Hyper-linked Communications: WebRTC enabled asynchronous collaboration

Henrique Lopes Rocha
email1: henrique.rocha@ist.utl.pt

Instituto Superior Técnico

Abstract. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

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, 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 seem banal nowadays, none of this was possible before the computer's invention.

As [13] states, "No computer in our lifetimes will ever rival a human voice's capacity to conveying rich and complex social and emotional meaning" , although nothing replace 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.

1.2 Problem Statement

As communications technologies appeared, we adapted the way we communicate. This project don'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.

For multiple reasons, we often need to repeat or postpone some of our tasks, some people tend to forget what they ear or see, either due to health problems or lack of sleep, a real-time system is not appropriate for people with short term memory loss, 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. Real-time communication applications can make a difference on business, education and health sectors by providing tools to understand how we learn and work together.

This project aims to extend audio, text and video communications in order to create rich and collaborative interfaces with better content organization and time handling. All of these with help of only standard technologies like Web Real-Time Communication (WebRTC), any additional plug-in is avoidable, *JavaScript* libraries will be preferred as they can be downloaded on the fly.

1.3 Thesis Goals

A web application with an easy to learn user interface will be developed to accomplish solve our problem. Our application, unnamed yet, is supposed to run on most web browsers that are compatible with JavaScript, WebRTC, HyperText Markup Language (HTML)5 and Cascading Style Sheets (CSS)3.

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

1.4 Document Structure

Section 2 is dedicated to related work, which is divided into five subsections.

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.

In section 3 we propose an architecture in order to develop our Web Application including all the need infrastructure and software.

Section 4 describes our work methodology and what is our plan in order to solve our problem.

Section 5 presents the summary and conclusions of our work.

2 Related Work

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 technology and the number of future users was unpredictable, caused that some protocols weren't suitable for a huge increase on the amount of publicly known users. IPv4 limits the number of public addresses in such a way that today is scarce [18]. 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 to redirect messages to members of its group through their private addresses, each member of the private network is identified publicly by the same Internet Protocol (IP) address with a different port, this technique is also known as Network Address Translation (NAT).

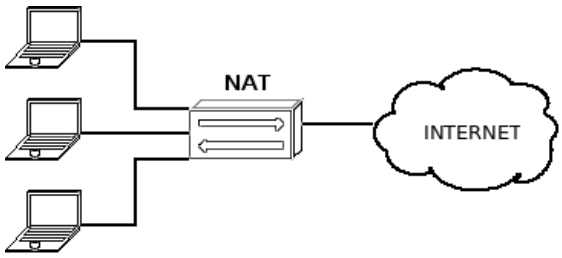


Fig. 1: Network Address Translation

Initially NAT offered an alternative to address exhaustion and a minimal sensation of security. There are four types of NAT implementations[22], *Full Cone NAT*, *Restricted Cone NAT*, *Port Restricted Cone NAT*, *Symmetric NAT*.

Full Cone NAT maps public each 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 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.

Asymmetric NAT became a vulgar configuration on the web. 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 difficult end-to-end communication, forcing most of applications that follow this model to be implemented ineffectively.

Applications based on multimedia and file sharing have been one of the most strained by NAT. Those kind applications require real time communication in order to achieve the best performance. Session Traversal Utilities for NAT (STUN) and Traversal Using Relays around NAT (TURN) [11] servers are a possible solution to overpass NAT, although, none of those can establish direct connections on multiple level NATs.

STUN servers are quite simple, they receive requests from NATed clients, the source address of a request is the public address that NAT mapped to the client, STUN servers will then reply the mapped public address to the client, so it knows its associated public IP address and port. Symmetric NAT changes IP port for each different connection, for that reason STUN servers will reply the IP address and port of their connection, which will be useless to clients connections, that's why Symmetric NAT represents a problem for real time communications.

On the other hand, TURN uses public servers to redirect 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) uses STUN when it's possible and TURN otherwise.

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 this type of communication requires a more expensive infrastructure and, in most cases, more network usage, leading to a worse quality of service. The requirements of 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 real time communications without installing any additional application or plug-in.

