Hyper-linked Communications: WebRTC enabled asynchronous collaboration

Henrique Rocha
Instituto Superior Técnico
Av. Prof. Dr. Anibal Cavaco Silva
2744-016 Porto Salvo, Portugal
henrique.rocha@tecnico.ulisboa.pt

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

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

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

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

1. Introduction

1.1. Background

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

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

Furthermore, real time communication applications can make a significant difference on business, education and health sectors by providing tools for developing teaching and learning online, teamworking and socializing web applications.

1.2. Proposed Solution

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

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

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

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

1.3. Thesis Contribution

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

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

According to Martin Geddes, the quality of the interaction worsens as the number of users increase[5]. In our testing phases we will quantify and qualify the impact of increasing users on the interface and performance of our prototype.

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

1.4. Outline

This rest of this document is structured as follows:

- **Chapter 2** describes the previous work in the field.
- **Chapter 3** describes the system requirements and the architecture for an Web Application that fulfills the goals of this thesis.
- **Chapter 4** describes the implementation of our Web Application and the technologies chosen.
- **Chapter 5** presents the evaluation tests performed and the corresponding results.
- **Chapter 6** summarizes the work developed and proposes future work.

2. Related Work

One way to overcome the Internet Protocol Version 4 (IPv4) address scarcity problem [6] 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)[8].

NATs weaknesses are being exposing at the application layer, namely impacting applications that require direct communications between two private networks.

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

Due to the use of NAT, bi-directional communications between clients have different needs from request-response based communications between clients and servers. This lead to the appearance of mechanisms such as Session Traversal Utilities for NAT (STUN), Traversal Using Relays around NAT (TURN) and Interactive Connectivity Establishment (ICE), in order to bypass the limits imposed by NAT.

STUN, TURN and ICE [4] servers are a possible solution to overcome NAT's exposed weaknesses to applications that require communications between two private networks.

When connection is established, WebRTC came to simplify how audio and video are transmitted through web browsers. 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.

WebRTC uses Session Description Protocol (SDP) [3] 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.

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.

However, WebRTC by itself does not define how users get to know each other nor how information flows between users. For this reason, we have studied the multiple ways we could implement this *get-to-know* mechanism which is known as signaling protocol.

Signaling is the process by which applications exchange connection information about peers and servers, their capabilities and meta-data. In particular, WebRTC does not implement signaling, as different applications may require different protocols and there is no single answer that fits all problems. As a consequence, multiple options are available for filling the missing WebRTC's signaling component, which can be performed using Session Initiation Protocol (SIP)[7], Extensible Messaging and Presence Protocol (XMPP), *WebSockets*, *Socket.io*[1], Signaling-On-the-fly (SigOfly)[1] or by implementing a custom protocol.

One of WebRTC signaling's requisites is bi-directional communication. The *WebSocket* protocol allows bi-directional communications over a full-duplex socket channel [2], by other words it supports sending and receiving data simultaneously.

With the communications establishment issue solved, we had to discuss the different types of media and what can be done with each kind in order to increase the value of communications among users. In this context, we have studied solutions and libraries that allow us to implement our prototype with time manipulation features, collaborative text edition, record and playback interactive video.

Hypermedia concept brings the possibility to organize and overlay multimedia elements into a nonlinear linear structure holding the promise of future technology and features. Languages such as Synchronized Multimedia Integration Language (SMIL) [11], *HyVAL* [11] and HTML can be used to implement the *Hypermedia* concept. Two examples of applications that captured our attention were *HyperCafe* [9] and *Hyper-Hitchcock* [10] which explored interactive video features.

Using technologies that relies only on web standards, like CSS, HTML5, *JavaScript* and Scalable Vector Graphics (SVG), will make possible to develop an application that applies the hypermedia concept with the advantage of being compatible with a greater amount of web browsers.

2.1. 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.

Our first concern on real time collaboration applications is the data storage and representation. Storing

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

multimedia content on a web client is not a viable solution because the local storage is limited to at most five megabytes per origin. Additional servers will be required to process and record the large amount of data generated by audio and video streaming.

