks*. *Hypermedia* is an evolution of *hypertext*, it includes audio, images, text and video.

Hypermedia concept brings the possibility to organize and overlay multimedia elements into a nonlinear linear structure.

In the beginning of analog video technology, the navigation over it was quite limited to simple operations such as play, stop, rewind and fast-forward. As video started to be digitalized, new operations over video emerged, such as random jumps and chapter navigation through interactive menus present on Digital Versatile Discs (DVD).

Some Moving Picture Experts Group (MPEG) implementations, like [29], added hypermedia information to empty space present on MPEG frames in order to provide interactive television, modifying the MPEG encoder and decoder in order to handle hypermedia content. Hypermedia is a concept that holds the promise of future technology and features but it is also already present in our daily lives.

Subtitles are an example of information that might be required. The need to translate movies, raised the problem whether it is appropriate to change the original video or audio. For example, subtitles should be an entity independent from the video, in order to be personalized or replaced easily.

Synchronized Accessible Media Interchange (SAMI), and SubRip Text (SRT) are two of the multiple formats for subtitles commonly supported by video players. Although those formats have styling available, they are quite limited to text.

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* [30] 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.

Detail-on-demand is a subset of hyper-video that allow us to obtain additional information about something that appears along the video, like obtaining information about a painting that appears in a particular segment. *Hyper-Hitchcock*[33] is an editor and player of detail-on-demand video.

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[35] is an XML based language that was proposed for modeling composition, synchronization and interaction of hypermedia. HyVAL defines defines video structure, internal video and external media objects. *HyVAL* uses a primary video stream, around which all other elements are organized and synchronized. *HyVAL*'s video structure object defines a structure derived from traditional video, which divides video into segments, scenes, shots and frames hierarchically. This approach is quite restrictive if we want to apply hyper-video concepts to videos that don't follow this structure. External media objects are linked by primary video, those objects can represent other videos, images, text, animation and sound.

Synchronized Multimedia Integration Language (SMIL)[2] 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.

SMIL is an XML based language that defines twelve modules: *Animation*, *Content Control*, *Layout*, *Linking*, *Media Objects*, *SmilText*, *Metainformation*, *Structure*, *Timing*, *Time Manipulations*, *State* and *Transitions*.

- The **Animation** module contains elements and attributes that define a time based mechanism for composing the effects of animations. For example, this module can perform changes on XML or CSS attributes like color and dimensions.
- The **Content Control** module contains elements and attributes that provide optimized alternatives for content delivery. For example, it could be used to change audio language in function of user's nationality, for videos with multiple audio channels.
- The **Layout** module contains elements and attributes for coloring and positioning media content. Other layout mechanisms are also possible, such as CSS.
- The**Linking** module contains elements and attributes for navigational *hyperlinking*. Navigation can be triggered by events or user interaction.
- The **Media Object** module contains elements and attributes for referencing rendering behavior of external multimedia or control objects.
- The **SmilText** module contains elements and attributes that define and control timed text. For example, this module could be used to create labels and captions.

– The **Metainformation** module contains elements and attributes that allows describing the SMIL document. For example, this module could be used to define movie details such as category, director, writers and cast.
– The **Structure** module defines the basic elements and attributes for structuring SMIL content. This module defines a *head* element that contains non temporal behavior information defined by *Metainformation*, *Layout* and *Content Control* modules. This module also defines the *body* element, where all temporal related module information is contained.
– The **Timing** module is the most important module on SMIL specification. Due to its complexity, it is divided into seventeen sub-modules for coordination and synchronization of media over time. The three main elements are *seq*, *excl* and *par*, that, respectively, play child elements in sequence, one at a time and all at the same time.
– The **Time Manipulations** module adds time behavior attributes to SMIL elements, such as speed, rate or time.
– The **State** module defines attributes that define the state of SMIL elements, such as element visibility, current element time, amount of repeated loops, playing state and many others.
– The **Transitions** module defines attributes and elements for transitions across multiple SMIL elements according to the *Timing* module.

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[24].

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's modules can be used to synchronize and animate eXtensible Hypertext Markup Language (XHTML) and Scalable Vector Graphics (SVG) elements.

SMIL fits our goals for creating a multimedia rich hyper-call, but it lacks on browser compatibility. Ambulant [1] was one of the SMIL players that were developed for browsers, although this player implements most of SMIL 3.0 [3] specifications, it needs to be installed on browsers as a plug-in.

SmillingWeb [8] attempts to implement a cross platform multimedia player designed for SMIL 3.0 presentations with *JavaScript* and *jQuery* which, unlike [1], doesn't require a plug-in to be installed and shouldn't have incompatibility issues. SmillingWeb already takes advantage of HTML5 and CSS3. It takes into account unsupported web browsers through the use of *Modernizr*[17], a simple *JavaScript* library that may require plug-ins if new features aren't supported. But SmillingWeb just implements a subset of SMIL 3.0 and their scheduler engine loads the SMIL file only once, which could raise problems when dealing with SMIL changes due to real time communications. Another problem with SmillingWeb is pre-loading and playing elements at the correct interval

[17] `http://modernizr.com/`(accessed June 2, 2015).

of time, which is not always possible due to high latency networks leading to pauses during playback.

An alternative to SMIL's *Layout* module is to use HTML which is a markup language based on XML that is used for creating web pages. HTML alone is a very poor language when we are focused on visual appealing and interactive web pages. Languages like CSS and *JavaScript* are typically used along with HTML for improving the interaction and appearance of a web page.

CSS's goal is to separate the structure of an XML document from its appearance. CSS defines styles for XML tags based on their name, class, identifier or position. Besides static styling CSS also supports animation and transitions leading to more dynamic content.

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.

With the emergence of HTML5, tags like *video*, *audio* and *track* allow us to play video with multiple codecs, audio and subtitles in Web Video Text Tracks (WebVTT) format. Another important tag is *canvas* that allows drawing graphics with *JavaScript* on a rectangle within a web page.

For example, with APIs like WebGL[18], it is now possible to manipulate a three dimensional environment in the context of a hyper-call. Another example would be a collaborative spreadsheet using WebRTC. With this, hyper-calls are not limited to only audio, image, text and video, but also interaction with complex graphical user interfaces that changes over time.

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.