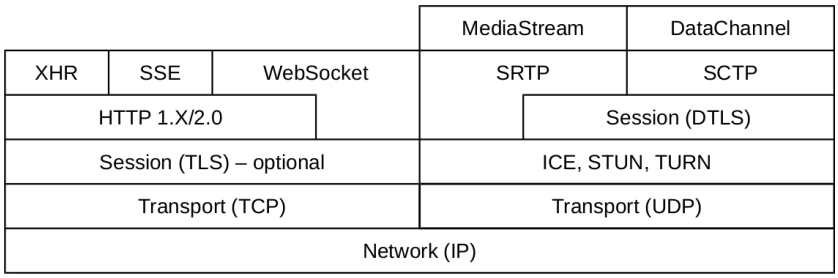


Fig. 2: WebRTC protocol Stack

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

- **MediaStream** allows from the browser to access to camera, microphone and device screen.
- **PeerConnection** acquires connection data and negotiates with peers.
- **DataChannel** allows to send whatever type of data to 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 packet order and integrity are not assured. 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 record decryption, for that it requires a reliable transport channel, typically TCP. The decryption of a record depends on the previous record, 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 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 TLS, 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

Fig. 3: 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 over the transfered data. A *Data Channel* has one incoming stream and one outgoing stream, providing bidirectional communication. Each data channel direction can be tweaked 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 for a particular stream compared to another.

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 communications are done through UDP without using DTLS.

Skype[1] is an application that allows video, voice and instant messaging communication over proprietary protocols, its main strength is the amount of users that are using it nowadays. But compared to *Skype*, WebRTC applications don't need to be pre-installed.

Google Hangouts[2] is a video conference web application, in the past in order to use *hangouts* on a web browser a plug-in was needed to be installed, nowadays hangouts is using WebRTC.

[1] `http://www.skype.com/`
[2] `http://plus.google.com/hangouts`

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 Videobridge* is a server that enables multi-party video calls.

2.3 Signaling, meet and get to know

Signaling is the most important process for applications to exchange connection information about peers and servers, their capabilities and meta-data.

WebRTC doesn't implement signaling, different applications may require different protocols, there is no single answer that fits all problems. Amongst multiple options, signaling can be done by 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) [8] 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 signaling requisites is bi-directional communication over Hypertext Transfer Protocol (HTTP). HTTP works on a request followed by a server response, by other words, follows a unidirectional communication. Sometimes it's required that some informations are obtained in real time, as we saw, some NAT's don't support callbacks from servers, one technique to overcome this problem is polling.

Polling consists on sending periodic messages that the server responds immediately with empty content or fresh information. Real time communications 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 later.

A technique called long polling consists on making the server hold the request until there is fresh information or expiring after some time, after the message receipt, 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 adds data overhead, especially for short messages.

The WebSocket protocol [6] allows bidirectional communications over a full-duplex socket channel. WebSocket handshake phase specifies an HTTP header in order to upgrade to *WebSocket* type of communication, the remainder messages are done without HTTP headers, which leads to much smaller messages and better network usage. WebSockets may not be available on every web browser, frameworks like *socket.io*[4] and *SockJS*[5] uses HTTP when there is no support for WebSockets.

[3] `http://jitsi.org/Projects/JitsiMeet`
[4] `http://socket.io/`
[5] `http://github.com/sockjs`

Bidirectional-streams Over Synchronous HTTP (BOSH)[20] is a technique based on long polling that uses two socket connections and allow sending client messages to server while a previous request is held.

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 the server may think 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, it worsens even more 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.

If the client has data to send while a request is still open, it establishes a second socket connection to the connection manager to send a new request. The connection manager immediately responds to the previously held request (possibly with no data) and holds open this new request. This results in the connections switching roles.

SIP [21] is 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 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 [12] and instant messaging [4].

SIP is used in Voice Over IP (VoIP) applications due to its compatibility with Public Switched Telephone Network (PSTN). Service providers making their SIP infrastructures available through WebSockets. Frameworks like *jsSIP*[7], *QoffeeSIP*[8] and *sipML5*[9] are used on 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 interoperate web communications with SIP networks, mobile and fixed phones.

[6] `https://datatracker.ietf.org/wg/simple/documents/`
[7] `http://jssip.net/`
[8] `http://qoffeesip.quobis.com/`
[9] `http://sipml5.org/`

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 are attractive for applications that need structured messages and rich hypermedia. XMPP advantage is the addition of extensions, for example [16], which adds file transfer capabilities between two entities and [23] which enables multi-user chat.