Kurento Media Server (KMS) supports streaming over *WebRTC*. Another important component of KMS is *Kurento Repository*, which supports recording and playing directly from *MongoDB*. That is important for providing a scalable media storage.

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

Among mutiple OT platforms and libraries we present *ShareJS*[2], *TogetherJS*[3], *Goodow*[4], *Etherpad Lite*[5] and *otJS*[6].

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

3. 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 features in real time.

3.1. Requirements

In a general way our system's goal is to provide a multi-party video and audio conference environment that supports chat, time manipulation, collaborative text edition and hyper-content creation.

For our system, our application must provide: a simple way to send instant text messages to the conference participants, ability to recording and playback recorded video including all the hyper-content displayed at that time, mixing multiple user streams into a single stream, create annotations associated to a specified time, allow users to superimpose hyper-content to video given a range of time, search every objects related to a conference room such as hyper-content, time annotations and users, share files and time links among users, sound detection for showing the current speaker, provide a collaborative text edition tool, interpret QR codes in order to ease content creation and support database replication.

Moreover we allow clients to discover chat rooms and other clients by navigating on the web pages provided by our web server. In addition users can create rooms for multi-party audio and video communication communication which is achieved by using WebRTC's *PeerConnection*.

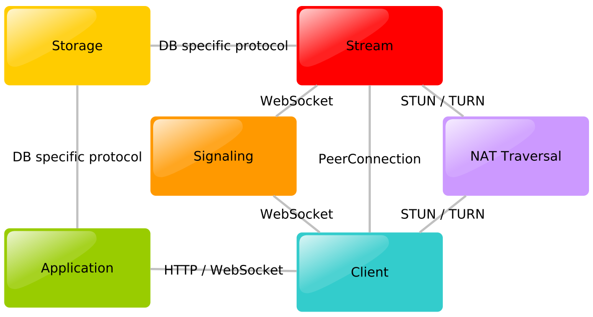

Figure 1. System Modules

3.2. Modules

In this section we present the several modules that were designed in order com fulfill the set requirements. Figure 1 presents the structure of our system which was divided into six modules:

- **Application module** - responsible for providing information about the relevant modules (*NAT Traversal* and *Signaling*) and user interface to the *Client* in the form of web pages in HTML and *JavaScript* libraries through Hypertext Transfer Protocol (HTTP).
- **Signaling module** - responsible for *Client* and *Stream* coordination which will be performed using *WebSockets*.
- **NAT Traversal module** - STUN and TURN techniques used by *Client* and *Stream* modules during the *Signaling* phase which ends by establishing the connection between them.
- **Stream module** - responsible to deliver and receive multimedia content from the *Client* using WebRTC.
- **Storage module** - provides two main functionalities: store the model information and media recorded. This is the single module responsible for persistent storage. It stores user and communication data as well as all the data required among user sessions. It is also use to store all the communication streams, so that they can be viewed later.
- **Client module** - responsible for the interaction with the user.

3.3. Implementation Proposal

The infrastructure is composed by: web server, stream server, signaling server, database and video repository.

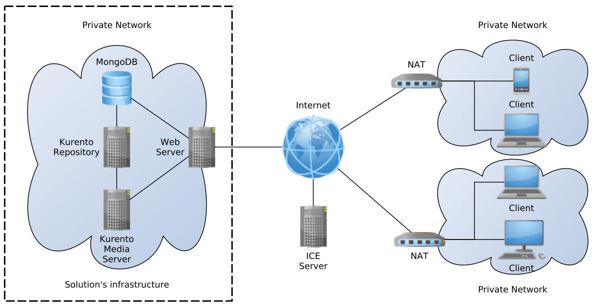

Figure 2. System Infrastructure

2. http://sharejs.org/(accessed June 2, 2015).
3. http://togetherjs.com/(accessed June 2, 2015).
4. http://realtimeplayground.goodow.com/(accessed June 2, 2015).
5. https://github.com/ether/etherpad-lite(Accessed 20 March 2016)
6. http://operational-transformation.github.io(accessed March 10, 2016)

In order to simplify our solution, we propose that the *Application*, *Signaling*, *Stream* and *Storage* modules be implemented within the same server application, so it could be easy to deploy as a single image. To this set of modules, we often call *backend*.

