Hub for Digital Communication

João Reis

Instituto Superior Técnico

Abstract

Many organizations use a POTS system in conjunction with a VoIP, this creates an availability problem to system because the repair and replacement of the hardware that keeps the POTS system working is very hard due to lack of spare parts on the market. Given this problem this work has the objective of offering a backup system that allows the organization that implements it to keep functioning as normal and at the same time offering extra features working as an extension of the existing system.

This problem can be solved using a system that integrates the new webRTC technology with the existing VoIP technologies offering the future users of the system a Web application that allows them to do calls without the need of any native code.

The system is composed of a Web server, an Asterisk PBX and an IVR server, the Web server is used to deliver a WebApp, signaling server for WebRTC browsers and to set up users inside the Asterisk server. The Asterisk PBX is used to connect the WebApp to the already existing SIP infrastructure and the IVR server serves dynamic IVR menus to the users, the menus are written using NodeJS which makes them easy to integrate with the webservices.

With this system the users can do all the operations that a real pone allows inside a browser, search other users by name, review messages before sending and see all voicemail inside the browser.

Keywords: Voice over IP, WebRTC, Asterisk, NodeJS, Web development

1 Introduction

1.1 Motivation

With the expansion of the internet and the surging of VoIP (Voice over IP) technology the old analog and digital networks quickly became obsolete. Their incapacity to compete against the quality and convenience of the new technology as the new services that it offers, because of this incapacity to compete POTS networks started to be slowly replaced by the new VoIP networks.

Unfortunately, the substitution of a POTS network is a very expensive process that involves acquisition of new phone terminals, new cables, routers and switches. To cut on costs several companies and organizations kept their POTS networks working in parallel with their new VoIP one using special hardware to connect them.

1.2 Problem

Nowadays a lot of organizations keep an old POTS (Plain Old Telephone Service) network and newer VoIP network. This network normally has at its core a PBX (Private Branch Exchange) server, the POTS network its connected to the PBX with the help of an electronic central that's able to convert the old telephone signal into the new format allowing communication between the two networks as seen in Figure 1.

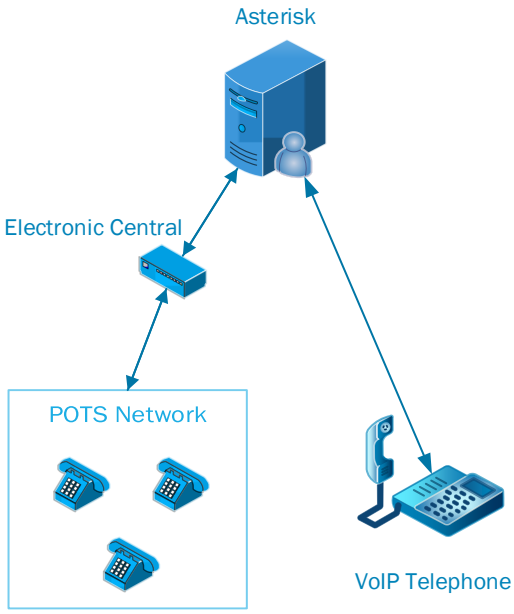

Figure 1 - Network POTS plus VoIP

Unfortunately, the electronic central were only meant to be used during the first stages of VoIP deployment and so not many were produced making it very hard service one in 2018. This

creates a problem to all organizations that depend on one since if it breaks the availability of the service will be affected until a way to repair the electronic central is found or a new network is installed.

1.3 Solution

A way to solve this problem is to develop backup system using existing networks and devices to make it cheap and easy to deploy. With current technology using WebRTC is a good option, it can be used from a desktop computer, a tablet or a smartphone using the LAN (Local Area Network) of the organization. Users using this system can be authenticated using the OAuth framework allowing also the WebRTC app to obtain data to identify the user. Using OAuth gives the application all it needs to run without asking the user for any information making the system extremely easy to setup requiring only a simple login.