XMPP's bi-directional communication is achieved through BOSH [19], which basically consists on long polling. This kind of communication is also possible through WebSockets [29].

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].

Another interesting approach for signaling would be Signaling-On-the-fly (SigOfly) [5] which allows inter-domain real-time communications. 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 need, 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 o 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 that raised with it consisted of how to add more information onto it without generating multiple versions. Some implementations like [24] added hypermedia information to empty space present on Moving Picture Experts Group (MPEG) frames in order to provide interactive television, the MPEG encoder and decoder were changed in order to handle hypermedia content.

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.

[10] `http://jabber.org/`
[11] `http://jabberd.im/`
[12] `http://lightwitch.org/metronome`
[13] `http://igniterealtime.org/projects/openfire/`
[14] `http://prosody.im/`
[15] `http://en.wikipedia.org/wiki/Comparison_of_XMPP_server_software`

Amongst multiple formats for subtitles, Synchronized Accessible Media Interchange (SAMI), and SubRip Text (SRT) are used by video players that support them. 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. *HyperCafe* [25] was an experimental project to expose hyper-video concepts that consisted on an interactive film by switching between conversations 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*[28] is an editor and player of detail-on-demand video.

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

HyVAL[30] 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'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's possible to synchronize multiple sections of video, either in parallel or in sequence, reproduce a different audio track, overlay user interface elements with hyper-links, amongst 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*.

- **Animation** module contains elements and attributes that define a time based mechanism for composing the effects of animations. For example, this module can changes on XML or CSS attributes like color and dimensions.
- **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.
- **Layout** module contains elements and attributes for coloring and positioning media content, another layout mechanisms are also possible, such as CSS.

- **Linking** module contains elements and attributes for navigational hyperlinking. Navigation can be triggered by events or user interaction.
- **Media Object** module contains elements and attributes for referencing rendering behavior of external multimedia or control objects.
- **SmilText** module contains elements and attributes that defines and controls timed text. For example, this module could be used to create labels and captions.
- **Metainformation** module contains elements and attributes that allows describing the SMIL document. For example, this module could be used to define a movie details such as category, director, writers and cast.
- **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 defines the *body* element, where all temporal related module information is contained.
- **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*, respectively, they play child elements in sequence, one at a time and all at the same time.
- **Time Manipulations** module adds time behavior attributes to SMIL elements, such as speed, rate or time.
- **State** module defines attributes that defines the state of SMIL elements, such as element visibility, current element time, amount of repeated loops, playing state and many others.
- **Transitions** module defines attributes and elements for transitions across multiple SMIL elements according to *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.

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 are used to synchronize and animate eXtensible Hypertext Markup Language (XHTML) and Scalable Vector Graphics (SVG) elements.

In order to create a multimedia rich hyper-call, SMIL fits our goals, 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.

HTML 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 combined to HTML for improving the interaction and appearance of a web page.

CSS idea 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 a 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 amongst different web browsers, leading to different behaviors. To solve that problem, there are libraries written in *JavasSript*, namely *jQuery* which implements the same functionality for multiple browsers, solving most of incompatibility issues.

SmillingWeb [7] attempts to implement a cross platform multimedia player designed for SMIL 3.0 presentations with *JavaScript* and *jQuery* which, unlike [1], doesn't need to be installed and shouldn't have incompatibility issues.

SmillingWeb already takes advantage of HTML5 and CSS3, they take into account unsupported web browsers though *Modernizr*[16], 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 leading with SMIL changes due to real time communications.

Another problem with SmillingWeb is pre-loading and playing elements at the correct interval of time, which is not always possible due to low latency networks leading to experience pauses during playback.

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 to draw graphics with *JavaScript* on a rectangle within a web page.

SVG is an XML based format that incorporates the animation module of SMIL. Currently, SVG allows to add movement and animate attributes of elements. When embedded on HTML5, it allows dynamic changes to inner content in real-time through DOM API, besides that, it also allows to call *JavaScript* functions on events such as animation end, mouse over and mouse click.

Video and audio features are already possible with HTML5, like SVG it is also possible to bind *JavaScript* functions for different kinds of events over video and audio elements.