3.3.1. Security and authorization. Having established that the *backend* modules are placed in the same machine, that helps controlling which resources the client has permission to access as those modules are seen as a private network.

To qualify the above we provide public access to HTTP server ports, maintaining the access to other components restricted through firewall rules. In relation to the database there is no directly access from the outside. All the database information is accessed via our application server which validates the permissions of users on our system.

On the other hand, the access to our streaming servers is also restricted, but clients can connect to them after concluding the signaling phase. This signaling phase may or not proceed in function of the client access permissions. For example, if a user is trying to access a private conference room that he is not a member of nor has an invitation link for, the signaling server refuses to start the signaling phase and the user cannot access the streaming server.

The placement of our streaming servers inside a NAT also has an important role with respect to external misuse prevention. Otherwise, placing our streaming servers could allow external clients to perform their own signaling protocol and, as a consequence, use our infrastructure without our consent.

3.3.2. Client connections. Although the delegation of processing work to clients can improve our system's scalability, we are concerned about using the least resources possible on the client side, as huge resource consumption may drain battery very fast or may even be impossible to run on mobile devices. We are aware that streaming video from clients is already a very intensive task which we cannot avoid but can improve by delegating the most intensive tasks to our servers.

In this context, with a centralized approach, each client must only have one *PeerConnection* to our streaming server and content shown to them is changed on demand either being it an individual or a composite view. Otherwise clients could follow a peer-to-peer connection which would result on maintaining more connections and performing the composition of videos on client side.

The composition of streams on client side is performed by receiving streams with the best quality possible but, due to undersizing the video of clients into a smaller region, this would result on wasting bandwidth on a quality that is not needed.

Although the peer-to-peer approach could be used on our system, we conclude that we need to record the video on our streaming server because web clients have a very limited storage and peer disconnections may result on recorded video loss.

The same can be concluded to instant message delivery, each client must have only one *WebSocket* connection to the application server which consequently relays the messages to other users.

Relaying instant messages from clients through the our web servers is easier if all clients are connected to the same server because all messages can be directly delivered without sending messages across multiple web servers.

In the context of this thesis, we will not implement sending messages across web servers but, in order to allow our system to scale, we will consider that all conference participants are connected to the same server and our system is scaled by having conference rooms distributed across different servers.

3.3.3. Software choices. Furthermore, we have taken into account the compatibility between the streaming server, database and the operation transformation solutions, in order choose the appropriate framework to implement our web server.

We have decided that our solution must use KMS. Our web server could be implemented easily with *NodeJS* or *Java* due to the fact KMS provides clients for both technologies. But others could also be used as KMS also exposes their API via *WebSockets*.

Due to the fact we are going to use KMS as streaming server solution, we could choose *NodeJS* or any *Java* based web framework for implementing our web application server. We have decided to implement our web server with the *PlayFramework*[7] using *Java* because of our previous experience with it.

By default, *Kurento Repository* is implemented over *MongoDB*, for convenience our storage model will also use the same database.

Importantly, for the collaborative text editor, we have chosen *OT.js* due to its server and storage implementation choice independence.

For the *NAT Traversal* module a public STUN server can be used for testing our solution. Nevertheless, we recognize that for a production environment we would need to maintain our own TURN servers in order to ensure connectivity to all clients.

Not less important, on the client computers, both *Mozilla Firefox* and *Google Chrome* could be installed as web browsers. As such, both should be supported. Libraries such as *jQuery*, *Bootstrap*, *Adapter.js*, *OT.js* can be downloaded from the web server and executed on the client side using any of these two browsers.

TABLE 1. APPLICATION ARCHITECTURE

Application						
jQuery	HTML5	CSS3 (Bootstrap)	Signaling	ot.js		adapter.js
HTTP	User Interface		WebSocket			WebRTC

Table 1 presents the application architecture and the underlying technologies seen from the user's perspective. *Adapter.js* 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.