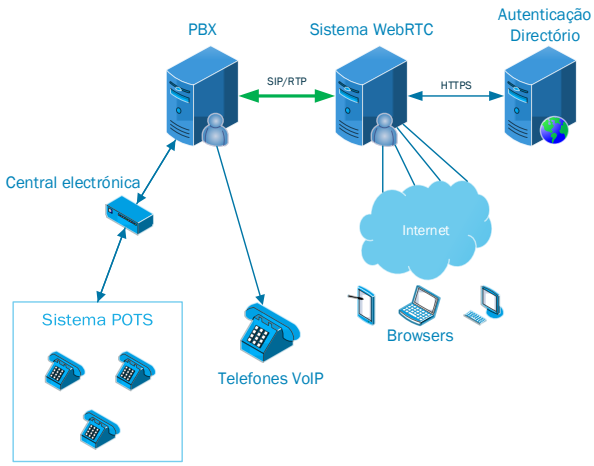

Figure 2 - Original network with the backup system

The backup system, that can be seen in Figure 2, would allow users to communicate between themselves directly using a pure WebRTC implementation in which users send media directly to each other without the need for a proxy. In the case of contacts between WebRTC system users and VoIP/POTS it will be required the use of a proxy that in this case would be the PBX server.

2 Related work

2.1 POTS

POTS or Plain Old Telephone Service are the old telephone systems that used a cooper loop to transmit its signals. This system needs a dedicated physical link for each call making it very inefficient and expensive. It was able to acquire and transmit frequencies between 300Hz and 3300Hz a very small window of the audible sounds that vary between 20Hz and 20000Hz. Although very a very inefficient and low-quality system compared to the newer ones it was the only one available for a long time which lead to a large adoption all over the world.

2.2 SIP/SDP

SIP (Session Initiation Protocol [1]) it's a text-based signaling and control protocol for real-time communications. Its normally used in VoIP applications to start, manage and terminate a call.

SIP is often used with SDP (Session Description Protocol [2]), this protocol is send inside the body of SIP message at the start of a call and is used to negotiate codecs bitrates and other relevant session parameters. SDP is also a text-based protocol that can be used independently from SIP if desired.

2.3 VoIP

VoIP (Voice over IP) is the transmission of real-time audio or video from a call through a network that uses the internet protocol (IP). To achieve VoIP a lot of technologies can be used, in the case of terminals we can use softphones on a computer, a browser using WebRTC or compatible telephone. The transmission over the IP network his made using an RTP [3] or SRTP [4] protocol over the transport protocol that is normally UDP (User Datagram Protocol).

2.4 WebRTC

The WebRTC is composed of three API's that allow access to the devices, create a direct connection between end-users and creation of a data channel. WebRTC employs noise and echo suppression algorithms to enhance the quality of the sound captured by the devices that are being used.

WebRTC handles NAT transversal using the ICE (Interactive Connection Establishment [5]) protocol being able to transverse any NAT if a STUN (Session Traversal Utilities for NAT [6]) and TURN (Transversal Using Relays around NAT) servers are used. A STUN server is used by an endpoint to learn its own public Ip address and works as shown in the Figure 3.

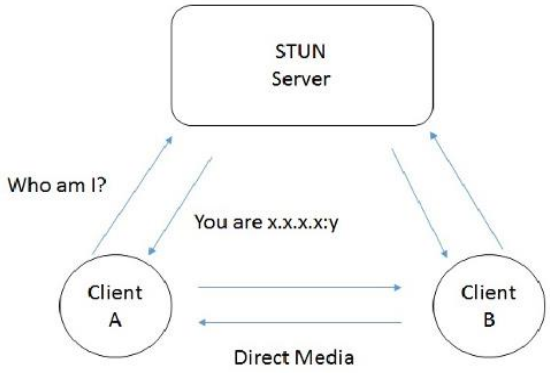

Figure 3 - STUN server [7]

A STUN server is just a tool used to generate the ICE candidates used by ICE protocol and has no part in the media exchange. The ICE candidates gathered by an endpoint must be sent to the other endpoint using a signaling server. An ICE candidate represents a transport address in which an endpoint is willing to receive a stream from the peer.

In the cases in which both users are employing symmetric NAT's a TURN server is required and in this case the TURN server takes part in the media flow as shown in the figure below.