Back in 1995, *flash*[19] was developed for web-based animations. Introducing video support in 2002, *flash* started to grow after that. Competitor's players, at that time were focused on playing video and audio, while *flash* had vector graphics and focused on streaming *on-demand* video across multiple platforms. *VP6* was their choice of video codec, providing half the video size (in Byte) for the same quality and providing video quality adjusted to *Internet* connection latency. In 2010, *Adobe Flash* was the most widely used applications for reproducing live broadcast and recorded video [16], it supports progressive video download using HTTP and streaming using Real Time Messaging Protocol (RTMP).

[18] `http://khronos.org/webgl/`(accessed June 2, 2015).
[19] `http://www.adobe.com/products/flash.html`(accessed June 2, 2015).

RTMP is a TCP based protocol used for streaming audio, video and data between a Flash Media Server (FMS) and *flash* players. A bidirectional connection is established between the two in order to allow real time communications. A *flash* player can stream a webcam video to a FMS using RTMP or it can request a video stream to FMS that can either be a pre-recorded stream, live stream or data. Multiple FMS servers can be used in parallel to increase capacity and handle more streams simultaneously.

FMS can stream video and audio to one or more subscribers by sending a separate copy for each subscriber. With Real-Time Media Flow Protocol (RTMFP) it is possible to stream video directly between *flash* players, allowing a publisher to break up a stream into pieces that can be cooperatively distributed in a P2P mesh. RTMFP uses UDP to speed packet delivery, which although it is not reliable, is well suited for video streaming. Like WebRTC, *flash* players also need to apply techniques like STUN and TURN for NAT traversal.

Although HTML5, *JavaScript*, CSS and WebRTC implement some of *flash*'s features, it doesn't mean that *flash* will be replaced soon. Instead, both technologies can be used to develop rich *Internet* applications. It is also important to note that HTML5 is better supported in mobile devices than *Adobe Flash*.

Like *Flash*, *Microsoft Silverlight*[20] is a cross browser plug-in and platform that is used to develop rich *Internet* applications. It supports vector graphics, animation and video. Compared to *flash*, which uses *ActionScript*, *Silverlight* applications can use languages like C#, *VisualBasic* and eXtensible Application Markup Language (XAML). *Silverlight* uses a technique called *Smooth Streaming* from *IIS Media Service* that consists on delivering video in real-time with adjusted quality in function of bandwidth variations and Central Processing Unit (CPU) usage.

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 2.5.1 presents the RTP protocol how it can be used to record media streams. Section 2.5.2 describes the media types and what types of media should be streamed. Section2.5.3 addresses how an interactive media can be recorded. Section 2.5.4 presents the collaborative environment and libraries to synchronize distributed object among users.

2.5.1 Streaming and Recording

If, for instance, one wants to rewind a real-time video, recordings will be needed from

[20] `http://www.microsoft.com/silverlight/`(accessed June 2, 2015).

whom is streaming the video. 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)[32] 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.

Real Time Control Protocol (RTCP) is used for controlling RTP multimedia streams, it provides bandwidth statistics and control information that can be used for changing the quality of the stream in real-time.

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 [31], which lists the payload encoding and compression algorithms. This profile also assigns a name to each encoding which may be used with other protocols like SDP. Another profile for RTP is defined by SRTP, which provides encryption, authentication and replay protection for RTP traffic. The analogous secure protocol for RTCP is Secure RTCP (SRTCP).

Both SRTP and SRTCP use Advanced Encryption Standard (AES) by default, which is a symmetric-key algorithm for data encryption. Each packet is encrypted using a distinct key-stream, as otherwise, using a single key-stream with AES on Cipher Block Chaining (CBC) would make it impossible to recover from packet loss.

Two key-stream generators for AES were defined: *Segmented Integer Counter Mode* and *f8-mode*. If a packet is lost, there is no impact on other packets, as the initialization vector is obtained through those key-stream generators and it is fixed for each packet.

RTP recorders are independent of payload encoding, they don't decode RTP packets, they record packets instead, allowing to record all video and audio formats even if they're encrypted.

Even though one on one calls are common, there are occasions when several people take part of the same video call. Multi-party video calls can be achieved on WebRTC by streaming video from each participant to all the other participants. Although this works, the bigger a conference room is, the bigger is the bandwidth used to stream video to all participants within the conference room. In this scenario, a more efficient alternative to peer-to-peer is the use of a Multipoint Control Unit (MCU).

Jitsi Video Bridge [21] receives one stream from every participant on a conference, either from a *jitsi* client or a *WebRTC* application, and redirects it to all the other conference participants, reducing the amount of data that each peer sends. Although all the participants still need to download all the streams from the *Jitsi Video Bridge* server, typically download rates are much bigger than upload rates, making this solution more feasible. *Jitsi Video Bridge* uses XMPP as a signaling protocol and its *colibri* exten-

[21] `http://jitsi.org/Projects/JitsiVideobridge`(accessed June 2, 2015).

sion [11] to reserve channels for video transmission. Despite this choice for signaling protocol, *Jitsi Video Bridge* also supports SIP.

Kurento Media Server (KMS) supports transcoding, group communications, recording, mixing, broadcasting, aplying filters, image and sound analysis, for example it allows face and qrcode detection. KMS functionalites are exposed through *JSON-RPC* over *WebSocket*, there are three clients available: JavaScript Client for web browsers, Java Client for Java EE Servers and JavaScript Cleint for Node.js servers. KMS suports streaming over *WebRTC*, HTTP and RTP endpoints. Another important component of KMS is *Kurento Repository* which suports recording and playing directly from *MongoDB*, that is important for providing a scalable media storage. Unlinke *Jitsi Video Bridge*, 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.

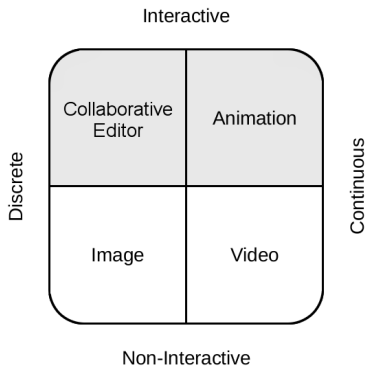


Fig. 3: Media Types

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.

Streaming protocols like RTP and Real Time Streaming Protocol (RTSP) were designed for continuous and non-interactive media types, such as audio and video. Discrete and non-interactive media don't need to be streamed through RTP because they don't change with time. For example, if an image appears in a specific time interval, just the HTML or *JavaScript* that will reference the image must be streamed, the image itself is then transferred through HTTP.