With respect to displaying content, the synchronization between multimedia elements will be performed

7. https://www.playframework.com/(acessed March 25, 2016)

through chains of *JavasSript* events or by specifying the interval of time which time content must be visible. Other animations can be implemented with SVG embedded on HTML.

4. Implementation

4.1. Data Model

The data model is a critical component of our solution, as a badly designed model can imply serious difficulties when implementing new features that are not part of the plans. During the course of this project, we had to redesign the model more than once in order to support new features.

In order to offer all the functionalities that we promise, some information about objects must be persistent such as users, groups,relations among users, group memberships, messages, hyper-content, recordings and collaborative editor state. For designing our model we have taken into account generic programming techniques. We observed that operations like searching for an object were quite repeated across different types of objects.

4.2. Signaling Protocol

Although we have mentioned that the signaling protocol is used to establish connections between peers, on our system our media server (KMS) is a peer that receives video streams and sends to its connected clients.

After the web application server validates the user access, the signaling protocol allows the users to directly connect to the KMS, which is placed in a private network, and lets the application server and users negotiate media types and encoding information to use during the conversation.

Our signaling protocol consists of sending and receiving JavaScript Object Notation (JSON) formated messages over *WebSockets* by both the application server and the client.

When a user enters a group conference, after the page is completely loaded, a WebSocket is created to maintain a connection with our web servers. But 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-agent (web browser) through the HTTP headers. The Web application server retrieves all the information needed from the database in order to check if the user has permissions to join that conference room. It is important to save the user identification before the *WebSocket* connection is created because, after the handshake is performed by the *WebSocket* protocol[2], the HTTP context is lost.

At this stage, the web application's user is asked if he wants to share its camera and microphone, share screen or just receive streams from the server.

If the user decides to share his camera or screen the user agent creates an offer, sets a *local session description* to its *PeerConnection* and sends it through the WebSocket to the Application Server.

The server receives 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 tries to gather ICE candidates.

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

Subsequently, 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.

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. When a connection is established, the user and server start to interchange stream data but other ICE candidates may arrive with better connections.

Having 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.

4.3. Stream Recording

Initially we experimented with local recording and synchronizing with our servers. Even thought it got it to work, it consumed to much bandwidth. We then tried recording on server side.

One of our concerns during the development of our solution was the storage scalability. Saving files directly into the file system would require an extra effort to distribute and replicate files among servers. For that purpose, the *Kurento* team developed *Kurento Repository*[8] which is based on *MongoDB*.

With server side recording, the user would maintain always the same stream URL even if it is playing real time video or reproducing recorded video. It is KMS that sends different content through that stream. When a user desires to play recorded video, a *webSocket* message is sent specifying the time and the intended user id. The server performs the calculations in order to find a block that intersects the requested time, plays it and when finished, the next part is automatically played without the user intervention.

4.4. Hyper-Content

Our system supports creating superimposed content to video, which is achieved by creating HTML tags on top of the video with the same size. The decision of which content must be displayed to each user is performed by our content scheduler which uses the user's current time in order to synchronize which content is shown or removed from the user interface.

In order to create content, the user has the option to write simple movie captions without writing any code, otherwise, as mentioned before, it can write HTML, CSS and *JavaScript*.

As manually content insertion is a laborious task that can be realized after the video is recorder, we provide an alternative mechanism for real time introduction of superimposed content. In order to help the content creator

8. http://doc-kurento-repository.readthedocs.org (accessed on 17 March 2016)

to introduce and synchronize its content in real time, we allow the user to encode its content into QR codes and show it to the camera in real time.

4.5. Collaborative editor

Our collaborative editor is a simple text editor that is synchronized with all participants within a conference room implemented using *ot.js*.

The state of our collaborative editor is not saved on the database every time it changes. Instead, the users just synchronize the editor content among themselves using the application server to relay editor changes and save on demand.

5. Evaluation

5.1. Tests Objectives

We have tested our solution with real users for a better understanding of their difficulties and what can be done in order to improve our solution's usability.

