DEVELOPMENT AND EVALUATION OF A WEB FRIENDLY TELEPHONE SYSTEM

by

DASKALAKIS DIMITRIOS CHRISTOS

Applied Informatics and Multimedia, T.E.I. Of Crete

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF APPLIED INFORMATICS
AND MULTIMEDIA

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

2016

Approved by:

Assistant Professor
SPYROS PANAGIOTAKIS

Left intentionally blank

Copyright

Abstract

In this thesis we exploit the relative maturity of technologies and libraries such as WebRTC, JsSIP and Asterisk in order to achieve enhanced communications abilities. We enable the browser to accept and make calls, without the need of any extra software needing to be installed from both mobile and standard PC devices. Also we developed a way to make one click, preconfigured calls catering to the needs of modern enterprise users and also a fully-fledged web based SIP phone. Those calls are cryptographically secured between browsers. The calls, as we have demonstrated, can be directed to a land line as well, if desired. Also by gathering live statistics provided by the browser and the asterisk server, we have determined the objective perceived call quality score (MOS).

Keywords: WebRTC, communications, click to call button, SIP, asterisk

Table of Contents

List of Figures

List of Tables

Abbreviation Index

API	Application Programming Interface
PBX	Private Branch Exchange
CA	Certification Authority
CSS	Cascading Styling Sheet
DOM	Document Object Model
DTLS	Datagram Transport Layer Security
FXO	Foreign eXchange Office interface
FXS	Foreign eXchange Subscriber interface
HTML	HyperText Markup Language
HTML5	HyperText Markup Language5
HTTP	HyperText Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer.
ICE	Interactive Connectivity Establishment.
IETF	Internet Engineering Task Force
IP	Internet Protocol.
JS	JavaScript
JSEP	Javascript Session Establishment Protocol
MPBX	Multimedia Private Branch Exchange.
NAT	Network Address Translation
P2P	PeerToPeer.
P2P	PeertoPeer
PBX	Private Branch Exchange.
PHP	PHP: Hypertext Preprocessor.
POTS	Plain Old Telephone Service
PSTN	Public Switched Telephone Network.
IVR	Interactive Voice Response
QoS	Quality of Service.

RTC	RealTime Communication
RTP	RealTime Protocol
SCTP	Stream Control Transmission Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SRTP	Secure Real Time Protocol
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TURN	Traversal Using Relays around NAT
UA	User Agent.
UAC	User Agent Client.
UAS	User Agent Server.
UCP	User Control Panel
UDP	User Datagram Protocol
UI	User Interface.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
VM	Virtual Machine.
VoIP	Voice over Internet Protocol.
VP8	Video compression format
QoE	Quality of Experience
QoS	Quality Of Service
W3C	World Wide Web Consortium.
W3C	World Wide Web Consortium
WebRTC	Web RealTime Communication
WebSocket	Protocol that supports bidirectional communication over a single TCP connection
XMPP	Extensible Messaging and Presence Protocol

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Spyros Panagiotakis for the continuous support of my MSc study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Malamos Athanasios and Dr. Kostas Vasilakis for their insightful comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives.

My sincere thanks also go to Dr. Athanasios Malamos who provided me an opportunity to join their team, and who gave access to the laboratory and research facilities.

I thank my fellow labmates in for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last two years. Also I need to thank Mr. Adamantios Aretakis for helping me in various ways with my life.

Last but not the least, I would like to thank my wife and family: my parents and to my brother for supporting me throughout writing this thesis and my life in general.

Motivation

As our world becomes more and more interconnected we strive for more access and ease of use regarding our communications. Devices and technologies these days enable this to be exceptionally true. Mobile phones have reached computing power well above a PC would have a decade ago and 4G networks make high speed affordable internet within reach. Even more so, web browsers have evolved from a static forum to get information to fully fledged bidirectional and programmable communications platforms. With technologies such as HTML5[1] and WebRTC[2] the boundaries and difficulties for multiplatform and multi device have become almost nonexistent. Thus the motivation of this thesis is to explore those emerging technologies and to evaluate them.

Challenges

There were numerous challenges concerning this thesis. The main categories were research related and implementation related.

As research challenges are concerned, WebRTC is a technology new to the web and not scandalized entirely. There are a lot of articles and how to written about it but they are not all exactly relevant as references, mainly because of different browsers have slightly different implementations on WebRTC and some of the implementation keeps changing with the updates of the browser, or if the browser is a beta or a nightly version. Moreover, there is not a lot of many open sourced projects and community to support SIP on the web, although it is gaining on this aspect fast!

There is no direct implementation between SIP and WebRTC in the commercial market. The two open source frameworks we used is sipml5 and JsSIP. The later was chosen after some initial testing. There is no plethora of references and documentation that could be helpful during development. We had to understand and developed the system and all the protocols involved and the fundamentals about SIP protocol, which is a very tedious protocol to understand fully WebRTC implementation in JavaScript, unknown to me prior to this thesis. In order to implement the thesis service based on SIP and WebRTC it required a lot of time on programming prototypes and evaluate the solutions in a very time consuming try and fail methodology. The JavaScript

programing was a big learning experience alone. Also the asterisk server, while known to me from previous work it posed a very big challenge to implement the security the WebRTC imposes (https, SRTP, WSS). As troubleshooting goes, analyzing traffic with wireshark is a nightmare. Furthermore, there were many design cases that we had to consider during the development because the thesis requirements.

1 Introduction

1.1 VOIP

Stands for "Voice over Internet Protocol," and is often pronounced "VoIP." VoIP is basically a telephone connection over the Internet. The data is sent digitally, using the Internet Protocol (IP) instead of analog telephone lines. This allows people to talk to one another long-distance and around the world without having to pay long distance or international phone charges.

In order to use VoIP, you need a computer, an Internet connection, and VoIP software or hardware. You also need either a microphone, analog telephone adapter, or VoIP telephone. Many VoIP programs allow you to use a basic microphone and speaker setup. Analog telephone adapters allow you to use regular phones with your computer or router. IP phones are another option that connect directly to a router via Ethernet or wirelessly. These phones have all the necessary software for VoIP built in and therefore do not require a computer. VoIP is also referred to as IP telephony, Internet telephony, and digital phone.

Protocols used by Voice over IP have been implemented in various ways using both proprietary protocols and protocols based on open standards. Examples of the VoIP protocols are: H.323, Media Gateway Control Protocol (MGCP), Session Initiation Protocol (SIP), H.248, Real-time Transport Protocol (RTP), Real-time Transport Control Protocol (RTCP), Secure Real-time Transport Protocol (SRTP), Session Description Protocol (SDP), Inter-Asterisk eXchange (IAX), Jingle XMPP VoIP extensions, Skype protocol, TeamSpeak.

Because of the bandwidth efficiency and low costs that VoIP technology can provide, businesses are migrating from traditional copper-wire telephone systems to VoIP systems to reduce their monthly phone costs. In 2008, 80% of all new Private branch exchange (PBX) lines installed internationally were VoIP.

VoIP solutions aimed at businesses have evolved into unified communications services that treat all communications—phone calls, faxes, voice mail, e-mail, Web conferences, and more—as discrete units that can all be delivered via any means and to any handset, including cell phones. Two kinds of competitors are competing in this space: one set is focused on VoIP for medium to large enterprises, while another is targeting the small-to-medium business (SMB) market.

VoIP allows both voice and data communications to be run over a single network, which can significantly reduce infrastructure costs.

The prices of extensions on VoIP are lower than for PBX and key systems. VoIP switches may run on commodity hardware, such as personal computers. Rather than closed architectures, these devices rely on standard interfaces.

VoIP devices have simple, intuitive user interfaces, so users can often make simple system configuration changes. Dual-mode phones enable users to continue their conversations as they move between an outside cellular service and an internal Wi-Fi network, so that it is no longer necessary to carry both a desktop phone and a cellphone. Maintenance becomes simpler as there are fewer devices to oversee.

VoIP can be a benefit for reducing communication and infrastructure costs. Some examples include:

- The ability to transmit more than one telephone call over a single broadband connection.
- Secure calls using standardized protocols (such as Secure Real-time Transport Protocol). Most of the difficulties of creating a secure telephone connection over traditional phone lines, such as digitizing and digital transmission, are already in place with VoIP. It is only necessary to encrypt and authenticate the existing data stream.
- Utilized existing network infrastructure to minimize the operating cost.
- Routing phone calls over existing data networks to avoid the need for separate voice and data networks.
- Eliminating the need of hiring personnel to greet and distribute incoming calls with the use of a Virtual PBX

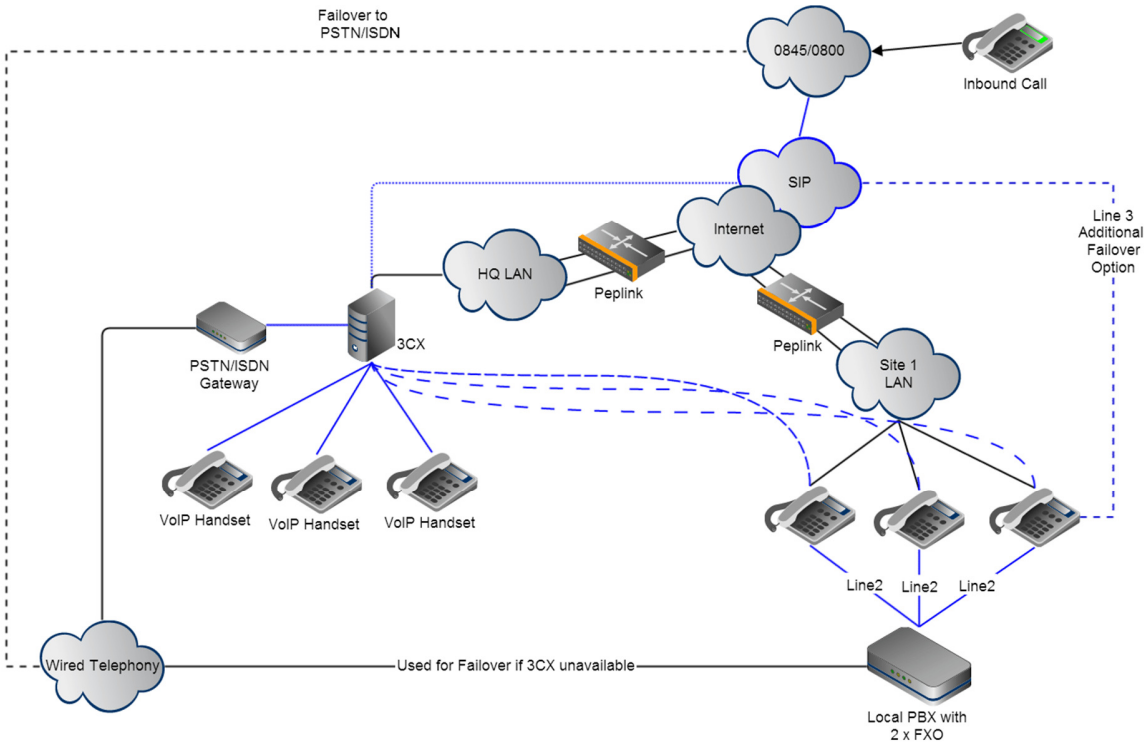


Figure 1: A typical VoIP setup

1.2 VoIP Protocols

In this section we discuss the critical protocols VoIP uses to enable communication in real world scenarios.

1.2.1 SDP

Session Description Protocol (SDP) [3] is a standard for describing the multimedia content of the connection such as encryption, formats, codecs, resolution, etc. so that both peers can understand each other once the data is transferring. This is the metadata describing the content and not the media content itself.