In order to play a stream, a player must be prepared to interpret the stream content. For interactive stream the player must download an environment, decode the RTP payload to determine the state and display it to the user. Streaming interactive media like a combination of HTML, CSS and JavaScript requires more than interpreting the code: a streamed user interface may contain an internal state that is not shown on code.

2.5.3 Recording and Streaming Interactive Media

Mauve et al. proposed an RTP profile for real-time transmission of interactive media[15]. This new profile reuses much of video and audio profile implementation, integrating the interactive component. Hilt et al. explained how to record interactive video with this new profile [10].

Every time an event is processed on one of the endpoints, both sender and receivers state must stay synchronized, otherwise events may behave differently. To achieve synchronization of interactive data, most packets have three types: *State*, *Delta-State* and *Event*. State packet defines the environment complete state. Delta-State packets transports just the piece of state that changed. Event packets informs that an event occurred over the interactive media.

An RTP recorder can have two operation modes, recording or playback. Traditional RTP players can do random access, in contrast, interactive RTP players must restore the environment and context at a given time. The environment is the initial state, so we can call it a non-interactive discrete media and handle it over HTTP. After the receiver has received the environment, it should calculate the state at the given time.

If the RTP recorder controls the correct data to send to the receivers, it cannot be a simple RTP recorder as it must compute the state or delta-state to send. Therefore, if the receiver receives all recorded packets, it can calculate the current state from the previous complete state. Streaming too many complete states results on more precise random accesses, but the trade-off is the higher bandwidth usage and used storage space on the recording server. On the other hand, if there are fewer complete states recorded followed by delta-states, the recorded stream will occupy less storage space, but random accesses will be less granular. It is possible to restore the media state even if messages are lost by recording and streaming the interactive media's complete state periodically.

In order to synchronize an interactive application state amongst participants, the needed objects to synchronize must be serializable and sent to other participants.

Mauve et al. concluded that the ability to extract the objects state in order to synchronize them and the ability to intercept events in order to control remote objects can be realized using the Model-View-Controller (MVC) concept[15]. This concept separates three components within an application. The *Model* represents the information itself. The *View* component shows the *Model* to the user in a suitable and interactive way. The *Controller* represents an action from the user to the *Model*.

Using the MVC concept will it make possible to implement an interactive WebRTC application that records, play, fast forward, fast rewind, stop and jump to random positions.

2.5.4 Collaborative Environment

Google Wave was a distributed collaboration platform based on *Jupiter*[19] that adopted

Operational transformation (OT) techniques. 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. Other Google products, such as Google Docs, are also using this type of technology. In 2010 Google stopped the development of Google Wave and released the main components as Open Source code to Apache, the project is currently known as *Apache Wave* and the reference implementation is named as *Wave in a Box*.

ShareJS[22] is an OT *JavaScript* library, developed by the ex *Google Wave* engineer Joseph Gentle, for collaborative text and JavaScript Object Notation (JSON) documents edition in real-time. It uses *ShareDB* [23] for its backend and data model, which supports simple integration with any database. One of *ShareDB*'s integration is over *MongoDB*[24].

TogetherJS[25] is a JavaScript library that uses WebRTC for collaborative web applications. It uses JSON messages for OT concurrency control but it does not provide storage. *TogetherJS* uses its own servers for the signaling phase and it supports microphone and mouse sharing between users. It requires a simple server (also known as hub) that echoes messages between clients, all the synchronization work is performed by the clients. Altough the hub's reduced complexity, *TogetherJS*'s server is implemented with *NodeJS*, if we would not choose *NodeJS* for server implementation we would need read the hub's source code and understand what changes are needed to perform on our web server in order to make it compatible with *TogetherJS*.

Goodow[26] is a collaborative framework with its own server implementation, it supports four types of collaborative elements: String, Lists, Maps and Custom objects.

OT.js[27] is a *JavaScript* library that only implements operation transformations on client side over plain text. 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.

For ease of development we decided to use the same backend for OT since our web server relies on *Play Framework* with Java and *ShareJS* relies on *NodeJS*[28]

2.6 Model

The model is the most important component of our solution, a badly designed model can imply serious deficulties when implementing new features that are not part of the plans, sometimes we had to redesign the model in order to support new features.

2.6.1 Generic model For designing our model we have taken into account generic programming techniques. We observed that operations like searching for an object was quite repeated amongst different types of objects.

[22] `http://sharejs.org/`(accessed June 2, 2015).
[23] `https://github.com/share/sharedb`(accessed March 10, 2016)
[24] `https://github.com/share/sharedb-mongo`(accessed March 10, 2016)
[25] `http://togetherjs.com/`(accessed June 2, 2015).
[26] `http://realtimeplayground.goodow.com/`(accessed June 2, 2015).
[27] `http://operational-transformation.github.io`(accessed March 10, 2016)
[28] `https://nodejs.org/en/(accessedMarch10,2016)`

Our first decision for our model in order to avoid repeated code, was the isolation of the object's attributes from themselves, so we could apply the search operation to a set of attributes indepently from the object type. To this generic set of properties we call data and each object of this type has a reference to the owner, which is a unique identification number.

Data	
◆ id	ObjectId
● owner	ObjectId
● properties	Document
○ searchableValue	List<Text>
○ identiableKeys	List<Text>

DataPermission	
◆ id	ObjectId
● owner	ObjectId
○ readPermissions	Document
○ writePermissions	Document

Table 3: Generic data model

The identication number by itself is not sufficient to identify an object, objects from different types can have the same identification number. In order to solve this problem, when an object is created, its correspondent data must contain the owner's object type.

When an attribute is created it must be specified if the attribute is searchable, otherwise it could be simple to search for attributes that could reveal sensible information about it. For example if we consider that a user could have a health related attribute, searching by a disease would reveal which users could suffer from a certain disease, the leak of that kind of information could, for instance, change the agreement between users and health insurance companies.

Another important attribute specification is the owner identifiability, which tells us if the attribute identifies the object. This specification let us create abstract authentication services, for example a user can login into our system by providing any attribute that identifies himself, for example the e-mail but others are possible like the username or cellphone number.

Not less important, our model supports defining read and write permissions for each attribute, for example an attribute can be set to public read access or private write access to a set of entities. Implicitly if an entity can write an attribute it can also read it.

The permission object is a mask of the data object, for each existing attribute in the data object it may exist the same attribute on the permission object specifying what entities can read or write it.