We have also tested the performance of our solution by measuring the used resources. Those performance tests are crucial to ensure that our solution is in fact stable and users can use it endlessly without decreasing the quality of their experience.

5.2. Performance Tests

In order to benchmark our system, we have implemented a small *Python* script using *psutil*[9] that collects with a periodicity of one second: CPU, physical memory information relative to each running process and network usage relative to each interface.

The performance test scenario that we have defined consists on two phases, the first phase consists only on having users, with similar computer and network specifications, entering sequentially on the conference room. The second phase consists on the users leaving the conference room. Each event, joining and leaving, occurs with intervals of one minute in total of thirteen minutes (780 seconds).

From the media server perspective if there are n clients connected, each of them sending and receiving one stream, it is expected that server sends and receives also n streams. As such, we expect that the amount of network traffic increases linearly as users join a conference room. Figure 3 confirms our expectations. Each vertical yellow line represents one event: the first seven events are users entering the conference room, the next ones represent users leaving the conversation.

The blue peaks are caused by the signaling phase and web page downloads, including resources such as images, stylesheets and javascript files. The green peaks are caused by video and audio being transfered between KMS and *MongoDB* through *Kurento Repository*. Each peak occurs every time a block of video is recorded, which in this case is every ten seconds. The recordings are synchronized so all user and mixed blocks start and end at the same time. That is why the amount of work done every ten seconds accumulates, and because

9. https://github.com/giampaolo/psutil (Accessed March 27, 2016)

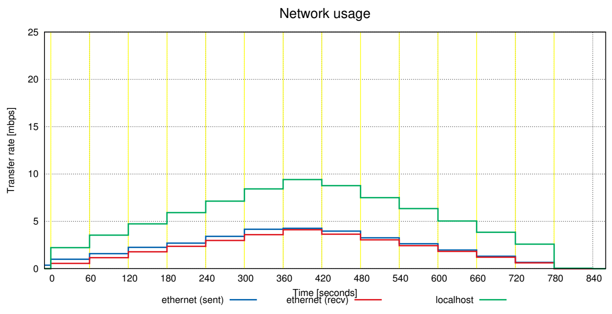

Figure 3. Network usage after implementing all features

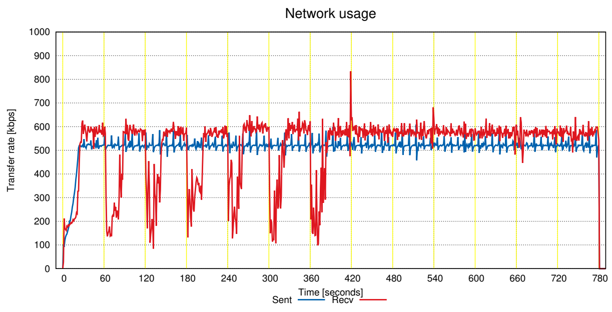

Figure 4. Client Network usage during our test case

this is performed locally, the maximum transfer rate is limited by the performance of the memory as buffers are written to buffers then to disks. Sent data transfer rate has no significant peaks as HTTP requests and signaling information contains little information.

With this results we conclude that if we want to scale our storage solution using the *MongoDB*'s cluster configuration, both *Kurento Repository* and KMS should be installed on the same machine because the loopback interface can handle bigger transfer rates than the remaining network interfaces. Installing the repository on the same machine as a database node does not ensure that recorded videos are stored in the same machine, for this reason we would prefer installing KMS and *Kurento Repository* on the same machine.

On the other hand, Figure 4 shows the respective network usage on the client side during our test case. We observe that in the first seconds the client adjusts the video quality it sends to KMS. Whenever a new client enters the conference, we observe that KMS decreases the video quality in order to instantaneously integrate a new user into the conference room. After a while, KMS realizes that the network can handle the increase of clients and sends the video with a better quality to every participant. When a user leaves the conference room KMS has no need to decrease the participant's video quality as less network bandwidth will be used.

Memory usage

Figure 5 shows the memory usage during our performance test. Both Java virtual machine (JVM), *MongoDB* and KMS performs their own memory management by holding and recycling objects when needed. The expected and observed behavior of the memory usage is growth of memory usage while the users are entering the conference room and a memory usage stabilization afterwards.