Here is a typical SDP message:

```
v=0
o=alice 2846524526 2890853526 IN IP4 host.com
s=
```

```
c=IN IP4 host.com
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
m=video 51372 RTP/AVP 31
a=rtpmap:31 H261/90000
m=video 53000 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

SDP is never used alone, but along with protocols like SIP and RTCP.

1.2.2 RTP

RTP RFC 3550 RTP provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services. RTP does not address resource reservation and does not guarantee quality-of-service for real-time services. RTP combines its data transport with a control protocol (RTCP), which makes it possible to monitor data delivery for large multicast networks. Monitoring allows the receiver to detect if there is any packet loss and to compensate for any delay jitter. Both protocols work independently of the underlying Transport layer and Network layer protocols. Information in the RTP header tells the receiver how to reconstruct the data and describes how the codec bit stream are packetized. As a rule, RTP runs on top of the User Datagram Protocol (UDP), although it can use other transport protocols. Both the Session Initiation Protocol (SIP) and H.323 use RTP. RTP components include: a *sequence number*, which is used to detect lost packets; *payload identification*, which describes the specific media encoding so that it can be changed if it has to adapt to a variation in bandwidth; *frameindication*, which marks the beginning and end of each frame; *source identification*, which identifies the originator of the frame; and *intramedia synchronization*, which uses timestamps to detect different delay jitter within a single stream and compensate for it. Compressed RTP (CRTP), specified in RFC 2509 [4], was developed to decrease the size of the IP, UDP, and RTP headers. However, it was designed to work with reliable and fast point-to-point links. In less than optimal circumstances, where there may be long delays, packet loss, and out-of-sequence packets, CRTP doesn't function well for Voice over IP

applications. Another adaptation, Enhanced CRPT (ECRPT), was defined in a subsequent Internet Draft document to overcome that problem.

1.2.3 RTCP

The Real Time Streaming Protocol (RTCP), [5] is a protocol that works in the application level and has to control the delivery of real time data, such as video and audio. Sources can be for instance live data streams or stored clips. This protocol controls numerous data delivery sessions, provides a mean for choosing delivery transports such as UDP, multicast UDP or TCP, and provides a means for choosing delivery mechanisms based upon RTP. Its job is not to deliver streams itself. In other words, we can see RTCP as a remote control for multimedia servers. Also by the W3C document [6] the RTCP packets contain statistical information about the streams they are controlling and also they may be multiplexed with the RTP packets. The packets Statistical flags that would be interesting for us are. In practice they are not mandatory to implement.

- SSRC (synchronization source identifier)
- NTP timestamp
- RTP timestamp
- sender's packet count
- sender's octet count (payload not including headers)
- fraction lost
- cumulative number of packets lost
- extended highest sequence number received
- interarival jitter

1.2.4 SIP

The protocol that has prevailed over the years for VoIP use is SIP. The Session Initiation Protocol (SIP) as we can see defined in RFC 3261 [7] is an application signaling protocol for setting up modifying and then terminating real-time calls - sessions between participants over an IP connection. SIP supports a plethora of single-media or multi-media session, including teleconferencing. SIP is the signaling protocol that enables one party to place a call to another party and to negotiate the parameters of a multimedia session. The actual audio, video, or other multimedia content is exchanged between session participants using an appropriate transport protocol. In many cases, the transport protocol to use is the Real-Time Transport Protocol (RTP). Directory access and lookup protocols are also needed.

The key driving force behind SIP is to enable Internet telephony. SIP is the typical IP signaling mechanism for voice and multimedia calling services. SIP supports five facets of establishing and terminating multimedia communications:

- User location: Users can move to other locations and access their telephony or other application features from remote locations.
- User availability: This step involves determination of the willingness of the called party to engage in communications.
- User capabilities: In this step, the media and media parameters to be used are determined.
- Session setup: Point-to-point and multiparty calls are set up, with agreed session parameters.
- Session management: This step includes transfer and termination of sessions, modifying session parameters, and invoking services.

SIP employs design elements developed for earlier protocols. SIP is based on an HTTP-like request/response transaction model. Each transaction consists of a client request that invokes a particular method, or function, on the server and at least one response. SIP uses most of the header fields, encoding rules, and status codes of HTTP. This provides a readable text-based format for displaying information. SIP incorporates the use of a Session Description Protocol (SDP), which defines session content using a set of types similar to those used in Multipurpose Internet Mail Extensions (MIME).

1.2.4.1 SIP Components and Protocols

A system using SIP can be viewed as consisting of components defined on two dimensions: client/server and individual network elements. RFC 3261 defines client and server as follows:

- Client: A client is any network element that sends SIP requests and receives SIP responses. Clients may or may not interact directly with a human user. User agent clients and proxies are clients.
- Server: A server is a network element that receives requests in order to service them and sends back responses to those requests. Examples of servers are proxies, user agent servers, redirect servers, and registrars.

The individual elements of a standard SIP configuration include the following:

User Agent: The user agent resides in every SIP end station. It acts in two roles:

- User Agent Client (UAC): Issues SIP requests
- User Agent Server (UAS): Receives SIP requests and generates a response that accepts, rejects, or redirects the request

- Redirect Server: The redirect server is used during session initiation to determine the address of the called device. The redirect server returns this information to the calling device, directing the UAC to contact an alternate Universal Resource Identifier (URI). A URI is a generic identifier used to name any resource on the Internet. The URL used for Web addresses is a type of URI as we see in RFC 2396 [8].
- Proxy Server: The proxy server is an intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy server primarily plays the role of routing, meaning that its job is to ensure that a request is sent to another entity closer to the targeted user. Proxies are also useful for enforcing policy (for example, making sure a user is allowed to make a call). A proxy interprets, and, if necessary, rewrites specific parts of a request message before forwarding it.

- Registrar: A registrar is a server that accepts REGISTER requests and places the information it receives (the SIP address and associated IP address of the registering device) in those requests into the location service for the domain it handles.
- Location Service: A location service is used by a SIP redirect or proxy server to obtain information about a callee's possible location(s). For this purpose, the location service maintains a database of SIP-address/ IP-address mappings.

The various servers are defined in RFC 3261 [7] as logical devices. They may be implemented as separate servers configured on the Internet or they may be combined into a single application that resides in a physical server.

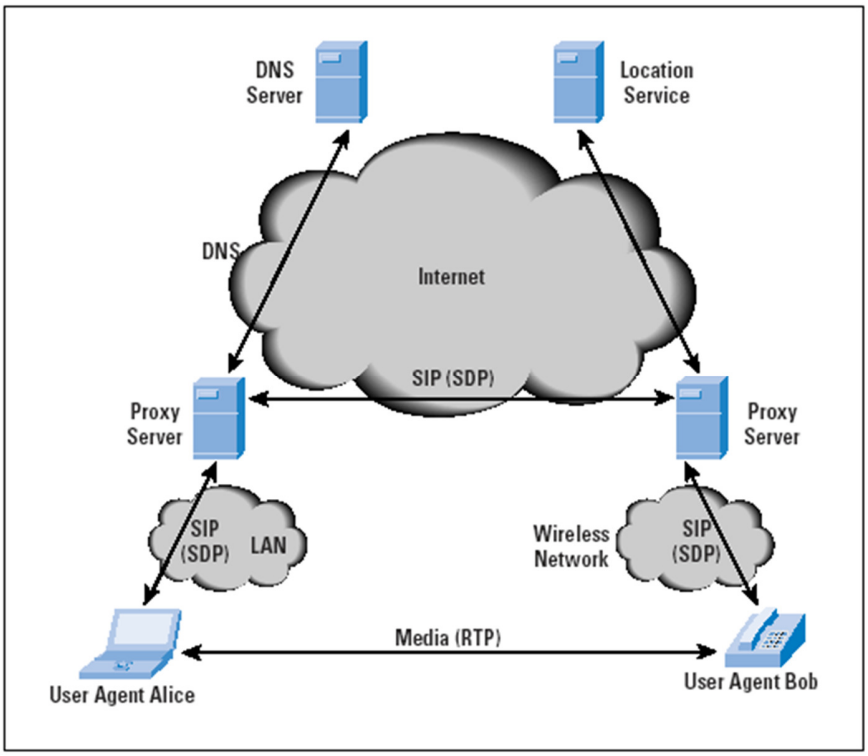


Figure 4: SIP Components and Protocols

Figure 1 shows how some of the SIP components relate to one another and the protocols that are employed. A user agent acting as a client (in this case UAC Alice) uses SIP to set up a session with a user agent that acts as a server (in this case UAS Bob). The session initiation dialogue uses

SIP and involves one or more proxy servers to forward requests and responses between the two user agents. The user agents also make use of the SDP, which is used to describe the media session.

The proxy servers may also act as redirect servers as needed. If redirection is done, a proxy server needs to consult the location service database, which may or may not be collocated with a proxy server. The communication between the proxy server and the location service is beyond the scope of the SIP standard. The Domain Name System (DNS) is also an important part of SIP operation. Typically, a UAC makes a request using the domain name of the UAS, rather than an IP address. A proxy server needs to consult a DNS server to find a proxy server for the target domain.

SIP often runs on top of the User Datagram Protocol (UDP) for performance reasons, and provides its own reliability mechanisms, but may also use TCP. If a secure, encrypted transport mechanism is desired, SIP messages may alternatively be carried over the Transport Layer Security (TLS) protocol.

Associated with SIP is the SDP, defined in RFC 2327 [9]. SIP is used to invite one or more participants to a session, while the SDP-encoded body of the SIP message contains information about what media encodings (for example, voice, video) the parties can and will use. After this information is exchanged and acknowledged, all participants are aware of the participants' IP addresses, available transmission capacity, and media type. Then, data transmission begins, using an appropriate transport protocol. Typically, the RTP is used. Throughout the session, participants can make changes to session parameters, such as new media types or new parties to the session, using SIP messages.

1.2.4.2 Examples of Operation

The SIP specification is quite complex; the main document, RFC 3261[7], is 269 pages long. To give some feel for its operation, we present a few examples.
Figure 4 shows a successful attempt by user Alice to establish a session with user Bob, whose URI is bob@biloxi.com. Alice's UAC is configured to communicate with a proxy server (the outbound server) in its domain and begins by sending an INVITE message to the proxy server that indicates

its desire to invite Bob's UAS into a session (1); the server acknowledges the request (2). Although Bob's UAS is identified by its URI, the outbound proxy server needs to account for the possibility that Bob is not currently available or that Bob has moved. Accordingly, the outbound proxy server should forward the INVITE request to the proxy server that is responsible for the domain biloxi.com. The outbound proxy thus consults a local DNS server to obtain the IP address of the biloxi.com proxy server (3), by asking for the DNS SRV resource record that contains information on the proxy server for biloxi.com

.

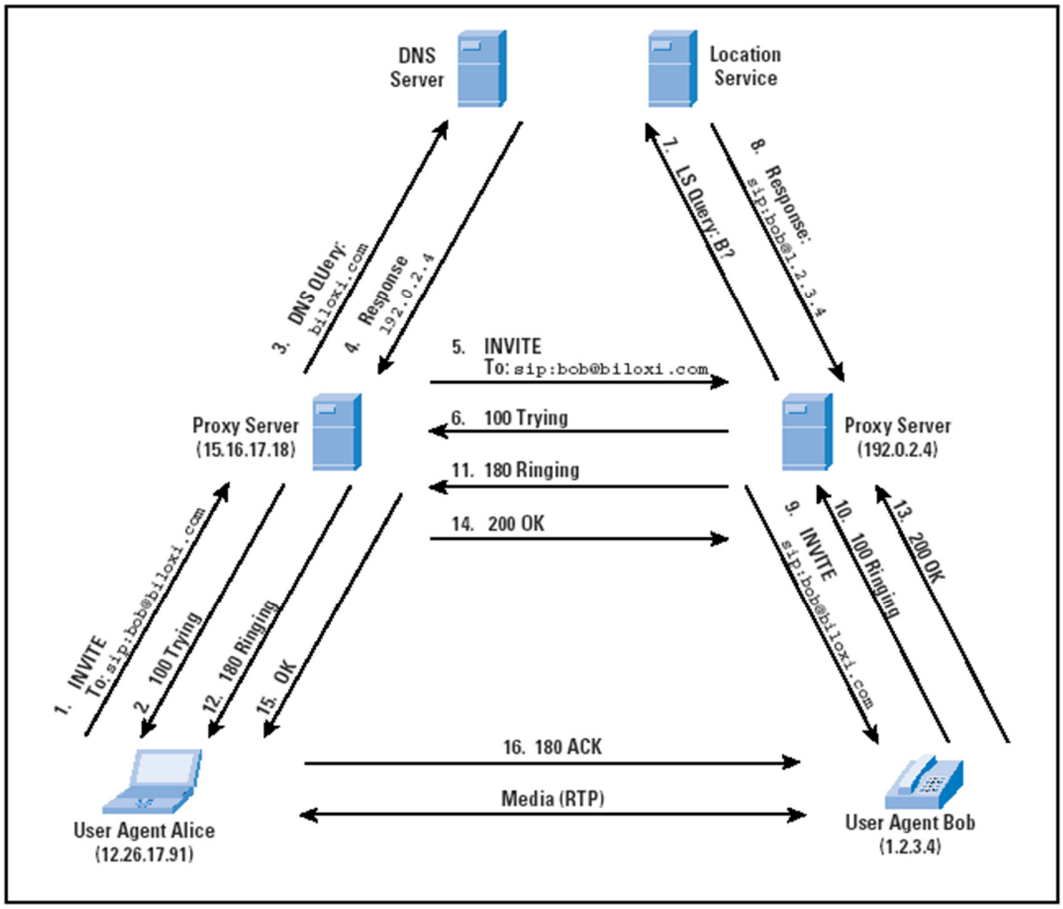


Figure 2: SIP Successful Call Setup

The DNS server responds (4) with the IP address of the biloxi.com proxy server (the inbound server). Alice's proxy server can now forward the INVITE message to the inbound proxy server (5), which acknowledges the message (6). The inbound proxy server now consults a location

server to determine Bob's location Bob (7), and the location server responds with Bob's location, indicating that Bob is signed in, and therefore available for SIP messages (8).

The proxy server can now send the INVITE message on to Bob (9). A ringing response is sent from Bob back to Alice (10, 11, 12) while the UAS at Bob is alerting the local media application (for example, telephony). When the media application accepts the call, Bob's UAS sends back an OK response to Alice (13, 14, 15). Finally, Alice's UAC sends an acknowledgement message to Bob's UAS to confirm the reception of the final response (16). In this example, the ACK is sent directly from Alice to Bob, bypassing the two proxies. This occurs because the endpoints have learned each other's address from the INVITE/200 (OK) exchange, which was not known when the initial INVITE was sent. The media session has now begun, and Alice and Bob can exchange data over one or more RTP connections.

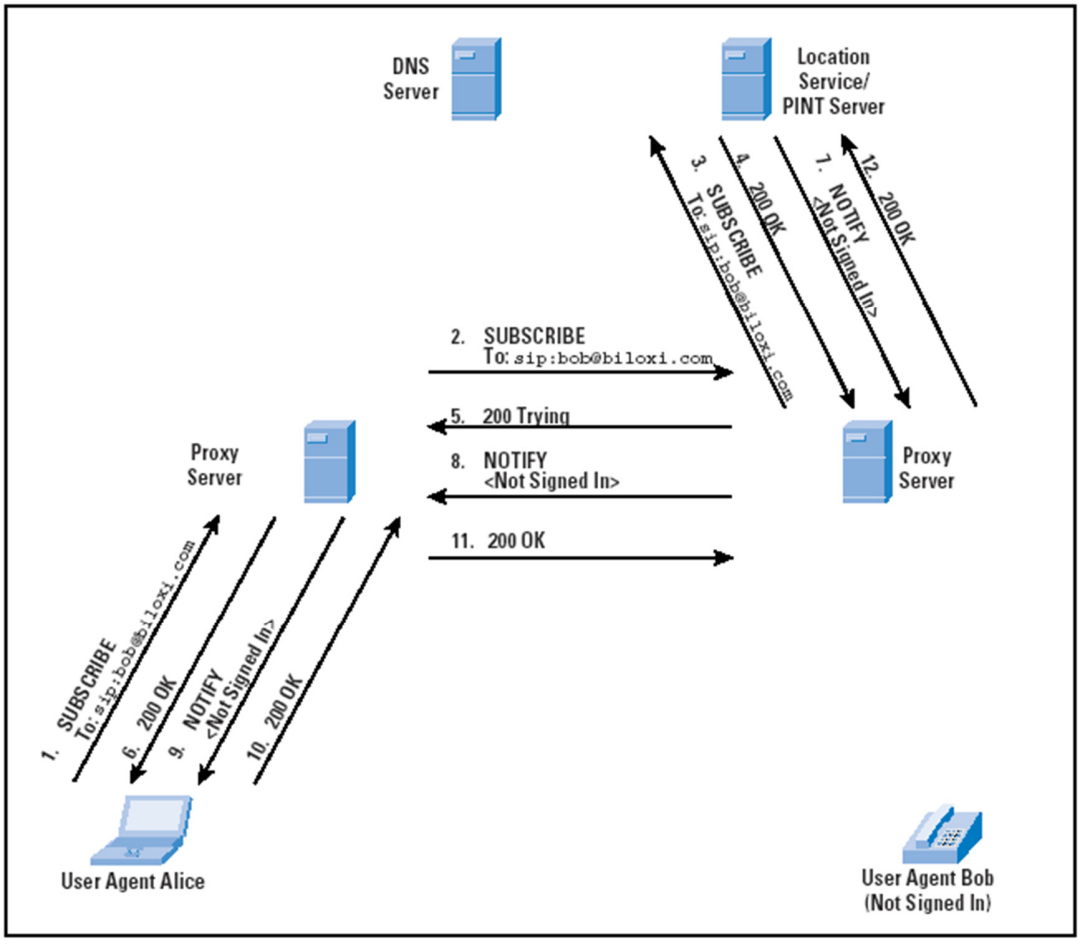


Figure 3: SIP Presence Example

The next example (Figure 4) makes use of two message types that are not yet part of the SIP standard but that are documented in RFC 2848 [10] and are likely to be incorporated in a later revision of SIP. These message types support telephony applications. Suppose that in the preceding example, Alice was informed that Bob was not available. Alice's UAC can then issue a SUBSCRIBE message (1), indicating that it wants to be informed when Bob is available.

This request is forwarded through the two proxies in our example to a PINT (Public Switched Telephone Network [PSTN]-Internet Networking) server (2, 3). A PINT server acts as a gateway between an IP network from which comes a request to place a telephone call and a telephone network that executes the call by connecting to the destination telephone. In this example, we assume that the PINT server logic is collocated with the location service. It could also be the case that Bob is attached to the Internet rather than a PSTN, in which case the equivalent of PINT logic is needed to handle SUBSCRIBE requests. In this example, we assume the latter and assume that the PINT functionality is implemented in the location service. In any case, the location service authorizes subscription by returning an OK message (4), which is passed back to Alice (5, 6). The location service then immediately sends a NOTIFY message with Bob's current status of not signed in (7, 8, 9), which Alice's UAC acknowledges (10, 11, 12).

Figure 6 continues the example of Figure 5. Bob signs on by sending a REGISTER message to the proxy in its domain (1). The proxy updates the database at the location service to reflect registration (2). The update is confirmed to the proxy (3), which confirms the registration to Bob (4). The PINT functionality learns of Bob's new status from the location server (here we assume that they are collocated) and sends a NOTIFY message containing Bob's new status (5), which is forwarded to Alice (6, 7). Alice's UAC acknowledges receipt of the notification (8, 9, 10).

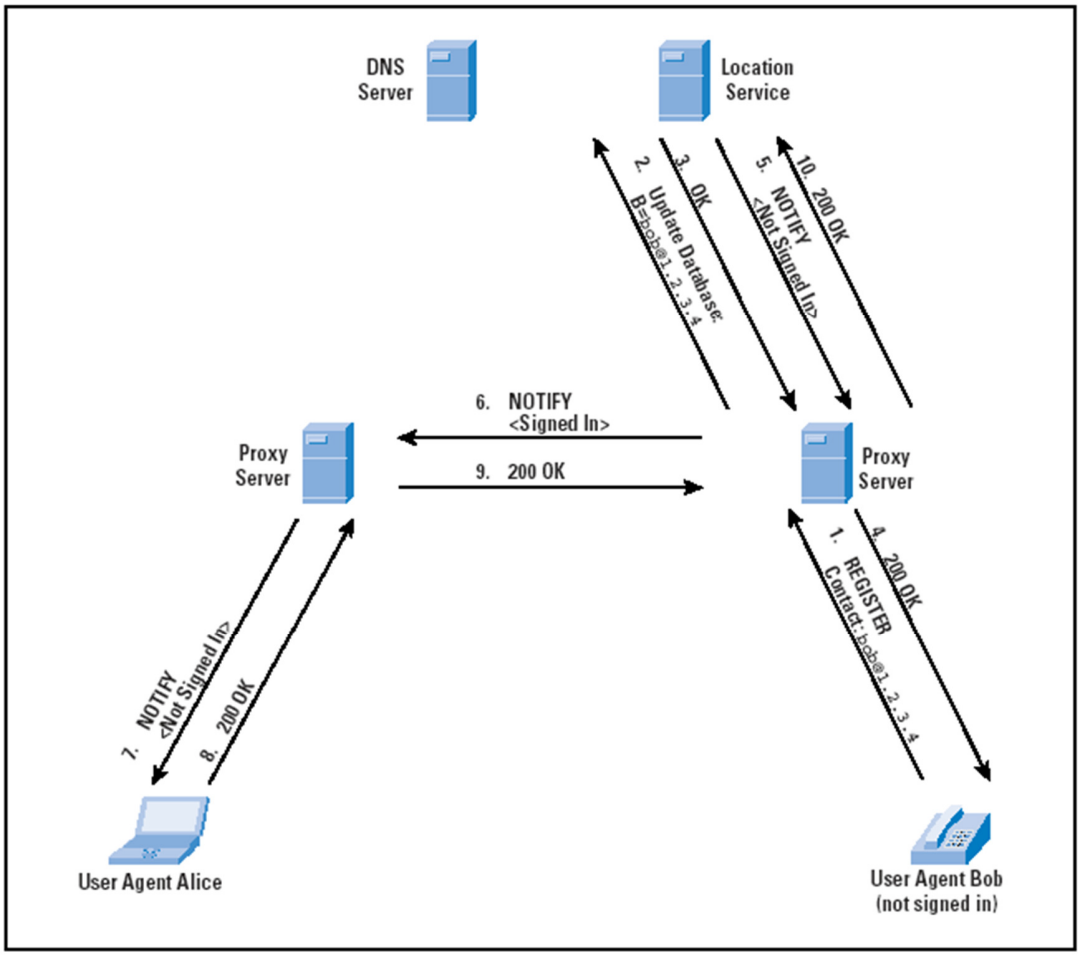


Figure 4: SIP Registration and Notification Example

1.2.4.3 SIP Messages