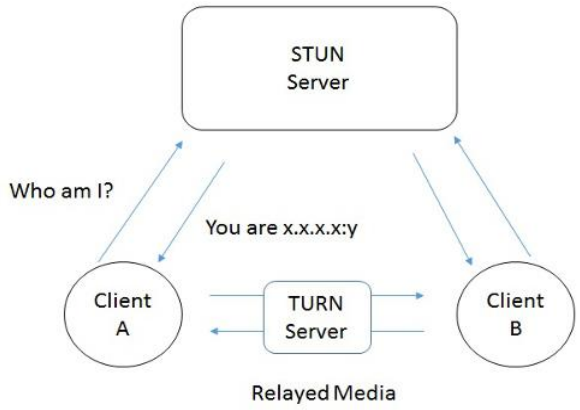

Figure 4 - TURN server [7]

In the scheme of Figure 4 the TURN server receives the traffic from client A and redirects it to client B, this works because clients A and B started the communication with the server. The port that is referred in the packet received is then used to relay the stream that is being sent by the peer. Using a TURN server is very expensive and most only be used when the is no other alternative.

WebRTC implements a new extension to the ICE protocol called Trickle ICE [8], this new extension allows the ICE candidates to be sent to the peer as they are discovered. It also allows the peer to test them as they are being received and to start streaming as soon as a working one is found. This implementation vastly reduces the setup time for the call creating a better user experience.

3 System

3.1 System requirements

The system must implement the following requirements to fully reach the goal of integrating the POTS/VoIP technology with the Web.

- Authentication using the OAuth2.0 framework [9];
- Communicate between WebRTC, VoIP and POTS
- Grant the confidentiality of all web communications;
- Voicemail on the browser
- Aggregate all SIP numbers associated with a user;
- Deliver a way to search for ways to contact a user;
- Support multiple sessions allowing for a single user to use the system in more than one device at the same time;
- Permitting a user to answer a call in any of the devices that he has connected.
- Give a basic framework to create IVR menus and integrate them with web technologies

3.2 System architecture

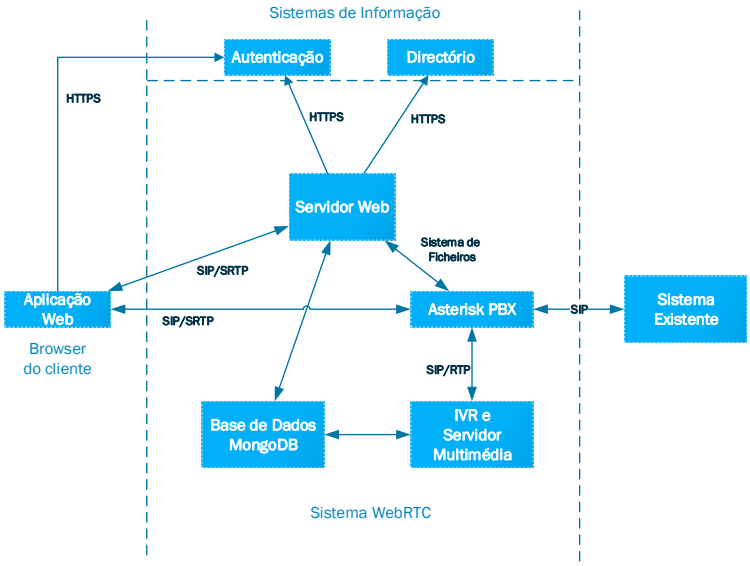

Figure 5 - General system architecture

The system is composed of a Web server, a database, an Asterisk PBX server, an IVR server, and a multimedia server as shown in Figure 5. This system was designed to run simultaneously with the existing infrastructure using the Asterisk PBX to communicate with the existing system.

3.2.1 Web Server

The web server is a very important component of the system having direct contact with almost every other component being the only exception the IVR system. This importance comes from the following 3 functions that the server does:

- Serve a WebApp that gives a way for the users to interact with the system;
- Provides a channel for the exchange of SDP offers/answers and ICE candidates allowing the establishment of a call;
- Configures the Asterisk PBX adding the users to the configuration file and loading the file right after.

The web server also does its more traditional role of providing an API to the WebApp. This API has routes to search for other users, establish WebRTC calls, associate SIP numbers with the user and leave voice/video messages.

3.2.2 WebApp

The WebApp provided by the server gives the users a way to do WebRTC calls and SIP calls to other users and extensions. The application also provides the users an interface to record audio/video messages, preview them and send them to other users. The WebApp also allows the user to search for other users and called them directly from the search results.