MongoDB memory usage keeps increasing because it tries to fit part of the database on Random-access memory (RAM) for fast read access. *MongoDB* checkpoints data

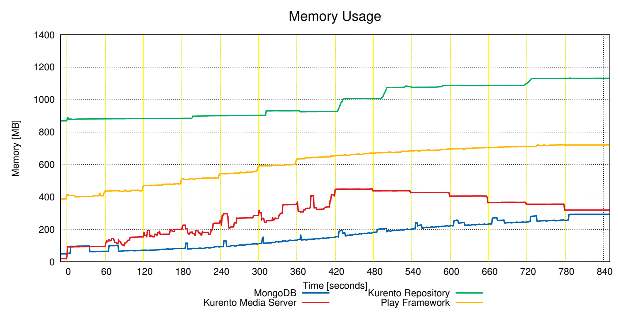

Figure 5. Memory usage after fixing recorder memory leak

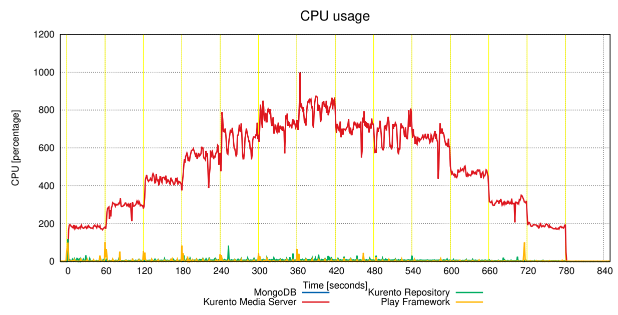

Figure 6. Percentage of CPU used during the performance tests

to disk every 60 seconds or when journal data exceeds 2GB[10], that explains the small memory usage peaks during our test case. When the conference room is empty there are no video recordings, which explains the memory stabilization at the end.

KMS memory management released resources as soon as users left the room.

CPU usage

Figure 6 shows the percentage of Central Processing Unit (CPU) usage during our performance test case. Each 100% represents one CPU core, although that does not mean one CPU is fully used, for example two cores at 60% represent 120% CPU usage. As we can see, the percentage of CPU used increases and decreases linearly in function of the amount of conference participants. KMS is responsible for most CPU usage.

Just for testing purposes, we performed the same performance tests disabling QR codes detection. We concluded that QR code detection is a very intensive task, approximately doubling the amount of work performed by the CPUs.

Even though, with this test results, we conclude that our solution's bottleneck is the CPU usage at KMS.

5.3. Usability Tests

In this section we describe usability test scenarios that we have applied and their respective results.

5.3.1. Tests Scenarios. In order to evaluate the usability of our solution, we have performed usability tests with the help of real users with different backgrounds and ages.

We handed a guide to the users with five tasks to perform. The metrics we used for each task were: number of clicks, number of errors (including a description) and time spent.

10. https://docs.mongodb.org/manual/faq/storage/(Accessed March 28, 2016)

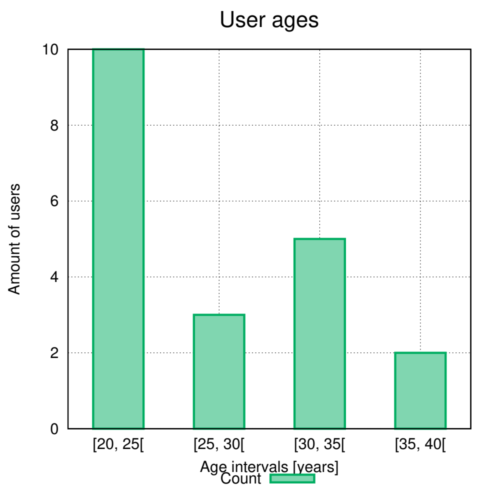

Figure 7. Ages of the users that tested our system

5.3.2. Test Results. In this section we present the results of our usability tests. The first time we tested our solution by providing tasks to users, we have observed that our solution was not perfect. Having faced usability problems during our tests, he had to improve our solution's usability and start the tests again.