As was mentioned, SIP is a text-based protocol with a syntax similar to that of HTTP. There are two different types of SIP messages, requests and responses. The format difference between the two types of messages is seen in the first line. The first line of a request has a method, defining the nature of the request and a Request-URI, indicating where the request should be sent. The first line of a response has a response code. All messages include a header, consisting of a number of lines, each line beginning with a header label. A message can also contain a body such as an SDP media description. For SIP requests, RFC 3261 defines the following methods:

REGISTER: Used by a user agent to notify a SIP configuration of its current IP address and the URLs for which it would like to receive calls

INVITE: Used to establish a media session between user agents

ACK: Confirms reliable message exchanges

CANCEL: Terminates a pending request, but does not undo a completed call

BYE: Terminates a session between two users in a conference

OPTIONS: Solicits information about the capabilities of the callee, but does not set up a call

1.3 Private Branch Exchange (PBX)

1.3.1 What is a PBX

A **PBX** (Private Branch Exchange) is a system that connects telephone extensions to the Public Switched Telephone Network and provides internal communication for a business. An **IP PBX** is a PBX with Internet Protocol connectivity and may provide additional audio, video, or instant messaging communication utilizing the TCP/IP protocol stack.

VoIP gateways can be combined with traditional PBX functionality to allow businesses to use their managed intranet to help reduce long distance expenses and take advantage of the benefits of a single network for voice and data (converged network). An IP PBX may also provide CTI features.

An IP PBX can exist as a physical hardware device or in software.

1.3.2 Functions of a PBX

Functionally, the PBX performs four main call processing duties:

- Establishing connections (circuits) between the telephone sets of two users (e.g. mapping a dialed number to a physical phone, ensuring the phone isn't already busy)
- Maintaining such connections as long as the users require them (i.e. channeling voice signals between the users)
- Disconnecting those connections as per the user's requirement
- Providing information for accounting purposes (e.g. metering calls)

In addition to these basic functions, PBXs offer many other calling features and capabilities, with different manufacturers providing different features in an effort to differentiate their products. Common capabilities include (manufacturers may have a different name for each capability):

Table 1: Functions of a modern VoIP PBX

• Auto attendant • Auto dialing • Automated directory services (where callers can be routed to a given employee by keying or speaking the letters of the employee's name) • Automatic call distributor • Automatic ring back • Busy override • Call blocking • Call forwarding on busy or absence • Call logging • Call park • Call pick-up • Call transfer • Call waiting • Camp-on • Conference call • Custom greetings • Customized abbreviated dialing (Speed dialing) • Direct inward dialing (DID) • Direct inward system access (DISA) (the ability to access internal features from an outside telephone line) • Do not disturb (DND)	• Follow-me, also known as find-me: Determines the routing of incoming calls. The exchange is configured with a list of numbers for a person. When a call is received for that person, the exchange routes it to each number on the list in turn until either the call is answered or the list is exhausted (at which point the call may be routed to a voice mail system). • Interactive voice response • Local Connection: Another useful attribute of a hosted PBX is the ability to have a local number in cities in which you are not physically present. This service essentially lets you create a virtual office presence anywhere in the world. • Music on hold • Night service • Public address voice paging • Shared message boxes (where a department can have a shared voicemail box) • Voice mail • Voice message broadcasting • Welcome message

1.4 A bit of telephone history

There was a time when switchboard operators had to operate company switchboards manually using cords. As automated switches and electronic switching systems gradually replaced the manual systems, the terms *private automatic branch exchange* (PABX) and *private manual branch exchange* (PMBX) differentiated them. Solid-state digital systems were sometimes referred to as *electronic private automatic branch exchanges* (EPABX). As of 2016, the term PBX is by far the most widely recognized. The acronym now applies to all types of complex, in-house telephony switching systems.

Two significant developments during the 1990s led to new types of PBX systems. One was the massive growth of data networks and increased public understanding of packet switching. Companies needed packet-switched networks for data, so using them for telephone calls proved tempting, and the availability of the Internet as a global delivery-system made packet-switched communications even more attractive. These factors led to the development of the voice over IP PBX, or IP-PBX.

The other trend involved the idea of focusing on core competence. PBX services had always been hard to arrange for smaller companies, and many companies realized that handling their own telephony was not their core competence. These considerations gave rise to the concept of the hosted PBX. In wireline telephony, the original hosted PBX was the Centrex service provided by telcos since the 1960s; later competitive offerings evolved into the modern competitive local exchange carrier. In voice over IP, hosted solutions are easier to implement as the PBX may be located at and managed by any telephone service provider, connecting to the individual extensions via the Internet. The upstream provider no longer needs to run direct, local leased lines to the served premises. once the advent of Internet telephony (Voice over IP) technologies, PBX development has tended toward the IP PBX, which uses the Internet Protocol to carry calls. Most modern PBXs support VoIP. ISDN PBX systems also replaced some traditional PBXs in the 1990s, as ISDN offers features such as conference calling, call forwarding, and programmable caller ID. As of 2015 ISDN is being phased out by most major telecommunication carriers throughout Europe in favor of all-IP networks, with some expecting complete migration by 2025. Originally having started as an organization's manual switchboard or attendant console operated by a telephone operator or just simply the *operator*, PBXs have evolved into VoIP centers that are hosted by the operators or even manufacturers.

Even though VoIP is considered the future of telephony, the circuit switched network remains the core of communications, and the existing PBX systems are competitive in services with modern IP systems. Five distinct scenarios exist:

- Hosted/virtual PBX (hosted and circuit-switched) or traditional Centrex
- IP Centrex or hosted/virtual IP (hosted and packet-switched)
- IP PBX (private and packet-switched)
- Mobile PBX solution (mobile phones replacing or used in combination with fixed phones)

- PBX (private and circuit-switched)

For the option to call from IP network to the circuit-switched PSTN (SS7/ISUP), the hosted solutions include interconnecting media gateways.

1.5 Goals and Objectives of this thesis

The goal of this thesis is to make a web based skype out like communication system that it will be served as a service and not as a software. Meaning that in order to communicate there will be no need to install anything. Just point the browser in the URL and 'as magic' the user will have a full-fledged communications platform. Also we will demonstrate how to connect a web page to the plain old telephone network, the difficulties and complexity this approach has and the potential to be a market changer! Also we will explore and implement ideas that companies can use for the costumers to reach them with, literally, a click of a mouse.

Also we will see various ways to get an estimate of the call quality and attempt to do the same with our implementation.

1.6 The Potential of this technology

Imagine how scared communications providers were when skype surpassed a hundred million users in the few years after its 2003 launch. Telecommunications companies large enough to be a country were scared of losing their bread and butter. WebRTC is part of an open protocol, open source, open standards response to 'Skype like applications' and the Internet's version of Too Big To Fail. It has the potential to change the paradigm of the world communications in a degree not easily imaged. The fact that it is breaking the monopoly held over by Microsoft and Google telephone 'web services' is enough to understand the stakes. Also the fact that it is inherently secure at least in the transport layer is of course a very big deal by itself. For these reasons WebRTC is a disruptive innovation because it creates a new market and value, displacing market leaders – such as an early example would be telephony as we know it was a disruptive innovation for telegraphy.

1.7 The need for web browser based communications

1.7.1 Why web browser based communications

Currently, a web browser has become the primary tool for user access to information. The web based technology originally had a very limited functionality, orientated on request response model and transmission of text data over HTTP protocol. An increase of end user communications has inevitably led to the need for implementation of communications in the browser. Also development of technologies for fast data transfer has resulted in an immense improvement of quality of communications through the internet and in particular the transition from text based communication to voice based communications and then to video based communications. Such communications in the browser have made possible by the several technologies. The use of various web browser plug-ins, such as Adobe Flash, as well as further implementation of direct support for streaming data by WebRTC. As our world becomes more and more interconnected we strive for more access and ease of use regarding our communications. Devices and technologies these days enable this to be exceptionally true. Mobile phones have reached computing power well above a PC would have a decade ago and 4G networks make high speed affordable internet within reach.

1.7.2 Unified Communications

This thesis is a small brick of the so called unified communications framework. This strives for a user to have a single reference to be contacted from for all his communications needs and not multiple as the current day to day person has. For example, instead for having a landline a mobile phone and a home phone, a fax machine and email account and different ways to differentiate between them, the user would have a simple way to give his credentials to someone and that someone would have been able to contact him regardless location and situation.

2 WebRTC

2.1 What is WebRTC?

The WebRTC initiative is a project supported by Google, Mozilla and Opera, amongst others. It is actively developed and maintained and its goal is to provide peer to peer RTC (Real Time Communications) in the web browser. One of the most desired use cases is the ability for audio and video conferences inside the browser without any plugin, which will be enabled by WebRTC. Other applications inside the browser depending on a real time communication like financial monitoring, peer to peer networks, games or device monitoring will also benefit from it. WebRTC enables peer to peer communication but WebRTC has to have servers for clients to exchange data in order to coordinate communication (signaling) and to find ways to work over network address translators (NATs) and firewalls. Bellow we can see a data flow diagram of how WebRTC architecture is laid out.

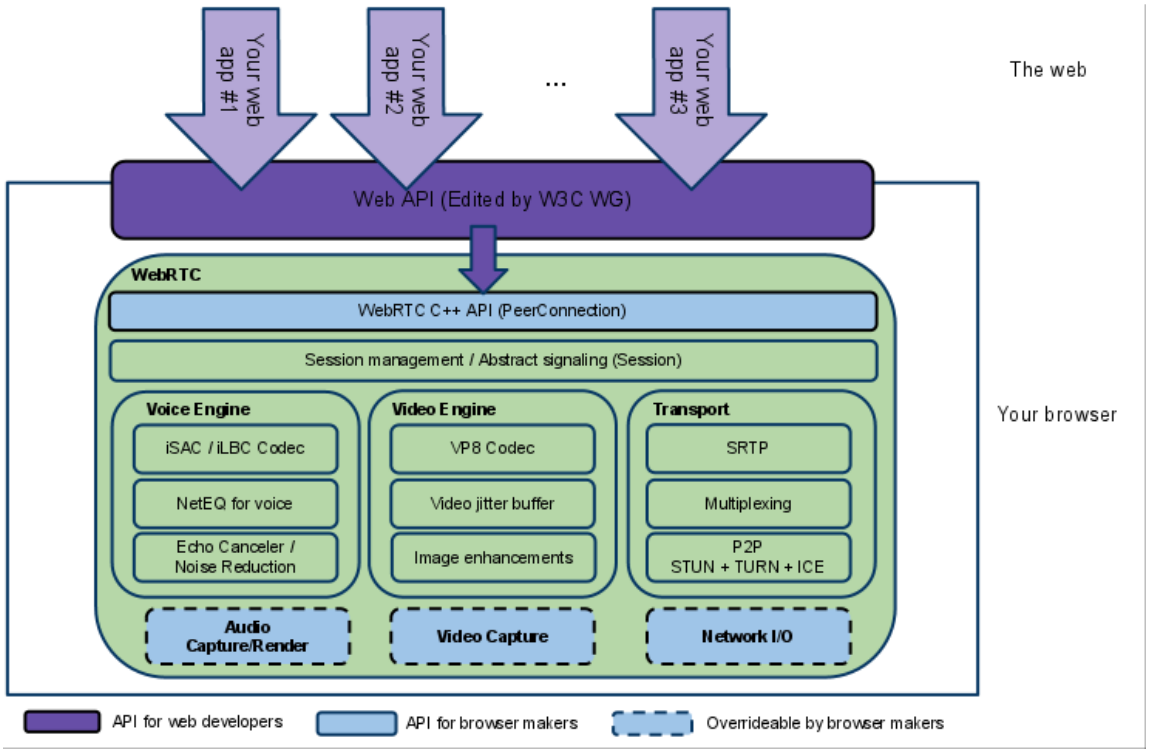


Figure 5: Data flow diagram of the WebRTC architecture

In Figure 6 we can see the data path and overal way two peers will be exchanging data. It is quite similar to the VoIP signaling model discussed in SIP in page 6. This is why signaling transportation mechanisms are out of the scope of the WebRTC project. Any well-known protocol can be used for signaling for as WebRTC cares. So the WebRTC capable device signals that wants to communicate with a peer. Then throught intermidiaries a path is established and then the media flow begins, throught the peers, not involving the apps if not nessesarry. This model has a lot of benefits as we will se in the next chapter.

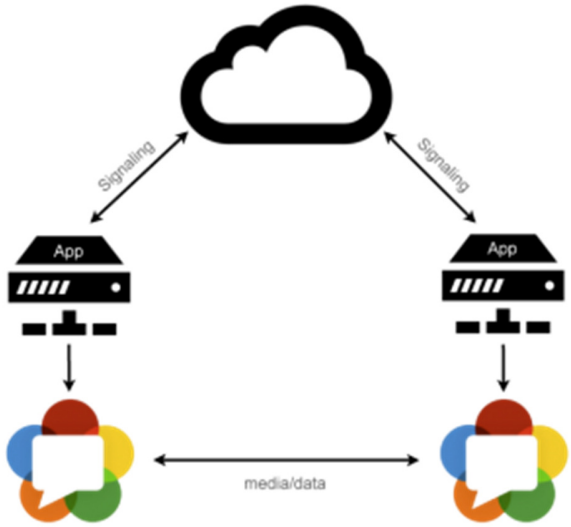


Figure 6: WebRTC Web Triangle

2.2 Benefits from WebRTC

WebRTC makes possible all kinds of real time communication with audio, video and text between users by utilizing the devices browser. Using WebRTC has different benefits for different market segments. For the end users it has two major advantages. Firstly, is ease of use. Real time communication is supported without the need for additional applications or plugins and it is provided through comprehensive APIs. The second one is security. WebRTC makes mandatory the usage of encryption for both the media and the signaling. Therefore, WebRTC provides a higher security level than most currently public commercial telephony systems.

For enterprises WebRTC can provide a lot of benefits including at least without the need for special applications cost savings on the costs of toll free telephone number for call centers and also to enrich communication. Enhance the communication to users and employers without the

need to have and deploy special applications and servers. Also uninterrupted communication: Hold the customers on the web site and at the same time start a video call with customer. In the mean time you are securing the communications with the customers as well as the employees that are in the home office and-or remote offices using state of the art encryption standards.

2.3 WebRTC APIs

2.3.1 GetUserMedia API

WebRTC applications use the GetUserMedia API [11] is present to allow access to media streams that come from local devices such as video cameras and microphones. The MediaDevices.getUserMedia() method prompts for permission to use one video and/or one audio input devices. If the user provides the permission, then it returns a promise object with the resulting MediaStream object or objects (audio and video or just audio). If the user denies that permission, or a media device is not available, then it is rejected as a PermissionDeniedError or NotFoundError. We can see the textbook example of the getUserMedia function below:

```
var p = navigator.mediaDevices.getUserMedia(constraints);

p.then(function(mediaStream) [
 var video = document.querySelector('video');
 video.src = window.URL.createObjectURL(mediaStream);
 video.onloadedmetadata = function(e) [
 // Do something with the video here if needed.];]);

p.catch(function(err) [ console.log(err.name); ]);
```

We can see that the getUserMedia function has is a MediaStreamConstaints (underscored code) object which has two inputs: video and audio and the description of the media types that are requested. If the browser fails or cannot find all the media tracks with the types that meet the constraints then it returns a promise with NotFoundError. As an example below we request both audio and video without any requirements:

```
[ audio: true, video: true ]
```

While the specifications about the user's cameras and microphones are inaccessible to the browser for privacy reasons but we can request the camera and microphone for capabilities as needed using additional constraints. The following is a constraint for 1280x720 camera resolution:

```
Constraints = [ audio: true, video: [ width: [ min: 1280 ], height: [ min: 720 ]]
```

The browser will try to make this work, but also may return another resolution if an exact match does not exist or the user sets it otherwise. To require a certain capability, we can use the keywords such as min, max, or exact.

2.3.2 PeerConnection API

WebRTC needs an API to provide the networking support to transfer all that media and data we want to the other peers The PeerConnection API as it is called has the methods and mechanisms that enable this transfer, at the same time it also handles all the signaling messages. Also SDP messages are sent to provide media and NAT reversal negotiation between two different endpoints prior to establish data transmission. This protocol is used in a modified version that allows for the usage of multiple media descriptions over a single set of Interactive Connectivity Establishment (ICE) [12]. This feature is described as Bundle [13] and can be used along with the existing SDP Offer/Answer mechanism to negotiate the different media on any given session. Thus by using Bundled SDP we multiplex all the traffic using a one single port, so the media, data and monitoring messages are sent over just one port. On the other side, signaling is not standardized as we see in "**Error! Not a valid bookmark self-reference.**" and has to be provided by the developer.

2.3.3 Data channel

So, someone could ask, we have AJAX, Server Sent Events and WebSocket. Why do we need yet another communication channel? WebSocket has bi-directionality, but all these technologies are designed for data transfer to and from a server. RTCDataChannel takes a different approach on this issue. That is because it works with the RTCPeerConnection API, which uses peer to peer connectivity. This has the advantage of lower latency because there are no intermediary server in the loop so fewer hops. And RTCDataChannel uses SCTP (Stream Control Transmission Protocol), allowing configurable delivery semantics like out-of-order delivery and retransmit configuration. RTCDataChannel is available with SCTP support, in Chrome, Opera and Firefox for both desktop and Android.

The RTCDataChannel API supports a flexible set of data types. The API is designed to mimic WebSocket exactly, and RTCDataChannel supports strings as well as some of the binary types in JavaScript such as Blob, ArrayBuffer and ArrayBufferView. These types can be helpful when working with file transfer and multiplayer gaming.

RTCDataChannel can work in either unreliable mode (analogous to User Datagram Protocol or UDP) or reliable mode (analogous to Transmission Control Protocol or TCP). The two modes have a simple distinction:

- **Reliable mode** guarantees the transmission of messages and also the order in which they are delivered. This takes extra overhead, thus potentially making this mode slower.
- **Unreliable mode** does not guarantee every message will get to the other side nor what order they get there. This removes the overhead, allowing this mode to work much faster.

Table 2: Transmission modes comparison table

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

Performance for both modes is about the same when there are no packet losses. However, in reliable mode a lost packet will cause other packets to get blocked behind it, and the lost packet

might be stale by the time it is retransmitted and arrives. It is, of course, possible to use multiple data channels within the same application, each with their own (un)reliable semantics.

2.3.4 RTCNinja API

WebRTC API wrapper to deal with different browsers transparently, as we see in eventually this library shouldn't be needed. We only have to wait until W3C group in charge finishes the specification and the different browsers implement it correctly

The main API calls and classes and functions for RTCNinja are:

- rtcninja.hasWebRTC()
    - Returns `true` if the browser supports WebRTC.
- `rtcninja.getUserMedia(constraints, successCallback, errorCallback)` function
    - Provides a wrapper over the native `navigator.(webkit|moz)getUserMedia()` function. As a feature, if WebRTC is not supported this function fires the given `errorCallback` instead of throwing an error
- rtcninja.RTCPeerConnection class
    - Provides access to the rtcninja.RTCPeerConnection class, which wrappes a native (webkit|moz)RTCPeerConnection.
- rtcninja.RTCSessionDescription class
    - Wrapper for the native RTCSessionDescription class.
- rtcninja.RTCIceCandidate class
    - Wrapper for the native RTCIceCandidate class.
- rtcninja.MediaStreamTrack class
    - Wrapper for the native MediaStreamTrack class.

`rtcninja.attachMediaStream(element, stream) function`

- Sets the given `stream` (of type `MediaStream`) as the source of the `<video>` or `<audio>` element pointed by `element`. Returns the `element` itself.

`rtcninja.closeMediaStream(stream) function`

- Closes the given `stream` (of type `MediaStream`).

`rtcninja.canRenegotiate attribute`

Boolean indicating whether SDP renegotiation is properly supported by the current WebRTC engine.

2.4 WebRTC Signaling

2.4.1 Introduction to WebRTC Signaling

Signaling is the process of sending control information between two points that want to communicate in order to make agreements and to determine the communication protocols, media codecs and method of data transfer, encryption as well as any required routing information. But the most important thing to know about the signaling process for WebRTC: it is not defined in the specification.

You may wonder why is something as fundamental to the process of establishing a WebRTC connection left out of the WebRTC specification. Since the two devices have no way to directly contact each other, how the specification will predict every possible use case for WebRTC? It makes more sense to let the programmer select a signaling technology and protocol.

To be more accurate, if a developer already has a working method in place for connecting two devices, it is not practical for them to have to use another one even if it is defined by the specification, just to have WebRTC. Since WebRTC doesn't live on its own, there is likely other connectivity in play, so it makes sense to avoid adding additional connection channels for signaling if an existing system can be used.

To continue in order to exchange signaling data, you can choose amongst a plethora of different signaling protocols. You can send JSON objects over a WebSocket connection. You can use XMPP or SIP or you could use XMLHttpRequest over HTTPS with polling and/or any other combination of technologies you can come think of. You could even use email as the signaling channel as well.

It's also worth taking into account that the channel for doing the signaling doesn't even need to be over the network. One end point can output a data object that can be printed out, physically carried to another end point, entered into that end point, and a response then output by

that device to be returned by car, and so forth, until the WebRTC peer connection is established. It'd be very high time to establish a connection but it could be done. It is called out of channel authentication.

2.4.2 Information exchanged during signaling

There are three basic types of data that need to be exchanged during the signaling phase:

- Control messages are used to set up, open and close the channel, and to handle any errors.
- Information needed in order to establish the connection: the IP address and port number are needed by the end devices to be able to talk to one another.
- Media capability negotiation: find out what codecs and media formats can the end devices understand? These need to be agreed upon before the WebRTC session can begin.

When the signaling phase has been successfully completed we start can the true process of opening the WebRTC peer connection.

Also it's worth noting that the signaling server does not have to understand or do anything with the signaling data between the two peers during signaling. The signaling server is in a way a relay: a common point of reference on which both sides connect to having that their signaling data can be transmitted through it. The server doesn't need to interact to this information in any way.

Which signaling to use for WebRTC is a controversial topic. It is mainly dichotomies in two groups each with its advantages:

2.4.3 WebRTC has to have Directory Services

One element native WebRTC does not have is a directory service. A DS is necessary so that WebRTC users can find each other or so that they may interact with someone who is on a plain old telephone or a mobile phone. There are cases that directory services will be provided by interfacing with a web site’s authentication system or with an existing enterprise class directory. Directory information can then be passed down to the browser GUI, allowing users to find and communicate with each other.

Another scenario would be a customer service web site that has an interface to a help desk. In this scenario, the user who is browsing the website does not have to authenticate, only the

contact center agent has to authenticate. The application service then can automatically create the correlation between the probable customer and a contact center agent. This communication can be a direct WebRTC to WebRTC session, or it can be a WebRTC - to - POTS session, depending upon what WebRTC capabilities the call center has.

2.4.4 JSON over Multiple Transports

The most intuitive for WebRTC applications to do signaling is the transmission of JSON objects over the best available bidirectional transport — WebSocket, or, some combination of COMET like mechanisms. This, for one, is the approach adopted by Google in the early and currently most popular applications.

2.4.5 JSEP protocol

JavaScript Session Establishment Protocol (JSEP) [14] is in draft state . WebRTC call setup has in general the full specification and control of the media plane, but the signaling is up to the application to be handled as desired. Hence, WebRTC signaling may use different protocols, such as the existing SIP or Jingle call signaling protocols, or a custom one. In this approach, the important information that needs to be exchanged is the multimedia session description, which has the transport and media configuration information needed to establish the media connection.

The various browsers also have challenges that pose problems for signaling. One of these is that the user may reload the web page. If the browser is in charge of the signaling state, this will have as consequence the loss of the call. But if, the state can be stored at the server, and pushed back to the new page, the call can be resumed with just a small interruption.

With these things in mind, JavaScript Session Establishment Protocol (JSEP) allows for full control of the signaling state machine from JavaScript. This mechanism effectively removes the browser almost completely from the core signaling flow and the only interface needed is a way for the application to give in the local and remote session descriptions negotiated by whatever signaling mechanism is used, and a way to interact with the ICE state machine.

The way JSEP handles session descriptions is simple and straightforward. Whenever an offer/answer is needed, the side that starts the connection, creates an offer by calling a

createOffer() API. The application then can modify that offer, and then uses it to set up its local configuration by the setLocalDescription() API. The offer then can be sent to the remote side over any transport mechanism like WebSockets. Upon receipt of that offer, the remote party installs it using the setRemoteDescription() API.

When the call is accepted, the callee uses the createAnswer() API to generate an answer, applies it using setLocalDescription(), and sends the answer back to the initiator over the signaling channel. When the offerer gets that answer, it installs it using setRemoteDescription(), and initial setup is complete. This process can be repeated for additional offer/answer exchanges.

As far as ICE [12] is concerned, JSEP separates the ICE state from the overall signaling state, because only the browser has the available knowledge of candidates and other transport information. Also in protocols that decouple session descriptions from transport, such as Jingle [15], the transport information can be sent separately. In the protocols that don't, such as SIP [7], the information can be used in the aggregated form. Sending transport information separately can allow for faster ICE and DTLS startup, since the necessary roundtrips can occur while waiting for the remote side to accept the session.

Through its abstraction of signaling, the JSEP approach does require the application to be aware of the signaling process, while the application does not need to understand the contents of session. So, JSEP needs a JavaScript library that hides this complexity from the application developer. This library would implement a given signaling protocol along with its state and serialization code, presenting a higher level call-oriented interface to the application developer. For example, this library could easily adapt the JSEP API into the API that was proposed for the ROAP signaling protocol [16], which would perform a ROAP call setup under the covers, interacting with the application only when it needs a signaling message to be sent. This will allow JSEP to provide greater control for the experienced developer without having any additional complexity on the novice developer.

2.4.6 Web Sockets

The Web Sockets API[17] is a specification maintained by the W3C Working Group. Browsers typically only supported client pull mechanisms, and server push mechanisms was not directly provided to web developers. This was a major drawback, as servers were not able to just send date to clients when they had any updated information. There are been several workarounds

in achieving server push mechanisms but not a 'real' solution. Some practices of abusing the HTTP protocol such as long polling have been thought of in RFC 6202[18]. The HTML5 WebSocket API is the first standardization that aims to help browsers implement a common mechanism of server push pull. The WebSocket protocol[19] has been designed for bidirectional communication and it is done by using a single Transmission Control Protocol (TCP) connection, avoiding in essence to create a new TCP connection every time the server or client want to communicate with each other. WebSockets is protocol, and is treated as a HTTP upgrade during the handshake procedure. Just like HTTP, it works on the default port 80, and for secure connection it on port 443 which is based on the Transport Layer Security (TLS) protocol [20]. Unlike most of the HTML5 APIs, the WebSockets API requires support from the server as well. Hence it is not enough if the browser alone implements the WebSockets. There are a number of server side implementations of the Web Socket protocol in most of the well-known languages like C sharp, Python, Java and even JavaScript. It, might be a bit surprising to know that JavaScript is used outside the browser, however this has been around for some time now (in fact it has been available soon after JavaScript was released for the browser in 1994). Node.js3 is one recent notable example of a server-side implementation of JavaScript.

2.5 WebRTC Security

Lessons that we have learned all those years from communication protocols – networking and social engineering the teams that developed the WebRTC have come up with a set of guidelines for the implementations in the various browsers. Those are not followed by all the browsers and are open to some interpretation but at least we see that there is a convergence within the community in most aspects of this, admittedly very important subject.

2.5.1 Trust Model

The first and foremost thing in the WebRTC security model is the browser as also stated by RFC draft[21]. The browser acts as the Trusted Computing Base (TCB) and is the only piece of the system user can really trust. Its job is to enforce user's desired security policies and to implement all the cryptography with whatever this implies. Also to authenticated entities such as

- Identity providers

- Other users (when cryptographically verified)
- Calling services (known origin)
- Sometimes network elements with the right topology (e.g., behind our firewall)

Authenticated does not mean trusted! But authentication is the basis of trust decisions

2.5.2 Same Origin Policy

The resources that are accessible need to be isolated. Scripts are allowed to make HTTP requests but those requests are not allowed to be made to any other server but only to the same origin from whence the script came [22]. Web Sockets provide an escape from this restriction.

The same origin policy (SOP) prevents server from making attacks on another server using the user's browser, which protects both the user and the server B. In general, same origin policy forces scripts from each site to run in their own sandboxes.

2.5.3 Permissions Models

Because no one wants to be seen or heard from his webcam without positive consent, one-time camera and or microphone access is always asked from the user. A permanent camera and or microphone access or user based permissions are not mandatory in this model. Data channels in the other hand may be created without any user consent. So consent for device access is mainly a matter of protecting the user's privacy from pernicious sites.

2.5.4 Permissions API

Browsers have to provide a mechanism to distinguish permissions type such as a new PeerConnection or getUserMedia and to display different UIs for each permissions level. Also it has to provide a mechanism to renounce any media stream access. It must make a site to commit not to observe your data and all this has to be implemented in a trusted UI. The Permissions UI has to clearly indicate when the camera/microphone are in use all the times and it should stop camera and microphone access when UI indicator is being masked, e.g. from a window overlap. Also the UI can provide a distinctive UI when user's identities are directly verifiable.

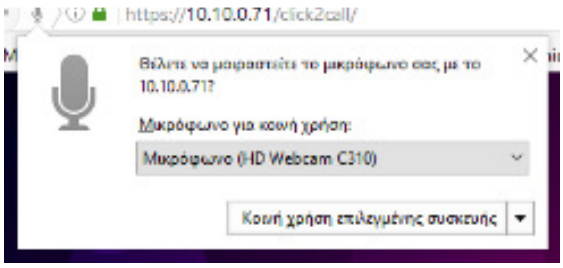


Figure 7: Part of the GUI permission user input

2.5.5 Communications Security

This is a problem all too known from the SIP world. For obvious reasons, it has to be possible for the communicating parties to establish a communications channel which is secure against message recovery and message modification. This has to be provided for both data and media (voice/video). Technology for providing this are, SRTP [23], DTLS [24] and DTLS SRTP[25]. It is paramount to understand that unlike a standard SIP proxy, the calling browser controls the channel between the communicating endpoints and also the application running on the user's browser.

2.5.6 Web Security Issues

Browsers must treat HTTP and HTTPS origins as different permissions domains. Mixed content must not be treated as if it were from the HTTPS origin. Furthermore, there is the question of where JS libraries come from. Are they trusted and by whom, so to be a continuation to the trust chain. The standard thing to do is to download from a CDN, like so