Back in 1995, *flash*[17] was developed for web-based animations, introducing video support in 2002, *flash* started to grow after that. Concurrency players, at that time, were focused on playing video and audio, *flash* had vector graphics and they were focused on streaming *on-demand* video across multiple platforms. *VP6* was their choice on video codecs, optimizing for half of video size for the

[16] `http://modernizr.com/`

[17] `http://www.adobe.com/products/flash.html`

same quality and providing adjusted video quality based on *Internet* connection latency.

Adobe Flash was the most widely used applications for reproducing live broadcast and recorded video [15], it supports progressive video download using HTTP and streaming using Real Time Messaging Protocol (RTMP).

RTMP is a TCP based protocol used to streaming audio, video and data between Flash Media Server (FMS) and flash player. A bidirectional connection is established between the two to in order allow real time communications. A flash player can stream a webcam video to FMS according to RTMP and another flash player can request a video stream to FMS that can be pre-recorded stream, live stream or data. Multiple FMS servers can be chained together in order 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) is possible to stream video between flash players, allowing a publisher breaking a stream into pieces that can be cooperatively distributed in a P2P mesh, RTMFP uses UDP to speed packet delivery, although is not reliable, it 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 are implementing some features of flash, it doesn't mean that flash will be replaced, instead of that both technologies can be used to develop rich *Internet* applications. It is also important to note that HTML5 is more compatible with mobile devices than adobe flash.

Like *Flash*, *Microsoft Silverlight*[18] 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 changing and Central Processing Unit (CPU) usage.

Using technologies that relies only on web standards, like CSS, HTML5, *JavaScript* and SVG, will make possible to raise communications to a new level. For example, with APIs like WebGL[19], 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.

In this project our goal is to enrich hyper-calls with no limits, every user should be free to choose how it wants to be contacted and it wants to share its contents.

[18] `http://www.microsoft.com/silverlight/`
[19] `http://khronos.org/webgl/`

In order to give users a personalized communication channel, each user must have a personal web page where its available plug-ins could be downloaded from other peers, after that, they can talk in the same language whatever it is.

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 couldn't be possible years ago because technology was limited or unavailable. Although today's technology is limited on some aspects, we are doing progress in order to improve the web ecosystem, by creating standards and migrating to newer technologies.

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 are recommended to limit local storage to at least five megabytes per origin.

If, for instance, one wants to rewind a real-time video, recordings will be needed from whom is streaming the video.

Real-time Transport Protocol (RTP)[27] is used for streaming audio and video over IP, the multimedia content is transported on the payload of RTP messages, RTP contains headers for payload identification. RTP is independent from its payload type, allowing 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 is the RTP profile for audio and video [26] which lists the payload encoding and compression algorithms. This profile also assigns a name to each encoding which may be used other protocols like SDP.

Another profile for RTP is defined by SRTP, which provides encryption, authentication and replay protection of 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 symmetric-key algorithm for data encryption. Each packet is encrypted using a distinct key-stream, otherwise using a single key-stream with AES on Cipher Block Chaining (CBC) would make 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's 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.

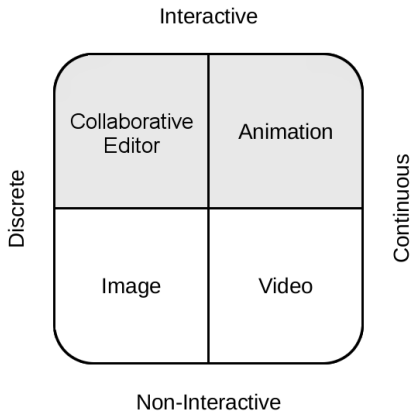


Fig. 4: Media Types

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 its type not depending from time, as continuous media depending from it. Interactive media is characterized by its state being changed by external events such as user interactions.

For example, an image is non-interactive and discrete, for instance, 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. 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 any kind of stream, a player for interactive stream is needed for downloading an environment, decoding the RTP payload to determine the state and display it to the user. Streaming interactive media like the combination of HTML, CSS and JavaScript require more than interpreting the code, a streamed user interface may contain an internal state that is not shown on code.

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 much 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.

[14] concluded that the ability to extract the objects state in order to synchronize them and the ability intercept events in order to control remote objects, are realized using the Model-View-Controller (MVC) concept. 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 MVC concept will make possible to implement an interactive RTP application that records, play, fast forward, fast rewind, stop and jump to random positions.