2.6.2 User model The user model is not tied to the user attributes, the information maintained in this model is just used for authentication purposes. Passwords are not stored in plaintext, instead we apply hashing and salting techniques [17] in order to make it harder to decode the password by an attacker. We use *SHA-1* and a random salt per user with 32 characters long.

User	
◆ id	ObjectId
● hash	Text
● salt	Text

Table 4: User model

2.6.3 Relation model A relation between two entities e_1 and e_2 is represented by the pair $e_1 \rightarrow e_2$, where e_1 is the source and e_2 is the target. This relation is said bi-directional if and only if it also exists the relation $e_2 \rightarrow e_1$.

Relation	
◆ id	ObjectId
● source	ObjectId
● target	ObjectId

Table 5: Relation model

A user can only interact with friends or with group members. In order to validate a friendship, both users must agree on that friendship, by other words it must exist a bi-directional relation between both users.

2.6.4 Group model A group can be public or private. If the group is public then it is visible to all users that mantain a friendship with a member of this group. If the group is private then it is only visible to their members.

Group	
◆ id	ObjectId
○ inviteToken	Text
● visibility	Text

GroupMembership	
◆ id	ObjectId
● groupId	ObjectId
● userId	ObjectId

Table 6: Group model

The group membership is a special case of relation, where the target entity is always a group.

When a group is created, a group membership is automatically assigned by its creator.

Entities that have a membership with a group can create more memberships by sharing an invite token or by specifying new group members.

2.6.5 Message model A message is composed by its content, sequence number, time of creation and source and target identification numbers. The message's target can reference any of object but our application is only covering messages to groups.

Message	
◆ id	ObjectId
● source	ObjectId
● target	ObjectId
● sequence	Number
● content	Text

Table 7: Message model

2.6.6 Hyper content model During a group conversation it is possible to tag points in time for making it easy to access that time either by searching or sharing with other users.

A time tag contains a title, the correspondent group identification number and the time itself.

The hyper content is used to synchronize content among users during a conversation. Every hyper content must have a start and ending time, the correspondent group identification number and the content itself in the form of text.

HyperContent	
◆ id	ObjectId
● groupId	ObjectId
● start	Date
● end	Date
● content	Text

TimeTag	
◆ id	ObjectId
● groupId	ObjectId
● title	Text
● time	Date

Table 8: Hyper content model

2.6.7 Collaborative Content model Within a conversation, users can write documents collaboratively. Each document has a content and a reference to the correspondent group.

CollaborativeContent	
◆ id	ObjectId
● groupId	ObjectId
● content	Text

Table 9: Collaborative content model

2.6.8 Recording model During a conversation, users may allow sharing their web cameras, by doing so their video is stored in recording chunks. Each chunk represents an interval of time $T = [c^{start}, c^{end}[$, it contains a reference to a group and a mapping between entity that was recorded and the correspondent video url.

RecordingInterval	
◆ id	ObjectId
● groupId	ObjectId
● start	Date
● end	Date

RecordingChunk	
◆ id	ObjectId
● groupId	ObjectId
● interval	ObjectId
● start	Date
● end	Date
● sequenceNumber	Number
● urls	Document

Table 10: Recording model

A set of chunks $S = [c_1, c_2, \ldots, c_n]$ is said continuous if $\forall c_i \in S, \exists c_j \in S$ where $j \neq i$ and $c_i^{start} = c_j^{end} \vee c_i^{end} = c_j^{start}$.

A recording interval represents a continuous set of recording chunks.

2.7 Signaling Protocol

The signaling protocol could be implemented using HTTP messages but they would transport extra information such as HTTP headers and would follow a request response signaling mechnism which would not be the best option, as multiple ice candidates can arrive at any time. For that reason bi-directional communications were prefered.

Our signaling protocol consists in send and receiving JSON formated messages over WebSockets by both the server and the client.

Listing 1.1: General structure of our WebSocket messages

```
1 {
2         "cmd":<cmd>,
3         "data":<data>
4 }
```

When a user enters in a group conference, after the page is completely loaded it is created a WebSocket that mantains a connection with our web servers. Before creating the web socket, we must identify the user and check if he has permissions to participate in the conference. The user identification is done by retrieving the session id from the cookie provided by the user through the HTTP headers then we retrieve all the information need from database in order to check if the user has permissions to join the conference room. It's important to save de user identification before the socket is created because after the the handshake performed by the *WebSocket* protocol[7] the HTTP context is lost.

When the connection is established between the server and the client, a *PeerConnection* is created on the client and imediately after an WebRTC endpoint on the server specifying a possible set of ICE servers to connect.

The user is asked if he whants to share its camera and microphone, share screen or just receive streams from the server. If the user decides to perform a screen share, *adapter.js*[29] may ask to install a pluggin if the browser doesn't support screen sharing.

If the user decides to share either from camera of screen, *getUserMedia* is called with the correspondent contraints in order to obtain a local stream.

Listing 1.2: Media constraints

```
var screenShareConstraints = {
        "video": {
                "mediaSource": "window" || "screen"
        },
        "audio": false
};
var cameraMicrophoneConstraints = {
        "audio":true,
        "video":true
};
var receiveOnlyConstraints = {
        "offerToReceiveAudio":true,
        "offerToReceiveVideo":true
};
```

A popup is raised in order to ask the user to give permission to share those resources. If the resources are shared successfully the user creates an offer, sets a *local session description* to its *PeerConnection* and sends it through WebSockets. If the resources are not shared or the user specified to receive stream only, an offer is created specifying the contraints for receiving only video and audio.

The *local session description* contains the session identifier, codecs, containers, transport protocols and ports used per media type. The *local session description* is useful to conclude if the client is receiving only, which means that KMS doesn't need to mix, record and analyse streams coming from the user WebRTC endpoint.

The server receive and processes the offer and sets the *remote session description* to its client associated WebRTC endpoint. Then a *local session description* is created on the server and sent back to the client, after that the server trys to gather ICE candidates.

The client receives the server answer, sets the *remote session description* and gets the ice candidates from the ICE server.

After a while both the server and client receive the ICE candidates that allow the client to connect directly to KMS and vice-versa. The candidates are received at the client which sets them to its *PeerConnection*. The same is done on the server which receives the ICE candidates from the client and propagates them to KMS.

[29] `https://github.com/Temasys/AdapterJS`(accessed March 15, 2015).

An ICE candidate contains an IP, port, used transport protocol and an attribute named *sdpMLineIndex* that is used for mapping to the *remote session description* media type.