```
 <script
crc="http://ajax.googleapis.com/ajax/libs/jquery/1.7.0/jquery.min.js">
```

At minimum then, you want a HTTPS connection (not all CDNs do this) so the CDN is now inside a security boundary.

2.5.7 IP Location Privacy

An unwanted side effect of the ICE behavior is that the other peer learns the other peers IP address, which in turn leaks location information. This has privacy concerns and consequences in some

circumstances. The API has to allow suppression of ICE negotiation until the user accepts the session. It also must provide a mechanism to do TURN only candidates and perhaps to allow conversion to ICE candidates once peer identity is verified.

2.5.8 Communications Security: Implementation

Of course being in essence an all-around communication solution, WebRTC team is not going to leave some key factors in their own. There are RFCs that state minimums in those key components and functions of the APIs. We will list the ones we feel are the major ones: i) The browser must support a baseline audio and video codec. ii) It must be possible to protect streams and data from wiretapping [26, 27]. iii) The browser must encrypt, authenticate and integrity protect streams and media and data on a per IP packet basis, and must drop incoming media and data packets that fail the per IP packet integrity check. iv) Also the browser has to have a way to support for cryptographically binding streams and data security keys to the user identity [28].

2.6 WebRTC handling NAT and Firewall Traversal

2.6.1 The different types of NAT

There are four types of NATs in today's infrastructure, presented below in order from least restrictive to most restrictive (i is for internal, e is for external and "Any" means the port number doesn't matter):

Full cone type NAT: is once an internal address (iAddress:iPort) is mapped to an external address (eAddress:ePort), any packets from iAddress:iPort will be sent through eAddress:ePort. Any external host can send packets to iAddress:iPort by sending packets to eAddress:ePort.

Address restricted cone type NAT: is once an internal address (iAddress:iPort) is mapped to an external address (eAddress:ePort), any packets from iAddress:iPort will be sent through eAddress:ePort. An external host (hAddress:any) can send packets to iAddress:iPort by sending packets to eAddress:ePort only if iAddress:iPort has previously sent a packet to hAddress:any.

Port restricted cone: (like addressrestricted cone, but the restriction includes port numbers) cone type NAT is once an internal address (iAddress:iPort) is mapped to an external address (eAddress:ePort), any packets from iAddress:iPort will be sent through eAddress:ePort. An external host (hAddress:hPort) can send packets to iAddress:iPort by sending packets to eAddress:ePort only if iAddress:iPort has previously sent a packet to hAddress:hPort.

Symmetric cone type NAT: they are often "enterprise" NATs that are hiding more devices than an average situation would require. Thus, their presence is significant and must be worked around. Symmetric NATs has each request from the same internal IP address and port to a specific destination IP address and port is mapped to a unique external source IP address and port. If the same internal host sends a packet even with the same source address and port but to a different destination, then a different mapping is used. Only an external host that has received a packet from an internal host can send a packet back.

The techniques needed in order to establish a direct connection between peers become more challenging as the NATed networks between them become more restrictive. In the worst case, a relay with a public IP address is needed in order to exchange packets between peers.

2.6.2 The Problem of NAT and Firewalls in VoIP

So, Network address translators (NATs) are common devices that effectively hide private networks behind public IP addresses, mostly to conserve IP namespace but also for a limited security. This has the implication that when an IP phone is located behind NAT the phone's inability to correctly understand the network environment and the NAT devices inability to understand the nature of the connection tried to be established or a combination of the two, results in a problematic situation. To further develop the problem, connections can be initiated from the private network to the Internet, but not the other way around. The situation is made worse by the fact that SIP controls separate media streams and thus transports addresses. Many firewalls only allow connections to be initiated from the private network, thus having the same effect as NATs. Also, firewalls commonly deny access to port numbers associated with VoIP. Some even inspect

the packet contents to identify and drop VoIP traffic. Result: VoIP users behind NATs and firewalls do not benefit from the end to end connectivity necessary for VoIP.

2.6.3 WebRTC connectivity

Offer/Answer and Signal Channel and Ice candidates are used in order to make connections for WebRTC. Unfortunately, WebRTC can't create connections without some sort of intermediate in the middle. This would be the the Signal Channel. It's any sort of channel of communication to exchange the necessary information before setting up a connection, whether by post card, email or even a pigeon.

Peer A who will be the initiator of the connection, will create an Offer. They will then transmit this offer to Peer B. Peer B will receive the Offer from the signal channel and create an Answer. They will then send this back to Peer A. As well as exchanging information about the media, peers must exchange information about the network configuration and connections they have. This is known as an ICE candidate and it has information about the available methods the peer is able to communicate (directly or through a TURN server).

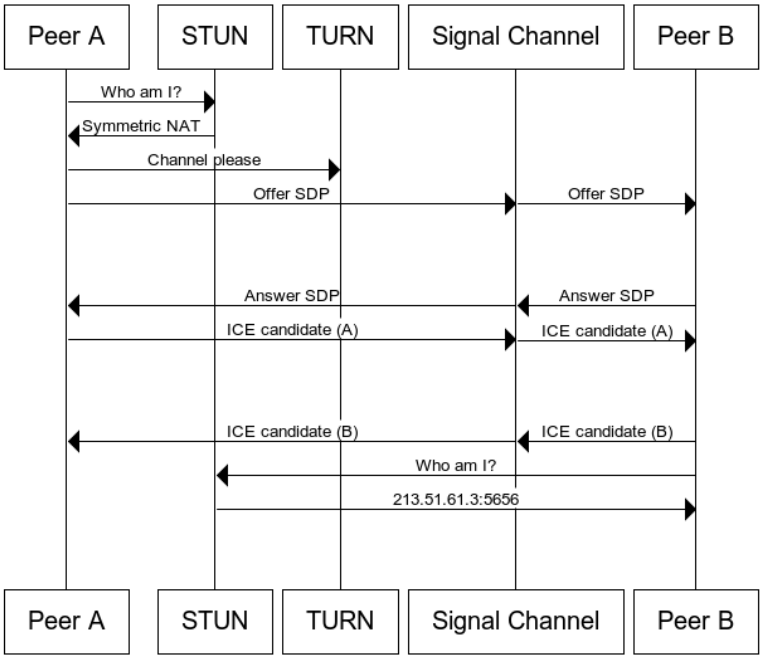


Figure 8: Typical WebRTC session

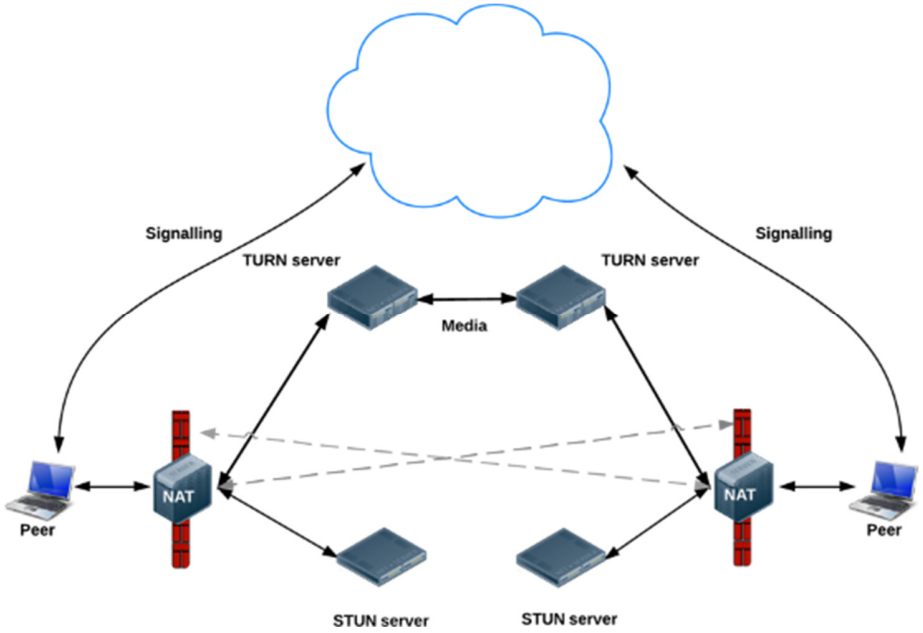


Figure 9: ICE connections

2.6.4 ICE – The NAT traversal solution for VoIP

Interactive Connectivity Establishment (ICE) [12, 29] is a very helpful framework to allow your web browser to connect with peers and coordinate to determine the best communication path between them. It provides a structured mechanism to acquire the optimal communication path for every two given peers. Session Initiation Protocol (SIP) uses extensions that are defined to enable the use of ICE protocol for setting up a call between two hosts.

There are many reasons why a connection from a Peer to another Peer won't work as we will see in "The Problem of NAT and Firewalls in VoIP". It needs to bypass firewalls that would otherwise prevent you from opening connections and give you a unique address if like most situations your device does not have a public IP address, and relay the given data through a server if your router doesn't allow you to directly connect with the end users. ICE uses some of the following techniques described in "WebRTC connectivity" to achieve this.

2.6.5 Trickle ICE

The Interactive Connectivity Establishment (ICE) protocol [12] as we saw above, gathers, candidates it prioritizing them and chooses default ones and finally exchanges them with the remote end device making pairs. Once all of the above has been done, and only then, all the parties ca select the pair of candidates that will be used for the session.

While the above sequence has the advantage of being very straightforward to implement and debug once done, it is also rather lengthy. That is because gathering candidates often involves things like asking STUN [30] servers and then discovering UPnP devices and allocating candidates at TURN [31] servers. And of course all of these is a big delay and very noticeable time and while it can be run in parallel, they still need to go along with the "pacing requirements" from [12], which is likely contribute to delays even more. So to make things worse some or all of the above has to also be completed by the remote device as well. Both of them must then perform connectivity checks and only then would they be ready to begin data transmission. No need to say again, all of the above will sometimes lead to lengthy session establishment and it degrades user experience.

So the purpose of trickle ICE is to define another 'mode' of operation for ICE, where the candidates can be sent and received incrementally. This would allow ICE users to exchange host and candidates' information as soon as a session has been initiated. Connectivity checks for a stream would also start as the first candidates for that stream are available.

Trickle ICE allows for limiting the amount of time of the establishment of connection in cases where connectivity is confirmed for the first exchanged candidates. Even when this is not the case, harvesting candidates for both agents and connectivity checks all in a parallel fashion allows to considerably reduce ICE processing times.

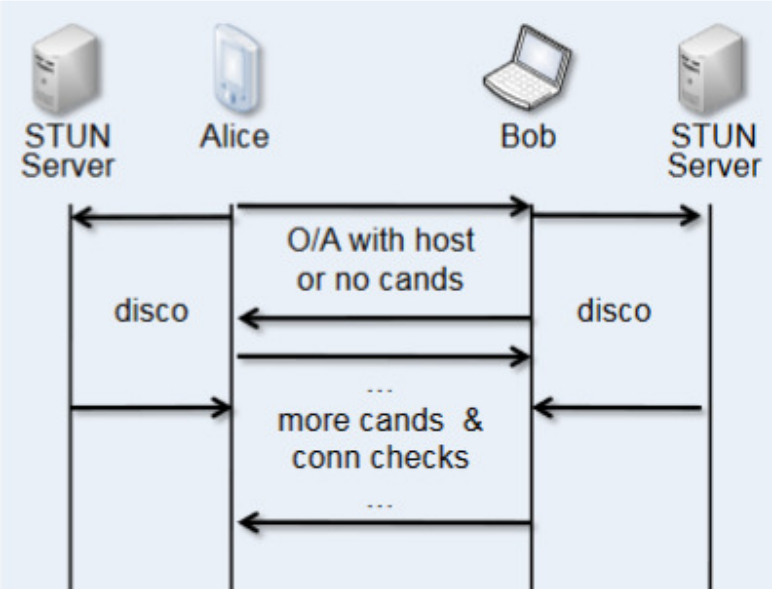


Figure 10: Trickle ICE data flow

2.6.6 STUN