3.2.3 Asterisk PBX

The Asterisk PBX serves as a proxy between the users and the POTS/VoIP system, it allows the client applications to register directly to receive calls in their devices. The authentication credentials are set by the web server whenever a new user starts using the system, a personal extension to receive calls is also set. This setup is done by simply changing the configuration files and reloading the relevant modules.

3.2.4 IVR and media server

The IVR server communicates with the web server through WebSocket's using the SIP protocol. From the point of view of the Asterisk PBX the IVR server is just a normal SIP client that registers like the others, allowing the server to receive calls.

The IVR server also communicates with the media server Kurento as a controller allowing the media server to connect directly to Asterisk. The IVR system passes the SDP offers that it receives from Asterisk to Kurento and then the answer from Kurento to Asterisk. After the connection is established the IVR server controls the media that the media server plays in accordance with the menu that it has programed.

3.2.5 Database models

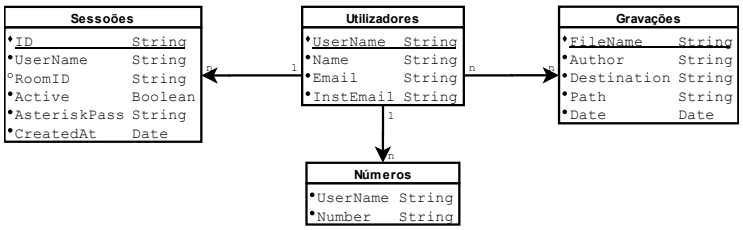

Figure 6 - UML representing the database

The database holds all the user data like associated numbers, active sessions, personal info and recordings metadata. This database is only accessible through the web server and the IVR server not allowing direct access by the clients or any client side executed code.

4 Implementation

4.1 Web Server

The web server was implemented using NodeJS, the language was chosen due to its simplicity and large repository of packets that make web development very easy. It also makes it easier to develop the JavaScript web application since the syntax it's the same.

The packets used in the server were the following:

- Express;
- SocketIO;
- Cookie-Session;
- Body-Parser;
- Multer;
- Simple-Oauth2;
- Random-ID;
- Mongoose;
- Memory-Cache.

The packet Express is the main framework of the web server used to configure all the HTTP routes and is also responsible for preprocess the requests received by parsing the JSON content. SocketIO is the framework for the WebSocket, its attached to the same server as the Express

packet because of that they share the port. SocketIO is used to set the routes for the WebSocket and for sending messages to users or groups of users. Cookie-Session is used to set and sign session cookies in the requests. Body-Parser is used in paired with Express to process the incoming requests. Multer is responsible for handling the files uploaded using to upload records. Simple-Oauth2 is used to simplify the use of the OAuth2.0 framework. Random-ID is used to create random ID's for the sessions and passwords for the Asterisk server. Mongoose is used to interact with MongoDB, acting as interface to create, edit and remove entries from the database. Finally, Memory-Cache is used to store temporary values in the server for use in future requests.

4.1.1 Direct browser to browser call

The Webserver implements the signaling process of a webRTC call using SocketIO, the process can be summarized in the following steps:

1. A user wants to call other and sends a HTTP request to the server containing the username of the called user;
2. The server receives the intention to start a call and learns the name of the caller from the database, after that it stores a call request object in the cache. This object represents the intent of user A call user B. After this the server uses SocketIO to notify all devices of the called user sending a callRequestID and the caller full name.
3. The user that receives the call can either accept or decline the call. If the user accepts it send a request with the callRequestID and the identifier (in this system) of the answering device.
4. The server retrieves the callRequest object using the callRequestID and checks if all if correct user responded. If everything is ok the server creates a "room" using SocketIO to serve as a signaling channel for the call.
5. The server maintains the channel until the end of the call destroying it after.

After the establishment of the communication channel the users can start to send SDP offers/answers and ICE candidates. The server has no part in the resulting connection since the data will flow directly from one WebRTC client to another. The channel is only maintained to signal the end of a call or any necessary changes.

4.2 Web Application

The web application was made using JavaScript and HTML with help of the AngularJS framework.