Both intervenients test the conectivity of each ICE candidate. When a connection is established the user and server start to interchange stream data but other ICE candidates may arrive with better connections, when that happens the connection changes seamlessly.

With the media session established, the server starts to record any received stream and the client creates an Uniform Resource Locator (URL) correspondent to the stream location.

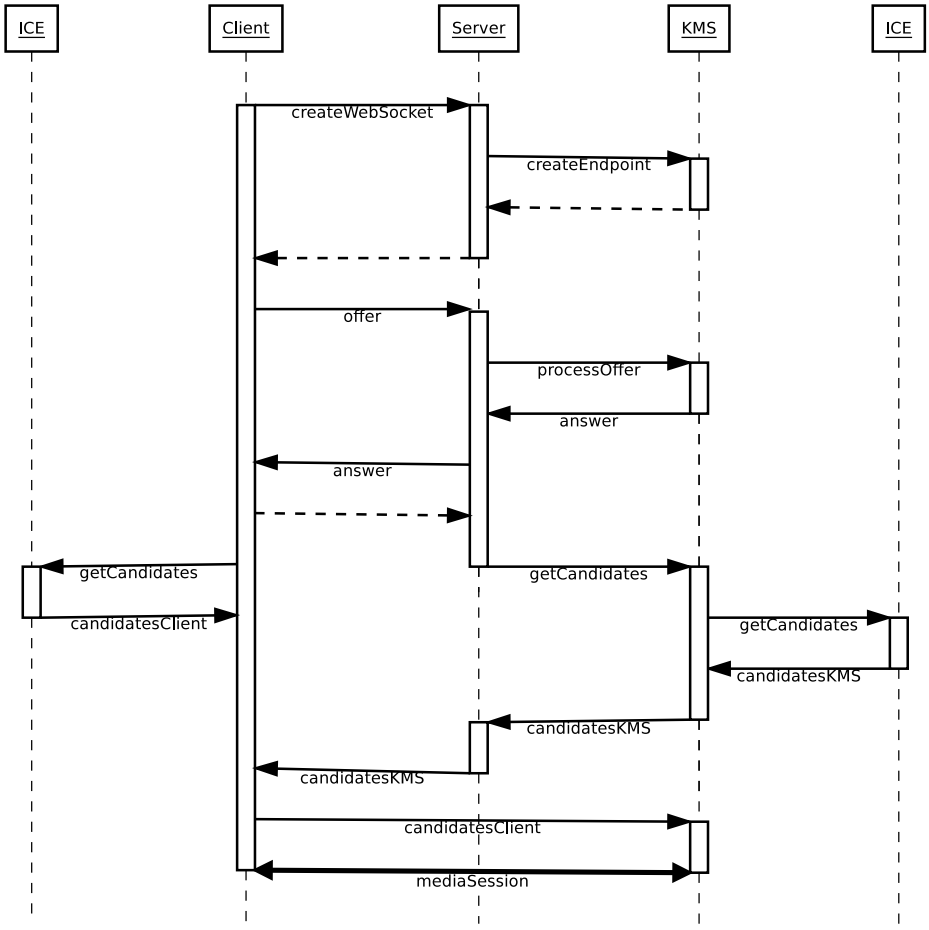


Fig. 4: Signaling sequence diagram

2.8 Stream Recording

In this section we describe our approaches for the stream recording implementation, namely recording on the client side and server side.

2.8.1 Client side recording In order to record used shared streams, our first approach consisted on recording video and audio streams into blobs of limited duration. Because each user was recording directly from their shared local resources, we achieved the best stream quality.

After recording, each block was uploaded to our server through HTTP and saved into the filesystem, the block metadata was created and inserted on our database. Each block metadata contained the file's location for the recorded block, the starting date, duration, user identification and group identification. When the file was completely saved the metadata was advertised to the remaining users. This metadata was simply used to refresh the user interface, the metadata was completely discard after that because an huge amount of small blocks metadata would use more and more memory as time went by.

The process to play a video was quite simple, the user specified which user and date was intended to play, the server calculated the intersection between the requested date and the block bounds and returned the file to the client.

Altough this idea was fairly simple, we couldn't achieve seamless sequential block switching. Downloading the video file always toke a noticeable amount of time. To solve this problem while we were playing a block, the next one was downloading in parallel so when the current block finished playing we would have an available block to play.

Block switching became more acceptable, but switching the url always produced a flash. We solved this problem by having two layers and set the url to the back layer, the front layer freezed with the last frame, then we changed the back layer to front.

After we implemented our recording solution using this approach, we tested localy and remotely. For remote connections we observed a fairly high bandwidth usage mainly because blocks were both sent and received at maximum quality.

2.8.2 Server side recording to filesystem Before recording any type of stream we had to analyse the user media offer in order to check if a video as realy being received by KMS, otherwise if we would not verify the user's offer, the recording video would be black.

The streaming content received on KMS was already compressed due to WebRTC's exchanged **QoS!** (**QoS!**) metrics data. As a direct consequence our recorder solution used on most cases less disk space per block, but would never use more storage than client side recording. KMS allows recording files using **WebM!** (**WebM!**) and **mp4!** (**mp4!**) containers.

With server side recording, the user would maintain always the same stream url even if it is playing realtime video or reproducing recorded video. When a user desires to play recorded video, a websocket message is sent specifying the time and the intended user id, the group identification is not sent because it is already associated to the websocket.

The server performs the same calculations in order to find a block that intersects the requested time, plays it and when finished the next part is automatically played wihtout the user intervention.

We observed differences in image quality when switching parts, that was even noticeable if we set a short block duration. We also noticed a small gap on audio when switching blocks but it was acceptable and speech recognition was not very affected.

Back when we were implementing our solution, KMS had not support for seeking videos, which meant that blocks would would always start playing from their beggining. This lead to a theoric playing time error that can be at most half the duration of a block. In practice the maximum playing time error coincides with the block duration because we perform the intersection between the requested date and the block but we could decrease the error by half if we calculate the intersection with the requested time plus half the duration.