STUN [30] stands for Session Traversal Utilities for NAT (acronym within an acronym) and is a set of methods to allow a host to find out its public IP address if it is located behind a NAT. This is a critical step in order to acquire the true public IP address, NAT address and of course the port number that the NAT has allocated for the application's (UDP) User Datagram Protocol connections to the remote hosts. For this protocol to work it requires assistance from a STUN server located on the public side of the NAT. It uses both TCP and UDP and for some secure applications TLS can be used also. UDP is by far the most common.

Keep in mind that STUN is not implemented in all of VoIP solutions and is not a NAT traversal solution. It can be used in order to discover candidate address – port mappings to help other protocols (ICE for instance) to determine and establish a 2-way communication path.

The method is that the client will send a request to a STUN server on the public internet who will reply with the client's public address and with the information if the client is accessible behind the router's NAT.

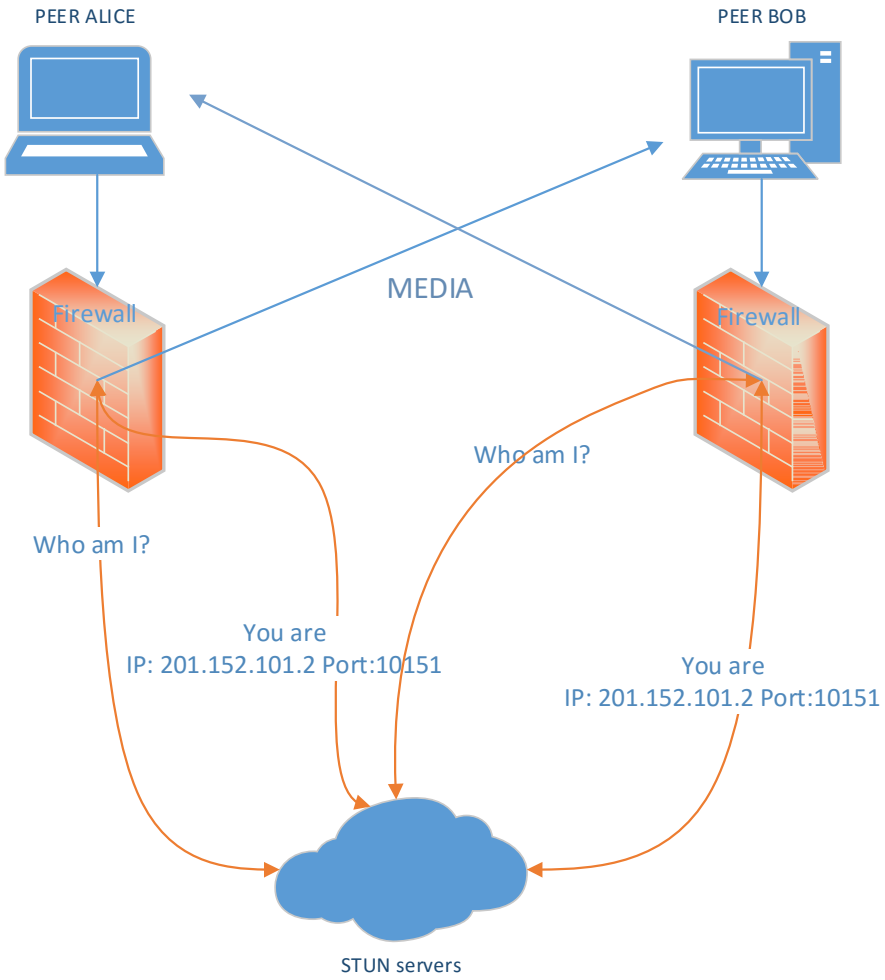


Figure 11: Stun server – client message exchanges

2.6.7 TURN

STUN provides a way to traverse a NAT which can be useful for receiving packets from a peer. However, if an address is obtained by STUN may not be usable by all peers. This address work depending on the topological arrangement of the network. So to conclude, STUN by itself is not able to provide a complete solution for NAT traversal because some routers that are using NAT employ a restriction that is called 'Symmetric NAT'. So this means the router will only accept connections from peers you have previously connected to them.

So to combat this we there is TURN [31] which stands for Traversal Using Relays around NAT. TURN offers better results with symmetric NAT than STUN does. But a complete solution needs a way by which a client can obtain an address from which it can connect and receive media

streams from any agent which can send packets to the public Internet. This can only be achieved by relaying the information through a server that resides on the public side of the Internet. To further elaborate, Traversal Using Relay NAT (TURN) is a protocol that makes it possible for a client to obtain IP addresses and ports from such a relay. This obviously comes with an overhead so it is only used if there are no other way.

While TURN always provides connectivity to a client, it is a resource intensive task for the provider of the TURN server. It is therefore a last resort to use TURN. It also adds a significant delay for real time applications, the server must remain available for the duration of the call and also for the time being is only implemented for IPv4 and UDP.

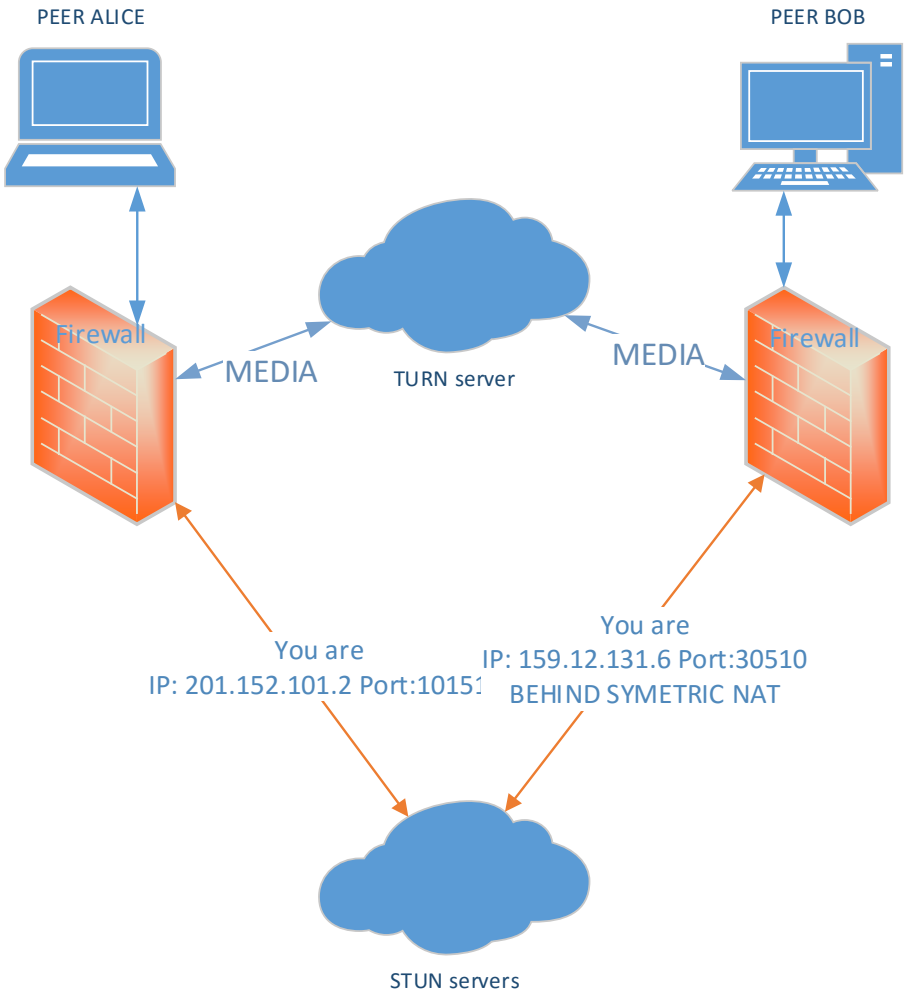


Figure 12: Turn server – client message and media exchanges

2.6.8 Other ways to traverse NAT for VoIP purposes apart from ICE

Port Forwarding: SIP devices behind NAT use ports that you may need to set to be forwarded:

- The main SIP connection port – usually this is port 5060 and/or 5061. The protocol is nearly always UDP
- The RTP media port or ports – often 10000 to 20000 UDP protocol.

One to one NAT: One to one NAT does not require any port address translation. If the VoIP phone specifies in the SIP INVITE that it will be listening for RTP on Port 15001 then you set up the NAT router to forward Port 15001 to that VoIP phone. This should work provided the external IP address is also implementing one to one NAT or another effective NAT traversal technique.

VPN: VPNs can be used not only to secure further the already safe enough (see WebRTC Security section) communication but also to completely bypass all NAT problems, since all the traffic would be tunneled from HTTPS port 433, which all firewalls and routers understand and do not touch. The downside is that VPN is an expensive process because it decodes – encodes data and adds to the latency of the system.

SIP aware firewalls and NAT devices: Strike the problem in the place it starts from. Some firewalls are SIP aware. They are configured to inspect packets and substitute the IP addresses and/or port numbers in the SIP messages to match the IP address and/or port number it is opened on the external interface of the firewall.

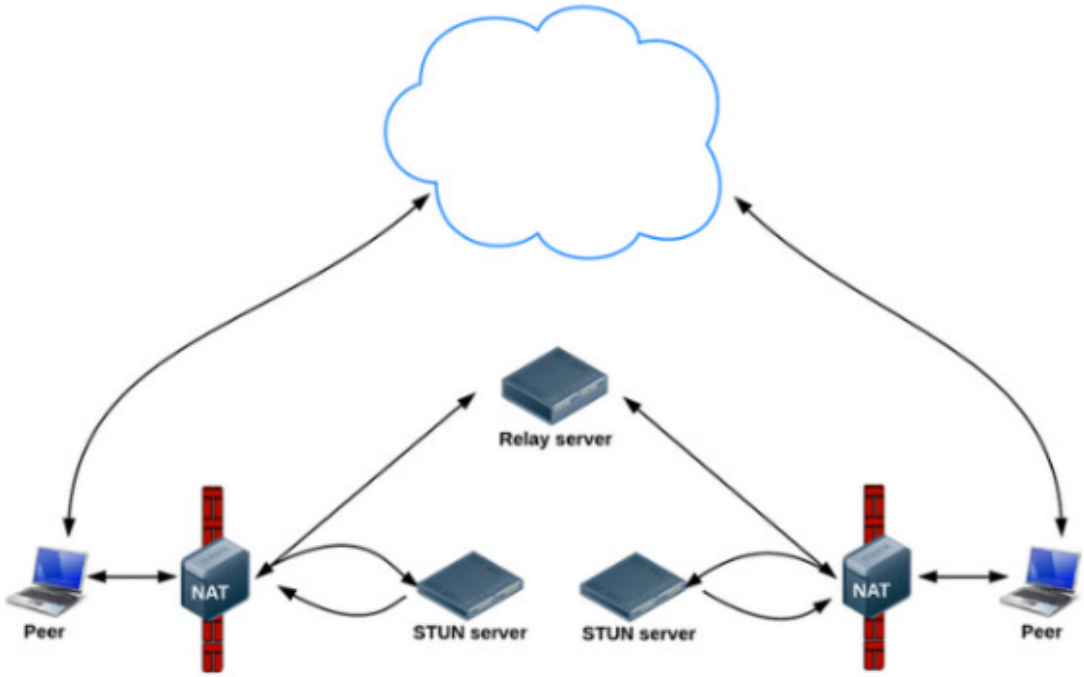


Figure 13: Nat Traversal techniques

3 Connecting Sip and Browser together

3.1 Integration of WebRTC with SIP

The Session Initiation Protocol (SIP) [7] is a crucial communication technology of the last decade. SIP brought new and standardized control mechanisms. This allows the transformation of IP networks into real multimedia communication platforms, which are now able to provide real-time communication and presence services. Wider acceptance of SIP by the internet community and telecommunication and ISP providers then has started the process of network convergence. The network convergence provides us with better and seamless integration of computer communication services with communication services of legacy telecommunication networks (telephone and mobile). SIP allows to integrate and mix together different kind of communication services into a new· kind of communication environment. The communication environment is usually accessible through feature rich terminals and computer applications.

Figure 14 illustrates the basic data paths and nodes in order a WebRTC-to-SIP connection to be established. In this scenario a webpage in a browser tries to communicate with a standard IP phone. So through JSEP a browser sends a call request to the WebRTC server, which in turn, via SIP, transfers the signaling to the Media Gateway of the VoIP network. When all the ICE paths have been determined and the best path has been chosen then the media path is created and so voice/video communications can commence.

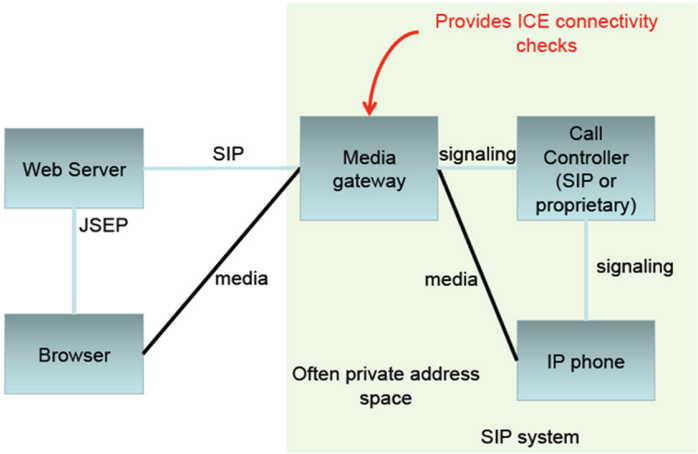


Figure 14: Integration of WebRTC with SIP

It was obvious from the start [32] that web was lacking the integration with telephony and only cumbersome and difficult proprietary technologies where available. WebRTC as was mentioned earlier, does not exactly need the SIP. but it requires a kind of signaling mechanism. The choice is up to JavaScript WebRTC application programmers. However, building a WebRTC communication environment with respects to already built “legacy” SIP systems, open an issue of WebRTC and SIP interworking. Solving issues brings the opportunity integrate SIP and WebRTC and create a kind of integrated communication environment, which allows initiate a real-time multimedia connection directly from a web browser through a WebRTC application interface. Such WebRTC application, together with backend server communication entities, allows initiate browser-to-browser as well as RTC sessions from browser to a SIP phone, communicator or legacy PSTN, mobile phones. The WebRTC application, which is a JavaScript based application, is easily integrated into any kind of web pages, web based e-learning LMS systems and etc. The whole communication environment can be easily customized following user requirements and needs using other programming APIs (HTTP or SIP APIs),

In order to be able to integrate WebRTC and SIP we need to solve several issues regarding of interworking at the media and the signaling plane. Each of planes expects different approaches. From the signaling point of view there are proposed two categories of WebRTC-SIP interworking scenarios; translation at a gateway or implementing SIP in WS JavaScript APIs [33], Here proposed and working solution is focusing on the second option, where the SIP signaling functionalities are provided through SIP WS API.

The interworking at the media plane brings several main issues as for example the media codec selection for audio/video media streams, management of media path (NAT and Firewalls), questions of different techniques of encryption and exchange of encryption keys[34],

Having a situation that you need asynchronous communication between the browser and an PBX the communication and in order to overcome the need for application specific signaling the telecommunications industry proposes a standardized approach based on the tunneling of SIP over WebSocket. SIP transports for UDP, TCP, TLS and SCTP already exist. By making the existing infrastructure accessible over WebSocket the service providers will be able to open their network to the web.

Parsing SIP messages in JavaScript is suboptimal but nonetheless open source frameworks such as sipML5 [35] and JsSIP [36] have shown that it is doable. The issues of such an approach are of different nature. Also relying just on WebSocket as a transport protocol is going to be a big obstacle for those environments where HTTP middle boxes such as corporate proxies or load balancers do not work with it. On the other hand, the SIP protocol is not designed and cannot easily adapt to make use of the Trickle ICE optimization, which as we discussed in "Trickle ICE" below, for minimizing connection establishment time. It is quite common that it can lead to delays intolerable for the end user.