[14] also proposed an RTP profile for real-time transmission of interactive media. This new profile reuses much of video and audio profile implementation, integrating the interactive component. [9] explained how to record interactive video with this new profile.

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.

Jitsi Video Bridge [20] 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 need to download all the streams from *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* extension [10] to reserve channels for video transmission. Although this choice for signaling protocol, *Jitsi Video Bridge* also supports SIP.

[20] `http://jitsi.org/Projects/JitsiVideobridge`

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

Google Wave was a distributed collaboration platform based on *Jupiter*[17] that adopted OT techniques, other Google products such as Google Docs are 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[21] 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.

TogetherJS[22] 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.

Goodow[23] is a collaborative framework with its own server implementation, it supports four types of collaborative elements: String, Lists, Maps and Custom objects.

3 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:

- Enrich calls with any kink of multimedia in real-time.
- Ability perform tasks on real-time collaborative environment.
- Ability to record and playback interactive multimedia.

3.1 Modules

Our system is divided into five modules.

[21] `http://sharejs.org/`
[22] `http://togetherjs.com/`
[23] `http://realtimeplayground.goodow.com/`

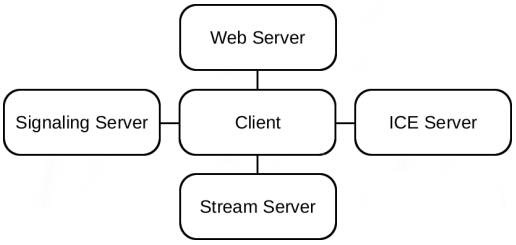


Fig. 5: System Modules

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. The web server will provide libraries to the client in order to interact with the other modules.

3.2 Implementation Proposal

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

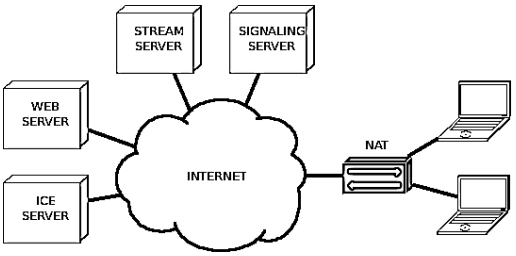


Fig. 6: System Architecture

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, *LAMP* server was considered, although the chosen technology for web development is *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 sake of simplicity and easy deployment the chosen platform chosen is the *Ejabberd*

XMPP application server since it implements [19] 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 cluster.

The streaming server will be *Jitsi Videobridge* as *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 share amongst 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.

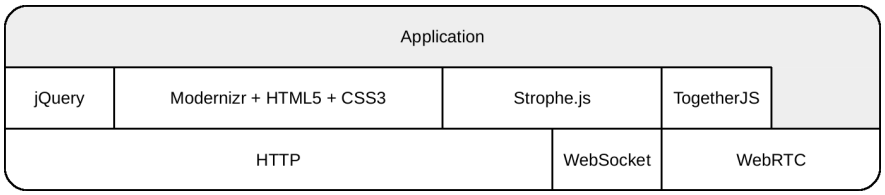


Fig. 7: App Architecture

Modernizr and *jQuery* will ensure that our application is compatible with the most popular web browsers.

Bootstrap will be used to make user interface more appellative and responsive. With *bootstrap* it's quite easy to develop applications that adapt to mobile devices with different screen sizes.

Strophe library will be essential to communicate with XMPP server and clients.

TogetherJS will be used for the collaborative component of our web application, other libraries were considered but, as said before, *TogetherJS* does not implement object storage making the object storage implementation free.