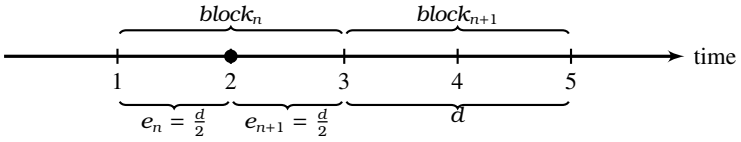


Fig. 5: Theoretic maximum playing error

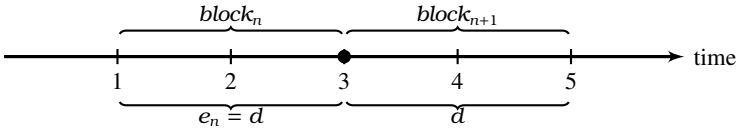


Fig. 6: Practical maximum playing error

For playing video with an higher velocity we used *ffmpeg*[30] to convert the block into a new video with the desired velocity and seek time. Because the media duration is known, when the video started to being convert the headers located at the begining of the file were already written and that made it possible to stream while converting.

Although we implemented a solution that worked, we imediately noticed that *ffmpeg* would take some time to initialize and that lead to pauses between switching parts.

Later the *Kurento* team released a version with support for seeking videos but we had to suspend the implementation of the fast forwarding feature as currently KMS is not supporting that.

We could implement fast forwarding without real time conversion by creating multiple versions of the same video with different velocities after the recording of a block.

[30] `https://www.ffmpeg.org/`(accessed: 17 March 2016)

When a user needed to play he would also need to specify one of the available velocities. We did not followed this approach has it would require a bigger disk space usage.

2.8.3 Server side recording to database One of our concerns during the development of our solution was the storage scalability. Saving files directly into filesystem would require an extra effort to distribute and replicate files among servers. For that purpose *Kurento* team developed *Kurento Repository*[31] which is based on *MongoDB*.

One of the features that *Kurento Repository* provides is the ability to play directly from the database without having to download the entire file to KMS. The same is true for recording, but because the file headers are in the beggining and the file is written until is stops, the headers don't contain the necessary information for seeking the file.

Although we gain with scalability with this approach, we lose access over the file for changing it to fit our needs, namely for using *ffmpeg* or other video manipulation tool.

Although we did not implemented recorded file seeking, that could be achieved by waiting for full file recording and then proceed to database insertion with the correct headers. Another approach would be the specification of the file duration before recording so the correct file headers could be written a priori. Both approaches were not possible to implement using just the *Kurento* clients, we would need change the source code of *Kurento* in order to add those new features.

---

3 Backend

3.1 Our approach

In order to implement a solution without the *XMPP*'s limitations that we presented on the previous section, we decided to implement our own backend.

One important aspect that we should take into account is the scalability of our solution. All the information that we need to store relative to the users and chat rooms should be stored on a scalable database.

As our database grows, we may need to distribue the load among multiple servers, which can be achieved on SQL servers by spliting tables among servers, but this could be a challenge operation when doing relations between tables. For this reason we opted for using a NoSQL database. Many of NoSQL databases are built having the scalability functionality from start.

One option for a NoSQL dabase is MongoDB, there are others but this one seems to have popularity, it is free and well documented. There are no strong reasons for excluding options like Apache Cassandra or others.

The authentication problem that XMPP raised when implementing a client on a web browser is easily solved by using cookies to identify authenticated users. A cookie is mantained as long as is specified by its expiration date and the user only needs to reauthenticate when it logs out or the cookie expires, changing from page to page does not raises any reauthentication problem.

[31] `http://doc-kurento-repository.readthedocs.org` (accessed on 17 March 2016)

On our solution the actions that are performed by clients are directly sent to web servers that control the validity of the actions and performs the respective changes to the database. This logic on the XMPP approach would be distributed among XMPP servers and not the webservers, so now our webservers will increase their complexity by implementing the features we needed from XMPP.

4 Proposed Architecture

Taking into account the goals of this project and all the technology presented so far. Our proposal is the development of a web application that provides communication and collaboration in real-time.

The requirements for our web application are:

- Text, Audio and Video communication using WebRTC.
- Enrich communications by overlaying multiple media types.
- Create annotations over video and voice content.
- Search and navigate through annotations.
- Structure presentation of the content.
- Ability to perform tasks on real-time collaborative environment.
- Ability to record and playback interactive multimedia.
- File sharing and collaborative edition.

4.1 Modules

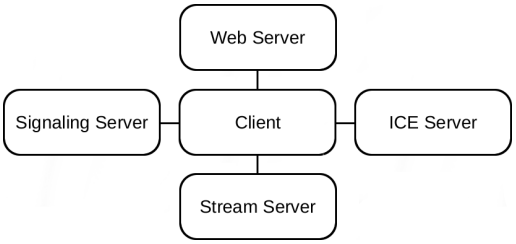


Fig. 7: System Modules

Figure 7 presents the structure of our system which is divided into five modules.

The Web Server sends Web Pages, containing the other modules information, to the client, after this, the client contacts all the other modules. The ICE server will be used for NAT traversal. The Stream Server will be used to record streams for further playback. The signaling server will be responsible for user and calls management and presence information. The web server will provide libraries to the client in order to interact with the other modules.

Clients can discover chat rooms and other clients by querying the signaling server, they can create rooms for multi-party communication which is achieved by using WebRTC's *MediaStream* and *DataChannel*, namely for audio, video and text communication.

Chat rooms can be public or private. Public chat rooms are moderated by a group of clients which initially is formed by the room creator, this type of room has no access restrictions by default, but that can be changed by its moderators. Private chat rooms will be visible for a list of clients or can be accessed by clients that have a link for that chat room.

Each room will be associated to multiple Media-Types, interactive and non-interactive, discrete and continuous. This media types can be structured with chains of events. Some components of the user interface may not be synchronized to all participants, for example an help or suggestion window.

4.2 Implementation Proposal

The infrastructure is composed by: Ice Server, Web Server, Stream Server, Signaling Server and Test Clients.

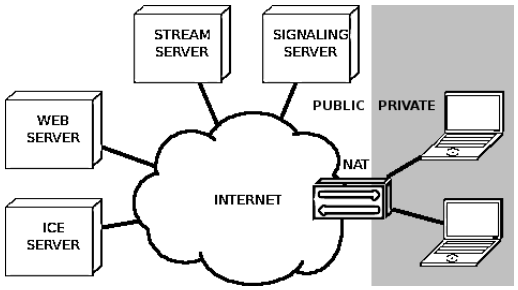


Fig. 8: System Infrastructure

Although the core infrastructure is important and crucial for the implementation of our web application, there is no preference for a specific software running on the web server. An important requirement for the Web Server is the WebSockets support, The *LAMP* stack was considered, although the chosen technology for web development is the *Play Framework* with *Java*, which has a more evident separation between *Model*, *View* and *Controller* components.