To elaborate, the delays with standard ICE connectivity happen when the endpoint is configured with a few network interfaces that cannot reach the STUN and TURN servers and thus wait for the timeout of the connection. This is even more prevalent with portable devices such as smart phones that can simultaneously connect to 3G/4G Wi-Fi networks, but also sometimes with laptops running VPN or are simply configured with non-reachable IPv6 address.

Finally, we need a WebRTC client with encoded WebSocket and SIP features. These functionalities provide JavaScript libraries. There are two widespread used libraries, JsSIP [36] and sipML5 [35]. There are also others, but less used JS libraries as for example QoffeSIP and SIP-Js. Looking through the web there are several freely available WebRTC SIP clients, aka web browser RTC applications.

3.2 The JsSIP framework

JsSIP [36] is a simple to use JavaScript library which leverages latest developments in SIP [7] and WebRTC [2] to provide a fully featured SIP endpoint in any website. WebRTC enables Real-Time Communications (RTC) audio/video capabilities in Web browsers and other devices such as smartphones and that is where JsSIP stands on for its success. It provides the programmer with a very good library for developing WebRTC SIP based communication systems. It provides the SIP signaling obfuscation, in order to make it so simple that even someone who does not understand SIP can make it work. Also, it provides the shim for the getUserMedia() functions of the browser API.

With JsSIP any website can get Real Time Communications features using audio, video and more with just a few lines of code. As in the web page of the project we can see that version 2.0.x has already a plethora of features:

- Support for SIP over WebSocket transport.
- Enables Audio/video calls, instant messaging and presence.
- It is lightweight!
- It is 100% JavaScript-based from the ground up.
- Offers an easy to use, powerful user API.
- Works with OverSIP, Kamailio and Asterisk servers.

Also it supports the following SIP standards:

- RFC 3261 "SIP: Session Initiation Protocol"
- RFC 3311 "SIP UPDATE Method"
- RFC 3326 "The Reason Header Field for SIP"
- RFC 3327 "SIP Extension Header Field for Registering Non-Adjacent Contacts" (Path header)
- RFC 3428 "SIP Extension for Instant Messaging" (MESSAGE method)
- RFC 3515 "The SIP Refer Method"
- RFC 3891 "The SIP Replaces Header"
- RFC 4028 "Session Timers in SIP"
- RFC 5589 "The SIP Call Control – Transfer"
- RFC 5626 "Managing Client-Initiated Connections in SIP" (Outbound mechanism)
- RFC 5954 "Essential Correction for IPv6 ABNF and URI Comparison in RFC 3261"
- RFC 6026 "Correct Transaction Handling for 2xx Responses to SIP INVITE Requests"
- RFC 7118 "The WebSocket Protocol as a Transport for SIP"

At signaling plane (SIP protocol), JsSIP runs in any WebSocket capable browser. Figure 14 illustrates a screenshot of web page caniuse.com [37]. As we can see all major browsers support currently WebSockets.

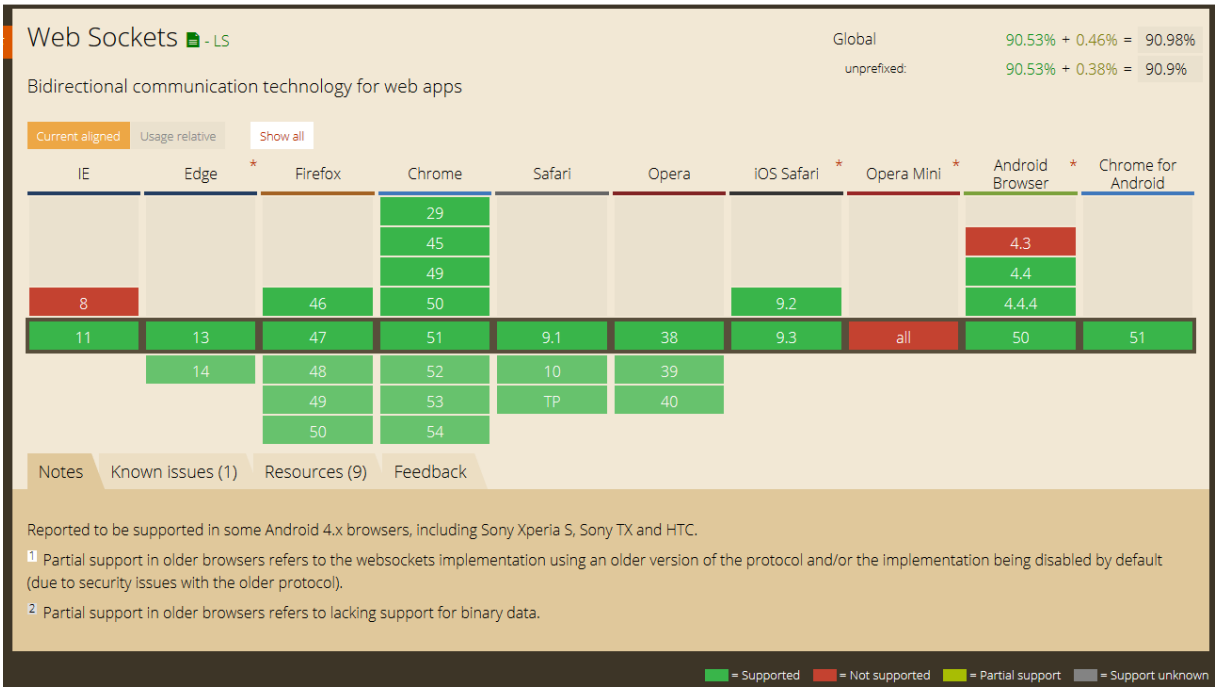


Figure 15: List of browsers that support WebSockets

3.3 JsSIP framework API explanation

In this section we will attempt to go through the API and the main and mote notable sections of the JsSIP.js use. We will avoid putting code examples in this part as we will explain the code in a different section of the thesis.

So in this JavaScript implementation of the SIP protocol there are 3 modules.

1. The rather uninteresting JsSIP module that only has one getter function and that is the version number
2. The JsSIP.debug module that you set it to enable in the start of the script and then "forget about it"
3. And the JsSIP.rtcninja module that provides the WebRTC API wrapper to deal with different browsers in a manageable way as we saw in the chosen JavaScript libraries section.

There are 9 Classes in the JsJIP API. The JsSIP.UA (User Agent) class in which there must be put initialization parameters. The mandatory parameters are the URI of the server and the URL of the web socket server. After that you can call the instance methods. The first method

typically to do first is `start()`, which registers with the server and initializes everything. Other important method is the `call( target, options)` method which makes an outgoing multimedia call. Also very important are the callback events that let the user execute functions for a given stimulus. Notably a connected event, disconnected call accepted, newRTCSession event etc.

The JsSIP.RTCSession class represents a WebRTC media session. It is the most heavily programed section of JsSIP and has a lot of functions and events. Examples of method implemented in this class are `mute(), sendDTMF()` and `terminate()`. The events callback in this class are numerus and mostly are

The class JsSIP.Registrator class manages the UA registration procedures and also you can put extra custom headers in the SIP if you so need.

The JsSIP.Message class is for a SIP based instant messaging system. It implements all the functions you would expect from a messaging API such as `sent()`, and `accept()` and if the message is incoming or outgoing.

The JsSIP.OutgoingRequest class sets the SIP request to be sent. Attributes include the call_id and from and to information.

The JsSIP.IncomingMessage class is the corresponding of the OutgoingRequest class, just for the incoming messages. Below we can see the part that JsSIP and RTCninja play in the browser and the interconnectivity between them. Also the way JsSIP connects to asterisk and finally to the outside POTS world.

4 Call Quality and Quality of Experience

4.1 Perceived call Quality

IP networks are best effort service and do not guarantee to pass information from one point to another or the do it in a timely fashion. This impairs the connection and packet loss, delay and jitter take their toll to the end to end perceived speech quality. There are QoS mechanisms and controls in the application level have been developed to minimize the effects of these problems and maximize the call quality. But in any way, considering a number of different and heterogeneous network scenarios is there any way we measure their effectiveness under certain conditions? Unlike Quality of Service (QoS) which is a somewhat well understood and established, Quality of Experience (QoE) is still a very active area of research. By the way, what really is voice quality? Defining a metric for voice quality is difficult because considering a good voice quality for one user may be just mediocre for other users, particularly in another culture or country that has linguistic and cultural differences [38, 39] . Voice quality is made up of two factors: subjective such as expectation and conversational effort and objective like hardware, software and particular network conditions and traffic. To sum up voice quality measurement is classified into two categories: subjective and objective.

Table 3: Voice Quality Measurement Type comparison

	Voice Quality Measurement Type comparisons	
	Subjective Type	Objective Type
Time consumption	Very long consumption - 5 minutes per participant	Short
Accuracy and Reliability	High	Medium - high
Management skill	High	Low
Endeavor requirement	High	Low
Special test facilities requirement	Soundproof room(s)	An objective measurement tool
Automatic acquisition of measurement	No	Yes
Collaboration requirement	High 20-50 participants	Low
Cost	High (for paying for participants and to prepare test facilities like a sound proof room	High for commercial tools low for standard tool. e.g. E- model

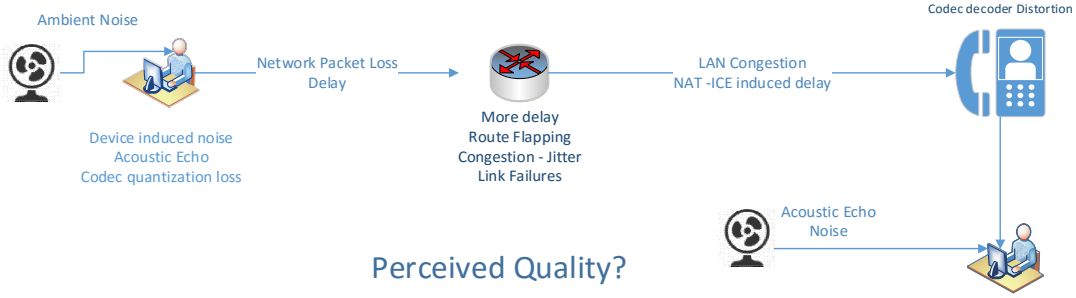


Figure 16: Simple factors that affect perceived call quality

4.1.1 Subjective quality tests

There are several challenges for evaluating performance of multimedia systems. Video streaming and VoIP traffic are mostly determined and or evaluated using a group of users at the endpoint while using the services. So subjective voice quality measurement is to quantize users perceptual call quality in telecommunications in subjective way.

It has been said by others in [39-43] that subjective voice quality measurements are the most true method. Absolute category rating (ACR) is the prevailing subjective method, while it has been stated in [44] that the conversation tests are one of the recommended methods by ITU-T [45, 46] because it can reach maximum realism covering loss and delay effects. Subjective quality measurement and or evaluation is very important because it is necessary to calibrate measurements, as shown in Figure 17 that is adopted from [47].

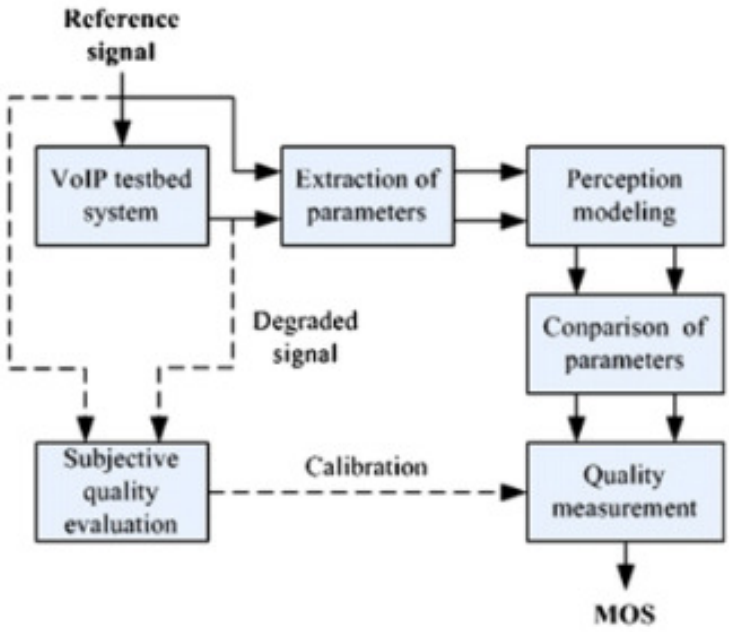


Figure 17: Measuring MOS with calibration factor

4.1.2 E-model - Objective voice quality measurement