The core packets used in the web application were the following:

- AngularJS
- JQuery
- JSSIP
- SocketIO
- Adapter.js

As said earlier AngularJS is used in the app as the main framework, it allows the use of AJAX (Asynchronous Javascript and XML) to be used in very simple way by creating routes for the application using ngRoute that's one of many modules delivered by Angular. The use of AJAX is vital for the since the JavaScript code that controls the calls must stay running without interruptions. The packet jQuery needs to be loaded to ensure the correct behavior of AngularJS.

JSSIP is the packet responsible for managing all SIP connections with asterisk. It registers the application so that it can receive and make calls and handles all the low level signaling required giving the programmer a comprehensive API to control the calls in an easy way.

SocketIO like in the server is used to receive/send information to the server in an asynchronous way giving the application a way to receive information from the server at any time.

Adapter.js is shim library that gives the application the capacity of working with the WebRTC implementations of several browsers and several versions of the same browser.

4.3 Asterisk Server

The Asterisk server has several modules that must be configured to setup the system like the integrated HTTP server, SIP channel driver, RTP handlers and the dial plan.

The integrated HTTP server is used to host the WebSocket that works as a signaling interface for the web clients it can be used both in the secure version WSS and the unsecure WS. The only requirement to use the secure version is configuring the HTTPS protocol in the web server.

The RTP configuration must be changed so that the server uses the ICE protocol and a defined STUN server.

The SIP channel driver chosen was PJSIP this module configures the transport types possible, the authentication points and AOR's that then associates with endpoints. There is no fixed association between endpoints and transport layers so that an endpoint can be created using any of the transport types.

The dial plan defines all extensions in the system and so one extension must be associated with every user allowing calls to be received by any user. The dial plan must also explicitly be configured to make Asterisk contact all active endpoints associated with a given user when there is an incoming call.

4.4 IVR Server

This server was implemented using NodeJS to maintain the coherency across the system.

The packets used for this server were:

- SIPJS
- Kurento Client

The SIPJS module is used to connect the IVR server with the Asterisk server using WebSocket's. The module also handles the SIP communication allowing the server to answer calls while forwarding the SDP description to the multimedia server and the SDP answer to the Asterisk server. This module also receives and processes the DTMF tones allowing an IVR to be built based on this input.

The Kurento client library is used to control the multimedia server (Kurento), this control is made through a WebSocket that allows the creation of endpoints in the server for communication and for playing and recording media.

The server delivers an effective way of developing IVR menus that far surpass the usual Asterisk ones allowing developers to link it directly with any kind of database which allows an easy integration with web technologies which in turn opens the possibility to deliver complex services using IVR menus.

4.5 Media Server

The Kurento media server was used to record and play audio for all SIP devices. The setup of this service is very easy it only requires an installation that can be done directly from the repositories of a Linux distribution like Ubuntu. The server will work right out of the box without any special configuration nevertheless it will be required an RSA key pair and some changes on the config file to enable WSS.

5 Discussion

5.1 Security

5.1.1 Communications via Webserver

All the communications between the users and the web server and the users are encrypted using TLS (Transport Layer Security [10]). This used to protect both the HTTP server and the WebSocket implementing HTTPS and WSS. This ensures the integrity and confidentiality of all messages exchanged between the server and the users.

With the use of HTTPS, the cookie can safely contain the identifier of the user in the system without the risk of it being used by a malicious user to impersonate another person. For additional security the server also signs its cookies so that they can't be easily forged.

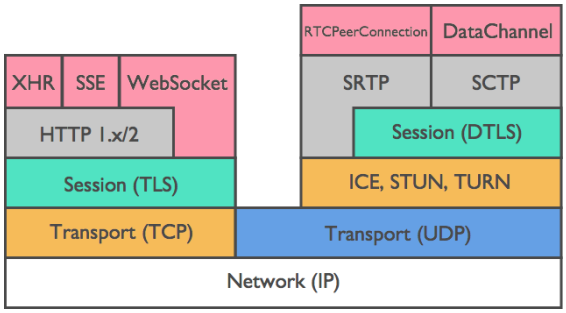

Figure 7 - WebRTC protocol stack [11]

The use of WSS protects the signaling of all calls made from a browser to another, the level of security is the same of the HTTPS server since they share the same transport layer, in fact the difference between the two lies only on the application layer.