For the signaling server there is also no preference between SIP and XMPP, as both support presence information feature. SigOfly could be adopted as well, but using it would be irrelevant for the final result of this project. For the sake of simplicity and easy deployment, the chosen platform is the *Ejabberd* XMPP application server, since it implements [21] which is crucial for web applications. *Ejabberd*, as said before, is also the XMPP server that implements more extensions. Initially, for simplicity, *Ejabberd* will be configured as a single node instead a member of a cluster.

The streaming server will be the *Jitsi Videobridge* as the *Jitsi* team is working with WebRTC and their code is open source.

The ICE server is not required to be on our infrastructure, as a public ICE server could be used. If TURN is used, the network speed could drop due to resource sharing with other users. An ICE server will be installed in order to prevent the influence of other users on our application.

On the client computers, both *Mozilla Firefox* and *Google Chrome* will be installed as web browsers. Libraries such as *jQuery*, *Bootstrap*, *Strophe*, *Modernizr* and *TogetherJS* will be downloaded from the Web Server and executed on the client side.

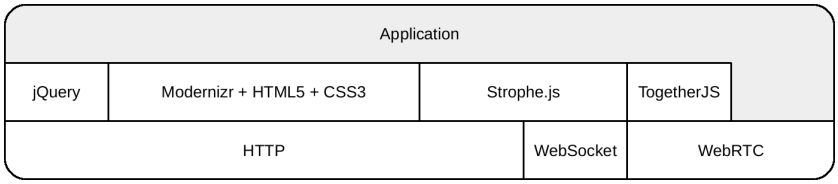


Fig. 9: App Architecture

Figure 9 presents the application architecture and the underlying technologies. *Modernizr* and *jQuery* will ensure that our application is compatible with the most popular web browsers. *Bootstrap* will be used to make the user interface more appellative and responsive. With *Bootstrap* it is quite easy to develop applications that adapt to mobile devices with different screen sizes. The *Strophe* library will be essential to communicate with the XMPP server.

TogetherJS will be used for the collaborative component of our web application. Other libraries were considered but, as said before, *TogetherJS* does not provide object storage giving us the ability to choose how to store collaborative objects. We assume that *JavaScript* objects are enough for implementing our application but *TogetherJS* provides the flexibility to change our storage model.

The synchronization between multimedia elements will be performed trough chains of *JavasSript* events, other animations can be implemented with SVG embedded on HTML.

5 Methodology

This section presents how we plan to implement and test our solution.

5.1 Evaluation Methodology

Our solution will be evaluated with several experiments, either qualitative and quantitative.

Our web application will be tested with real users by asking them to perform some tasks, count the respective spent time and note their difficulties. After these tests, it will be suggested to users to fill out questionnaires.

The results of user tests be used to improve the following prototype iterations, it is expected to notice both qualitative and quantitative improvements.

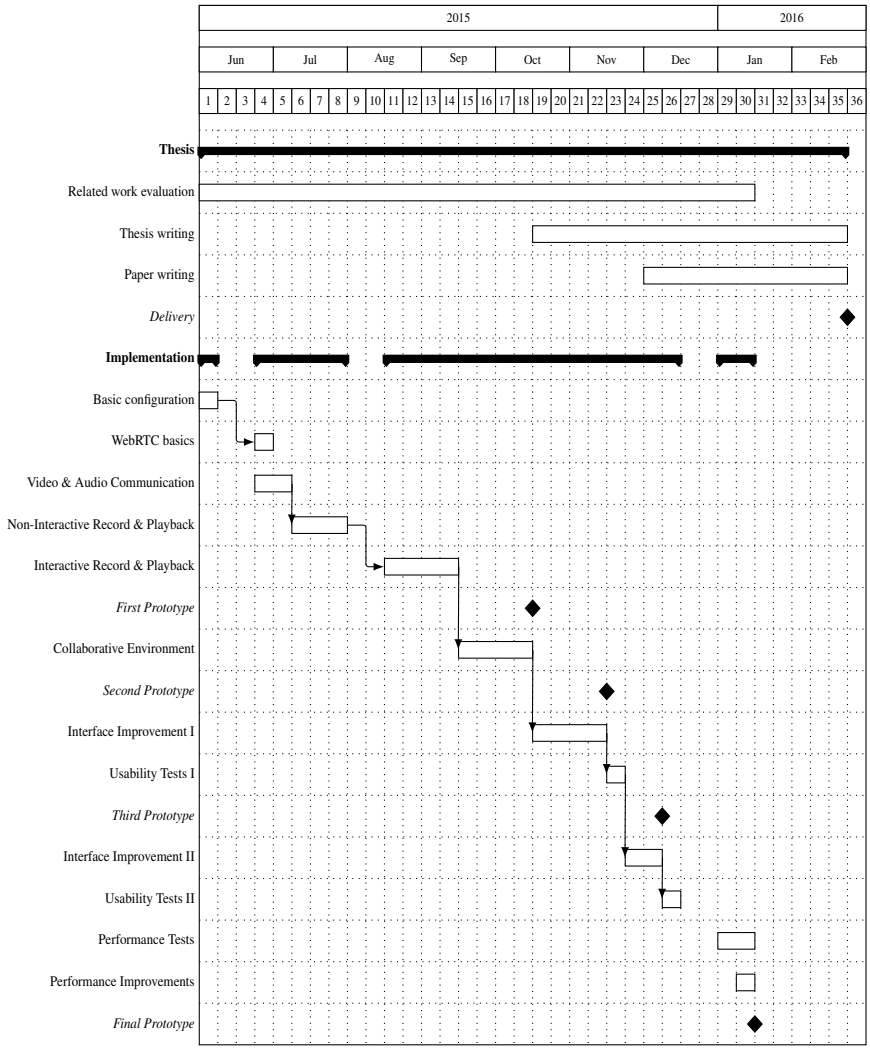
The system in general will be validated with benchmarks that will focus on application delay when performing heavy tasks such as sharing a file and editing it in collaboration with other users. The amount of users will gradually increase in order to understand how much the performance degrades.

5.2 Planned Schedule

The most important task in order to implement our project is the infrastructure, without it our project wouldn't work. The basic infrastructure will be our first concern, namely the Web Server configuration, ICE and signaling servers.

The Web development can only start after the basic infrastructure configuration. When leading with streams, the infrastructure will be complemented with stream servers.