4 Methodology

4.1 Methodology Evaluation

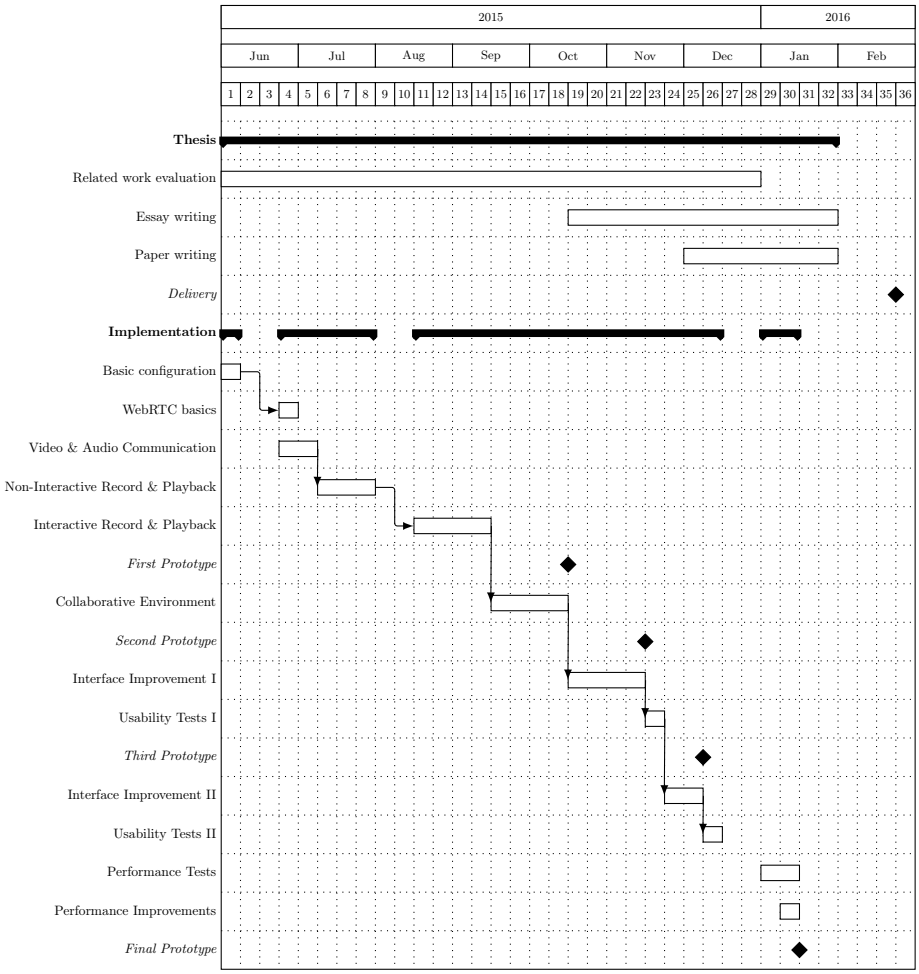
The most important task in order to implement our project is the infrastructure, without it our project wouldn't work. The basic infrastructure will be our first concern, namely the Web Server configuration, ICE and signaling servers.

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.

Briefly, our web application will be tested with real users and unit tests, the system in general will be validated with benchmarks such as analyzing the its behavior and delays by growing the amount of users and data.

4.2 Planned Schedule



5 Conclusions

5.1 Summary

Neste secção deve-se fazer o resumo do trabalho efectuado, retomando a ideia, as contribuições definidas e a forma como estas se materializaram

5.2 Conclusions

Nesta secção devem ser retiradas conclusões do trabalho realizado, em face dos resultados obtidos [**?**].

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. Fette, I., Melnikov, A.: The WebSocket Protocol. RFC 6455 (Proposed Standard) (Dec 2011), http://www.ietf.org/rfc/rfc6455.txt
7. 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
8. Handley, M., Jacobson, V., Perkins, C.: SDP: Session Description Protocol. RFC 4566 (Proposed Standard) (Jul 2006), http://www.ietf.org/rfc/rfc4566.txt
9. 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
10. Ivov, E., Marinov, L., Hancke, P.: COnferences with LIghtweight BRIdging (COLIBRI). XEP-0340 (Experimental) (Jan 2014), http://www.xmpp.org/extensions/xep-0340.html
11. 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)

12. 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
13. Martin, G.: Hypertext to Hypervoice - Linking what we say and what we do p. 6 (2012)
14. 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)
15. 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)
16. 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
17. 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
18. Paper, A.I.U.W.: Next Generation Internet : IPv4 Address Exhaustion , Mitigation Strategies and Implications for the U. S. pp. 1–26 (2009)
19. 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
20. 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
21. 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
22. 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
23. Saint-Andre, P.: Multi-User Chat. XEP-0045 (Draft) (Feb 2012), http://www.xmpp.org/extensions/xep-0045.html
24. 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)
25. Sawhney, N., Balcom, D., Smith, I.: Authoring and navigating video in space and time. MultiMedia, IEEE 4(4), 30–39 (Oct 1997)
26. 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
27. 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
28. 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)

29. 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
30. 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)