The media streams exchanged on a direct communication between browsers are encrypted using DTLS (Datagram Transport Layer Security) below SRTP and SCTP this ensures integrity and a very good level of confidentiality but due to a lack of header encryption in the SRTP header is still possible to see that a call is happening and the IP addresses of the users [11], as it can be seen in Figure 7. The exposure of the IP address can lead to the identification of the two parties that are engaging in the call, despite this the content of the call will remain confidential until there is a way to break the encryption used.

5.1.2 Authentication on the Webserver

The authentication on the webserver is made using the OAusth2.0 framework. This framework gives the webserver both the means to authenticate a user and a way to obtain user data in a secure way. A success authentication process results in the creation of a session in the webserver where the username is related with a generated identifier. The identifier is set in the signed cookie of the session and used as way to retrieve the session from the database. This schema allows a user to have more than one session at time if he repeats the authentication process in another device/browser. The sessions of a user are erased when he logs out or when it surpasses its set lifetime. If a user identifier gets exposed, it will only grant access to the system during the rest of its lifetime being this time impossible to extend.

5.1.3 Communication via Asterisk

The communication with Asterisk can be accomplished either via UDP or TCP using a WebSocket. Both methods have been configured in two ways a secure and an unsecure one. The UDP configuration uses one port to receive standard non-encrypted SIP messages and another to received messages encrypted with TLS. On the WebSocket side we also have an implementation of the WS protocol with the messages exposed in plain-text and a WSS implementation with TLS encryption protecting the all messages.

The endpoints defined within Asterisk can chose use encryption for the sessions or not. For all WebRTC communications the use of DTLS is mandatory and so the endpoints were configured to use it. To protect the signaling the transport used must be WSS to protect integrity and confidentiality of a future session.

UDP clients using the secure transport layer can also encrypt their media session using a cryptographic attribute in SDP protocol [12]. In conjunction with the TLS encryption that protects the SIP/SDP messages communications over UDP can achieve integrity and confidentiality.

The unprotected versions of the protocols were implemented to achieve compatibility with as much devices as possible, despite this their use must be kept to bare minimum and only within secure networks.

5.1.4 Authentication on Asterisk server

The authentication in Asterisk is made using the credentials delivered by the webserver using its API. The credentials consist of a username that it's the username of the user in the system and a password generated by the webserver. Asterisk allows the same user the possibility of authenticating more than one device using the AOR that works as an identifier service for multiple devices. The separation of credentials makes the overall system more secure. The OAuth2.0 username and password are used only once per session to acquire the Asterisk credentials and if for some reason these credentials are exposed only the Asterisk part of the system is exposed and so it can be fixed by changing periodically and automatically the Asterisk password which is easy because its value is not known by the user.

5.2 NodeJS IVR server Vs Asterisk

The Asterisk as most of the IP-PBX servers provides a framework to develop IVR menus, unfortunately these menus are limited and use special language only defined in the scope of the server. Using Asterisk only some Database's can be used and all of them are SQL databases, this fact combining with the complexity of using this inside the Asterisk configuration files.

On the other hand, the NodeJS IVR server can use almost any database both SQL and NoSQL. The use of JavaScript language allows the development of menus that can be easily integrated with any webservice. The possibility of integrating webservices with IVR menus allows the possibility to create more dynamic and powerful menus.

5.3 Directory

This system asks the user for input to set is attributed SIP numbers/extensions this method is both inconvenient for the user and unsecure for the system. Its inconvenient because it asks for something that should be acquirable using the OAuth2.0 framework and its unsecure because nobody certifies that the user claims are valid. There should be a way to retrieve the SIP numbers from the central system, with the permission of the user, using OAuth2.0.

5.4 Management and maintenance

This system was all written using the same programing language, JavaScript. This makes the maintenance of the system easier since it is possible for a programmer to handle all system just by knowing JavaScript.

The management of the Asterisk server is a bit trickier than the rest since its knowledge in SIP and several other auxiliary protocols is needed to fully understand how an Asterisk server must be configured. Despite this the entire philosophy of this project was oriented to take work away from the asterisk server to reduce the complexity of the configuration files.