There will be four prototypes, the first one should implement most of streaming and collaborative functionalities. The second prototype will have the same functionalities but a more friendly user interface which will be tested and reviewed by potential users. The third prototype will be an improvement based on the users feedback. In the final prototype we will improve our system performance.



6 Conclusions

The development of WebRTC by the World Wide Web Consortium (W3C) allows web browser-to-browser audio, video and data communication. WebRTC is enabling new usage scenarios for communication and collaboration applications. This and other technologies allow the Hyper-link concept to be applied to a new media: real-time audio and video.

This project proposes a solution that uses WebRTC for creating an application that provides audio, data, text and data real-time communications. Theses communications will be enriched using hypermedia concepts, the possibility to record and playback its content and the integration of collaborative tools.

A prototype will be implemented in order to validate these concepts. In order to implement it, significant challenges, such as the need to record interactive audio and video streams, will have to be overcome, making this a challenging project.

References

1. 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. pp. –. W3C (December 2007), http://oai.cwi.nl/oai/asset/11999/11999A.pdf
2. Bulterman, D.: Smil 2.0 part 1: overview, concepts, and structure. MultiMedia, IEEE 8(4), 82–88 (Oct 2001)
3. 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)
4. Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., Gurle, D.: Session Initiation Protocol (SIP) Extension for Instant Messaging. RFC 3428 (Proposed Standard) (Dec 2002), http://www.ietf.org/rfc/rfc3428.txt
5. 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. pp. 1–8 (Feb 2015)
6. Eatmon, R., Hildebrand, J., Miller, J., Muldowney, T., Saint-Andre, P.: XEP-0004: Data Forms (2007), http://xmpp.org/extensions/xep-0004.html
7. Fette, I., Melnikov, A.: The WebSocket Protocol. RFC 6455 (Proposed Standard) (Dec 2011), http://www.ietf.org/rfc/rfc6455.txt
8. Gaggi, O., Danese, L.: A smil player for any web browser. In: DMS. pp. 114–119. Knowledge Systems Institute (2011), http://dblp.uni-trier.de/db/conf/dms/dms2011.html#GaggiD11
9. Handley, M., Jacobson, V., Perkins, C.: SDP: Session Description Protocol. RFC 4566 (Proposed Standard) (Jul 2006), http://www.ietf.org/rfc/rfc4566.txt
10. 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, Lecture Notes in Computer Science, vol. 1718, pp. 291–304. Springer Berlin Heidelberg (1999), http://dx.doi.org/10.1007/3-540-48109-5_24
11. Ivov, E., Marinov, L., Hancke, P.: COnferences with LIghtweight BRIdging (COLIBRI). XEP-0340 (Experimental) (Jan 2014), http://www.xmpp.org/extensions/xep-0340.html
12. Lin, Y.D., Tseng, C.C., Ho, C.Y., Wu, Y.H.: How nat-compatible are voip applications? Communications Magazine, IEEE 48(12), 58–65 (December 2010)
13. Lonnfors, M., Costa-Requena, J., Leppanen, E., Khartabil, H.: Session Initiation Protocol (SIP) Extension for Partial Notification of Presence Information. RFC 5263 (Proposed Standard) (Sep 2008), http://www.ietf.org/rfc/rfc5263.txt
14. Martin, G.: Hypertext to Hypervoice - Linking what we say and what we do p. 6 (2012)
15. 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. vol. 2, pp. 641–646 vol.2 (Jul 1999)

16. 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. pp. 375–380 (Nov 2010)
17. Morris, R., Thompson, K.: Password security: A case history. Communications of the ACM 22(11), 594–597 (1979)
18. Muldowney, T., Miller, M., Eatmon, R., Saint-Andre, P.: SI File Transfer. XEP-0096 (Draft) (Apr 2004), http://www.xmpp.org/extensions/xep-0096.html
19. 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. pp. 111–120. UIST '95, ACM, New York, NY, USA (1995), http://doi.acm.org/10.1145/215585.215706
20. Paper, A.I.U.W.: Next Generation Internet : IPv4 Address Exhaustion , Mitigation Strategies and Implications for the U. S. pp. 1–26 (2009)
21. Paterson, I., Saint-Andre, P., Stout, L., Tilanus, W.: XMPP Over BOSH. XEP-0206 (Draft) (Apr 2014), http://www.xmpp.org/extensions/xep-0206.html
22. Paterson, I., Smith, D., Saint-Andre, P., Moffitt, J., Stout, L., Tilanus, W.: Bidirectional-streams Over Synchronous HTTP (BOSH). XEP-0124 (Draft) (Apr 2014), http://www.xmpp.org/extensions/xep-0124.html
23. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard) (Jan 2012), http://www.ietf.org/rfc/rfc6347.txt, updated by RFC 7507
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 (Oct 1998), http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001
25. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard) (Jun 2002), http://www.ietf.org/rfc/rfc3261.txt, updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141
26. 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) (Mar 2003), http://www.ietf.org/rfc/rfc3489.txt, obsoleted by RFC 5389
27. Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard) (Oct 2004), http://www.ietf.org/rfc/rfc3920.txt, obsoleted by RFC 6120, updated by RFC 6122
28. Saint-Andre, P.: Multi-User Chat. XEP-0045 (Draft) (Feb 2012), http://www.xmpp.org/extensions/xep-0045.html
29. Sampaio Gradvohl, A.L., Iano, Y.: Matching interactive tv and hypervideo. Latin America Transactions, IEEE (Revista IEEE America Latina) 5(8), 579–584 (Dec 2007)
30. Sawhney, N., Balcom, D., Smith, I.: Authoring and navigating video in space and time. MultiMedia, IEEE 4(4), 30–39 (Oct 1997)
31. Schulzrinne, H., Casner, S.: RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551 (Standard) (Jul 2003), http://www.ietf.org/rfc/rfc3551.txt, updated by RFC 5761
32. Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard) (Jul 2003), http://www.ietf.org/rfc/rfc3550.txt, updated by RFCs 5506, 5761, 6051, 6222
33. Shipman, F., Girgensohn, A., Wilcox, L.: Creating navigable multi-level video summaries. In: Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on. vol. 2, pp. II–753–6 vol.2 (July 2003)

34. Stout, L., Moffitt, J., Cestari, E.: An Extensible Messaging and Presence Protocol (XMPP) Subprotocol for WebSocket. RFC 7395 (Proposed Standard) (Oct 2014), http://www.ietf.org/rfc/rfc7395.txt
35. 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. pp. 421–426 (Jan 2005)