In the first testing phase we have performed tests with just three users and ceased for improvements. We gathered comments and suggestions after letting the users explore our system.

On the second phase of our user interface tests, in a general way, we have noticed great improvements on the learning time.

In order to measure the users learning speed, we have performed tests with experienced users in order to retrieve the optimal task duration and minimal task clicks.

To this end, with regard to optimal task duration and minimal clicks, we obtained the values shown in Table 2.

TABLE 2. METRICS FOR AN EXPERIENCED USER

Task	1	2	3	4	5
Duration (seconds)	20	35	30	25	35

From the data collected with twenty tests with users, namely the task duration (Figure 8) and task difficulty (Figure 9), we have calculated the confidence intervals in order to understand the most plausible values for each metric.

The youngest and oldest testers were, receptively, twenty two and thirty eight years old. Figure 7 the ages of the users that tested our system.

As a result of both true average and variance being unknown and the usage of a relatively small amount of samples, we had to use *t-distribution* to estimate our metrics confidence intervals. We have used a 95% confidence level.

According to Figure 9, we can observe that most users had less difficulties with the first two tasks, which represents types of tasks that most users are familiar with. As soon as users had to navigate in time, manipulate annotations and create content (respectively *task3*, *task4* and *task5*) we observed that they revealed more difficulties. Most of those difficulties, based on the users feedback,

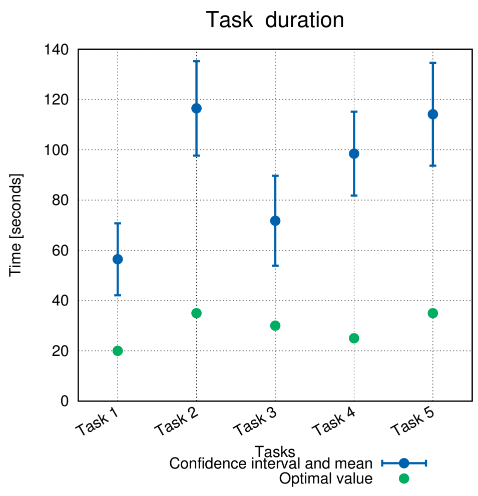

Figure 8. Time spent per task

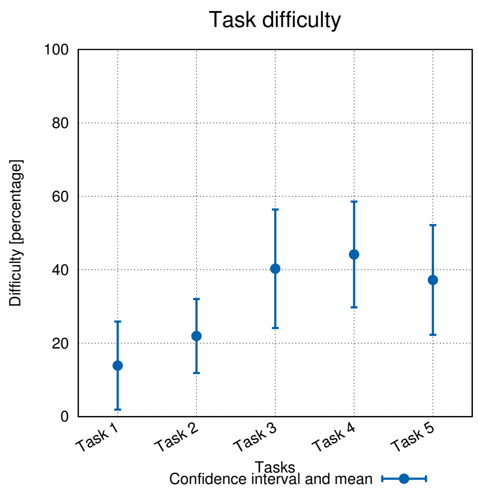

Figure 9. Difficulty per task

were mainly due to those concepts not being familiar to them.

As we had relatively bad results with some users, we explained to those users that could not conclude the tasks or performed them incorrectly, the most efficient way to perform the requested tasks. Some users suggested to display more hints in order to achieve a faster learning, but afterwards all were impressed and gave us a better evaluation.

Most users gave us worse evaluations on our user interface layout and content editor, which was due to having a lot of tools present in the same web page and some of them being hidden due the screen size. In some cases users had to scroll down in order to find the tools they were looking for.

Another weak aspect was our content editor, which, in fact, we recognize is difficult to work with, mostly due to the amount of information that is necessary to create a synchronized content (starting time, duration and content itself). Some users have suggested that the content should also be present on the timeline so they could be easily dragged and resized (on time).

We are aware that placing content on the timeline will reduce our solutions performance, especially when there is a relatively large amount of content, due to the content that is present on the timeline being loaded all at once. Although, we recognize that for some cases (relatively low amount of content), displaying the content on the timeline could not have a great impact on our solutions performance, we choose not to implement this.