6 Conclusions

This project was able to integrate WebRTC technology with exiting POTS/VoIP networks being able to deliver a real phone experience through the browser. The was also able to use other technologies like OAuth2.0 to authenticate clients and retrieve data.

Using the multimedia server Kurento this project was also capable of creating IVR menus using NodeJS allowing the integration of web technologies with VoIP which translates in an enhancement of the user experience since we can deliver a more personalized IVR for the clients/users.

The web application developed as part of the project can run in the Google Chrome and Mozilla Firefox browsers both in the mobile and desktop versions. Since these browsers correspond to the first and third most used browsers with a combined market share around 65% (during the month of February of 2018, data from the website NetMarketShare) it's safe to say that the application will be available to most people.

There are alternatives to create redundancy systems to cover a failure in the POTS system like softphones for example. But the use of a softphone requires individual installation and configuration in each device, which can be hard troublesome especially when using secure connections, all these can cause problems to less experienced users. The Web Application can be executed in the most popular browsers without any installation or configuration, the only requirement is a simple login into on a familiar website. With this the application can fetch all the data it needs to work.

Using a web application instead of a softphone gives more options to do further improvement and expansion of the system since it as full control over the interface offered to the user.

7 Bibliography

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, M. R. Sparks, Handley and E. Schooler, "RFC3261 - SIP: Session Initiation Protocol," June 2002. [Online]. Available: https://tools.ietf.org/html/rfc3261. [Accessed 13 April 2018].

[2] M. Handley, V. Jacobson and C. Perkins, "RFC 4566 - SDP: Session Description Protocol," July 2006. [Online]. Available: https://tools.ietf.org/html/rfc4566. [Accessed 13 April 2018].

[3] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, "RFC 3550 - RTP: A Transport Protocol for Real-Time Applications," July 2003. [Online]. Available: https://tools.ietf.org/html/rfc3550. [Accessed 13 April 2018].

[4] M. Baugher, D. McGrew, M. Naslund, E. Carrara and K. Norrman, "RFC 3711 - The Secure Real-time Transport Protocol (SRTP)," IETF, March 2004. [Online]. Available: https://tools.ietf.org/html/rfc3711. [Accessed 13 April 2018].

[5] J. Rosenberg, "RFC 5245 - Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," April 2010. [Online]. Available: https://tools.ietf.org/html/rfc5245. [Accessed 13 April 2018].

[6] J. Rosenberg, R. Mahy, P. Matthews and D. Wing, "RFC 5389 - Session Traversal Utilities for NAT (STUN)," October 2008. [Online]. Available: https://tools.ietf.org/html/rfc5389. [Accessed 26 April 2018].

[7] A. Prokop, "Understanding WebRTC Media Connections: ICE, STUN and TURN," 11 August 2014. [Online]. Available: https://www.avaya.com/blogs/archives/2014/08/understanding-webrtc-media-connections-ice-stun-and-turn.html. [Accessed 26 April 2018].

[8] E. Ivov, E. Rescorla, J. Uberti and P. Saint-Andre, "draft-ietf-ice-trickle-20 - Trickle ICE:

Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol," 9 April 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-ice-trickle-15. [Accessed 13 April 2018].

[9] D. Hardt, "RFC 6749 - The OAuth 2.0 Authorization Framework," October 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749. [Accessed 2018 April 18].

[10] T. Dierks and E. Rescorla, "RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2," August 2008. [Online]. Available: https://tools.ietf.org/html/rfc5246. [Accessed 3 May 2018].

[11] NTT Communications, "A Study of WebRTC Security," [Online]. Available: http://webrtc-security.github.io/. [Accessed 19 April 2018].

[12] F. Andreasen, M. Baugher and D. Wing, "RFC 4568 - Session Description Protocol (SDP) Security Descriptions for Media Streams," July 2006. [Online]. Available: https://tools.ietf.org/html/rfc4568. [Accessed 23 April 2018].

[13] A. Freier, P. Karlton and P. Kocher, "RFC 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0," August 2011. [Online]. Available: https://tools.ietf.org/html/rfc6101. [Accessed 18 April 2018].

[14] E. Rescorla and N. Modadugu, "RFC 6347 - Datagram Transport Layer Security Version 1.2," January 2012. [Online]. Available: https://tools.ietf.org/html/rfc6347. [Accessed 19 April 2018].