In conclusion, 100% of our testers though that our solution was an innovation and 95% recommended using our solution.

6. Conclusions

6.1. Achievements

We have successfully implemented the basic functionalities of our prototype and spare some time for adding more valuable features such as the ability to create content by exposing QR codes to the camera and lastly perform changes to the user interface in order to improve the quality of user's experience.

The performance tests that we have executed showed that our system is stable and more importantly that our web server is lightweight and most of processing power is dedicated to the streaming server.

Our usability tests show results that are considerably worse than the established optimal values due to our solution propose a different way to communicate that most people are not used to. Although we have obtained those results, in general our users gave us positive feedback and valuable advices which we have used to improve our system.

6.2. Future Work

Playing back video with a faster rate is not possible using the current version of KMS. Even though we expect the availability of that feature in a near future but we have proposed an alternative way to implement faster playback by using *ffmpeg* to convert the video before playing it.

Although we have tested our solution in a powerful machine for the current time, our performance tests revealed that the streaming component uses a lot of resources. We left for a future work a deep analysis on the scalability of our system for which we have proposed different approaches but we have not tested them.

Another aspect we could have tested was the performance of our solution when using TURN servers for relaying the traffic that fails using STUN.

Lastly, we have chosen functionality over security in respect to displaying content to users which lead to security flaws on our solution. Although we have not solved the security problems, we have proposed a solution which at the same time limits the flexibility of adding new functionalities to our prototype.

References

[1] P. Chainho et al. "Signalling-On-the-fly: SigOfly". In: *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on.* 2015, pp. 1–8. DOI: 10.1109/ICIN.2015.7073799.

[2] I. Fette and A. Melnikov. *The WebSocket Protocol.* RFC 6455 (Proposed Standard). Internet Engineering Task Force, Dec. 2011. URL: http://www.ietf.org/rfc/rfc6455.txt.

[3] M. Handley, V. Jacobson, and C. Perkins. *SDP: Session Description Protocol*. RFC 4566 (Proposed Standard). Internet Engineering Task Force, July 2006. URL: http://www.ietf.org/rfc/rfc4566.txt.

[4] Ying-Dar Lin et al. "How NAT-compatible are VoIP applications?" In: *Communications Magazine, IEEE* 48.12 (2010), pp. 58–65. ISSN: 0163-6804. DOI: 10.1109/MCOM.2010.5673073.

[5] Geddes Martin. "Hypertext to Hypervoice - Linking what we say and what we do". In: (2012), p. 6.

[6] An IEEE-USA White Paper. "Next Generation Internet : IPv4 Address Exhaustion , Mitigation Strategies and Implications for the U. S." In: (2009), pp. 1–26.

[7] J. Rosenberg et al. *SIP: Session Initiation Protocol*. RFC 3261 (Proposed Standard). Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141. Internet Engineering Task Force, June 2002. URL: http://www.ietf.org/rfc/rfc3261.txt.

[8] J. Rosenberg et al. *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. RFC 3489 (Proposed Standard). Obsoleted by RFC 5389. Internet Engineering Task Force, Mar. 2003. URL: http://www.ietf.org/rfc/rfc3489.txt.

[9] N. Sawhney, D. Balcom, and I. Smith. "Authoring and navigating video in space and time". In: *MultiMedia, IEEE* 4.4 (1997), pp. 30–39. ISSN: 1070-986X. DOI: 10.1109/93.641877.

[10] F. Shipman, A. Girgensohn, and L. Wilcox. "Creating navigable multi-level video summaries". In: *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*. Vol. 2. 2003, II–753–6 vol.2. DOI: 10.1109/ICME.2003.1221726.

[11] T.T. Zhou and J.S. Jin. "A Structured Document Model for Authoring Video-Based Hypermedia". In: *Multimedia Modelling Conference, 2005. MMM 2005. Proceedings of the 11th International*. 2005, pp. 421–426. DOI: 10.1109/MMMC.2005.15.