A tool which will be used to predict subjective quality of a conversational speech quality is the ITU-T E-model. The E-Model was originally developed by ETSI [48] as a transmission planning tool, and then standardized by the ITU as G.107 [49] and recommended by the Telecommunications Industry Association [50] as "a tool which will estimate the end-to-end voice quality, taking the IP telephony parameters and impairments into account". This methodology puts together individual impairments (loss, delay, echo, codec type, noise, etc.) owing to each the signal's properties and therefore the network characteristics into one R-rating. The transmission rating issue R will he within the vary from zero to one hundred: high values of R during a range of 90 > R > 100 ought to be taken as excellent quality, whereas a lower value of R indicates a lower quality. Everything below fifty is clearly unacceptable and everything above ninety-four.15 is unobtainable in narrowband telephony. The main output of the E-model is that the transmission rating factor R. based on this factor, one will simply predict however AN "average user" would rate a VoIP decision victimization subjective MOS scores. The generally-accepted limit for high-quality voice connection delay is 150 ms and 400 ms as a most tolerable limit. If the mouth-to-ear delay exceeds outlined bounds it noticeably disrupts interactive communication. As delays rise over this figure, talkers and listeners become un-synchronized, and sometimes they speak at the same time, or each wait for the other to talk. This condition is often known as, talker overlap. though overall speech quality is acceptable, holding such a conversation will be annoying. ITU-T recommendation G.114 [51] offers the following conclusions:

- small delays (10-15 ms) don't seem to be annoying for users and no echo cancellation is needed.
- delays up to one hundred fifty ms need echo control however don't compromise the effective interaction between users
- if the delays are within the range 200 ms to 400 ms, the effectiveness of the interaction is lower however will be still acceptable
- if the delay is beyond 400 ms, interactive voice communication is troublesome or not possible and conversational rules are needed (as "over" indicators)

Talker and listener echo each contribute significantly to perceived speech quality in VoIP telephony. As a general rule, the perceived quality decreases with increasing delay and/or increasing level of the received echo signal however listener echo will be neglected if there's sufficient control of the speaker echo. The degree of annoyance of speaker echo depends on the amount of difference between the first voice and the received echo signal. This level difference is characterized by the so called "Talker Echo Loudness Rating" (TELR). Anyway, ITU-T Recommendation G.131 [52] provides helpful info concerning speaker echo as a parameter by itself.

In the literature there are several methods for producing objective voice quality measurement [53] stats. However, the E-model is the most popular non-intrusive measurement method which is consistent with the IEEE*Xplore* statistics from 2010 to April 2016 which shows the found results of 3921 [54]. This evidence shows that the E-model is in the main research direction for VoIP quality measurements[55].

The output of the E-model is a 100-point scale called R-value that is related to over twenty parameters but can be obtained from calculation using (1), before mapping into MOS, which is a 5-point scale [49], using (2)

$$R = R_o - I_s - I_d - Ie_{eff} + A \quad (1)$$

$$\text{MOS-CQE} = \begin{cases} 4.5 & ; R > 100 \\ 1 + 0.035R + R(R-60)(100-R)\,7 \times 10^{-6} & ; 0 \le R \le 100 \\ 1 & ; R < 0 \end{cases} \quad (2)$$

where R is R-value, where

Ro: is the basic signal-to-noise ratio, including noise sources such as room noise and circuit noise.

Is: is the signal impairment factor which is a combination of all impairments which occur more or less with the voice signal simultaneously.

Id: is the delay impairment factor that caused by delay.

Ie: is the affective equipment factor that caused by codecs.

A: is the advantage factor that allows for compensation of impairment factors when there are other advantages of access to the user.

and MOS-CQE is the MOS - estimated conversational quality [56].

The characteristic of R-value can also be represented as in Figure 18, whereas

Table 4 shows MOS-CQE equivalence[49] .

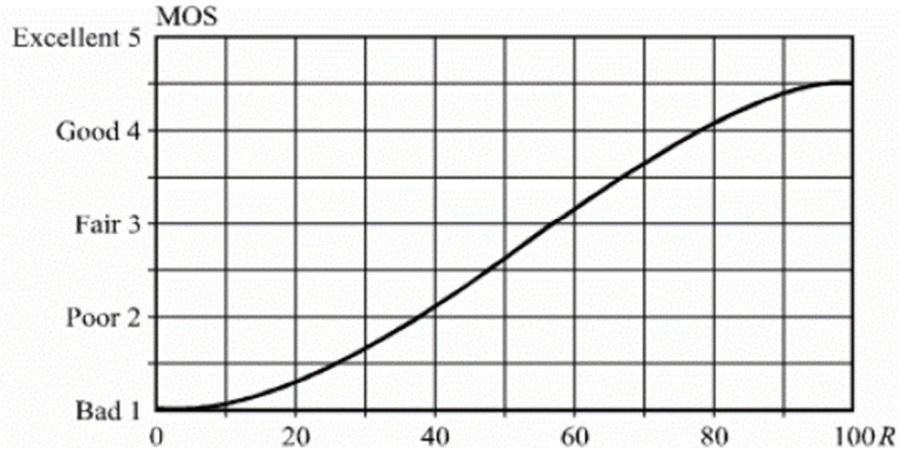


Figure 18: Relation between R-Value and MOS-CQE

Table 4: Relation among R-value, MOS-CQE and user satisfaction

R-value (lower limit)	MOS-CQE (lower limit)	User satisfaction
90	4.34	Very satisfied
80	4.03	Satisfied
70	3.60	Some users dissatisfied
60	3.10	Many users dissatisfied
50	2.58	Nearly all users dissatisfied

Although the E-model has been updated several times, ITU-T stated that it has not been verified by surveys or laboratory tests for the whole and very large number of possible combinations of input parameters [49]. It is consistent with the statements in [57, 58] that the development of the E-model is not successfully completed because the current version of the E-model does not reflect reality and pointed that the SG12 group failed to address significant factors (e.g., codec tandeming). Therefore, the improvement, and enhancement versions of the E-model have been issued as in [58-60].

Attempting to solve this QoE (Quality of Experience) problem, subjective tests were conceived for evaluating the perceived voice quality. It was called The Mean Opinion Score

(MOS). This test is a widely accepted benchmark for quantifying and quality rating purposes. In MOS tests procedures, callers rate the call quality in a scale from 1 (low quality) up to 5 (excellent quality). The number of listeners have to be enough in order to have a representative average score. This has the problem that subjective MOS tests are very time consuming, very expensive and do not allow for real time measurements [5, 61] In the present, methods for measuring MOS objectively are developed. As we saw earlier, one of them, the ITUT G.107 [61, 62]defines the E-model. This is a computational model that put together important parameters that affect a call into a factor that can be converted mathematically into a MOS scale. The (RTCPXR) Real Time Control Protocol – Extended Report, defined in RFC 3611 [63], proposes a scheme to exchange voice quality data given by the E-Model calculation. Further, the P.VTQ is being worked on by ITUT as a new VoIP voice quality measurement standard.

4.2 Problems that are affecting VoIP performance

4.2.1 Call Quality Problems

There are a number of reason that a typical VoIP call will not be optimal and of a quality we expect from a point to point network such as the plain telephone system. That is because the IP network is a best effort one, meaning that there are no guaranties for data being timely or orderly passed through the network, or that will arrive at all for that matter. Then is the codec and quantization losses from the sound being encoded to a digital format and the way the audio is presented to the listener that play a role as [64] states, the time delay between the speaker speaking and the listener receiving the communication. Also echo imported from devices and noise levels from the ambient room conditions play a major role in perceived audio quality. We will elaborate below those reasons that degrade audio quality.

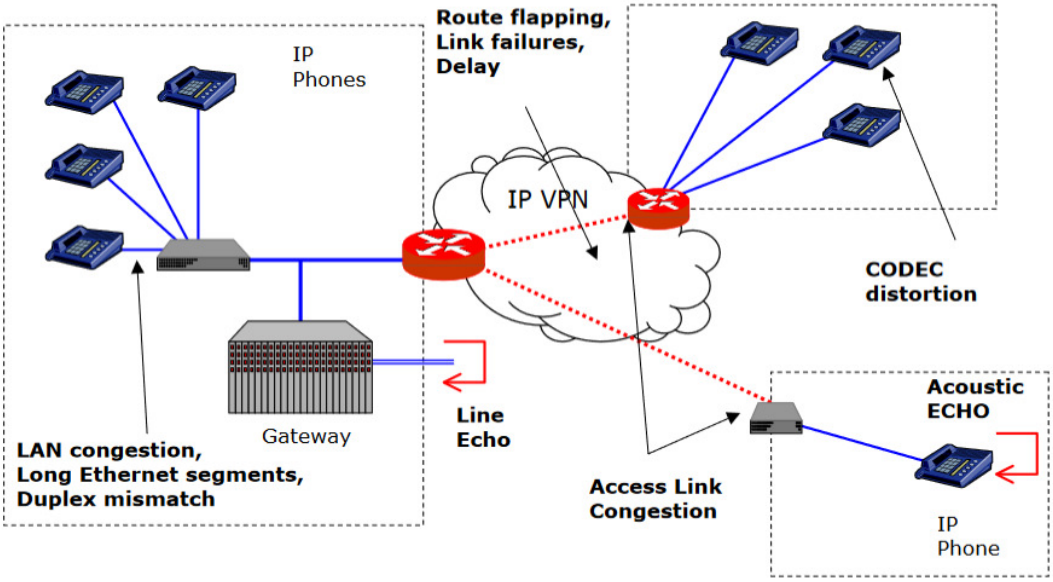


Figure 19: Potential Issues

4.2.1.1 Effects of Jitter

One of the most dreaded networking problem that effects RTC is network jitter or packet interarival time. This is caused first and foremost by the network route flapping between routes or from a congested network. As a countermeasure there are jitter buffers that will absorb low levels of jitter. But high levels of jitter will make packets being discarded and will cause the adaptive jitter buffer to get bigger thus increasing delay but reducing discards. If any packets arrive to late are discarded by the jitter buffer and are regarded as "discarded". Below we see an illustration of how a jitter buffer holds packets. The effect of the jitter discarded packets makes the audio come out as distorted and its pitch also usually changes.

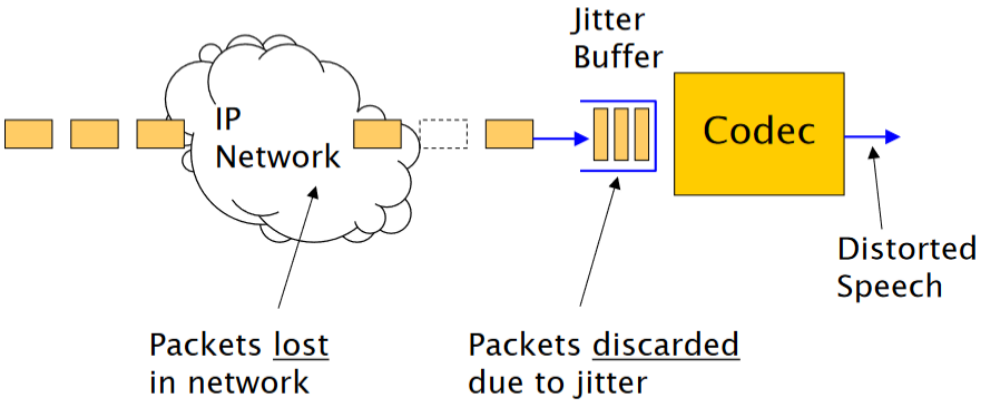


Figure 20: Packet Loss and Jitter

4.2.1.2 Loss and Discard

Packet loss is most often associated with high network congestion. Thus jitter is almost always due to congestion and leads to packets being discarded. So packet loss and discarded packets often come together. Other factors may apply such as duplex mismatch and link failures.

4.2.1.3 Effects and Interaction of echo and delay

Echo is the effect that you hear your voice back at your earphone. Its causes are mainly within the caller's phone. An amount of the speaker sound enters the microphone (acoustic

feedback) and transmitted to the other end which is perceived as echo. Echo with a bit of delay makes the call sound like a tunnel. For echo to be a problem, it needs to be loud and to have a delay from the original noise. Echo with over 60mS delay is noticeable by humans. VoIP systems do introduce a lot of milliseconds of latency into the call, and the more network hops the data go through or the greater the network congestion, the bigger the latency is. As we see below the level of echo above 55dB SNR is good and below 25dB or below is perceived as bad. We can also see the effect of RTT in this plot.

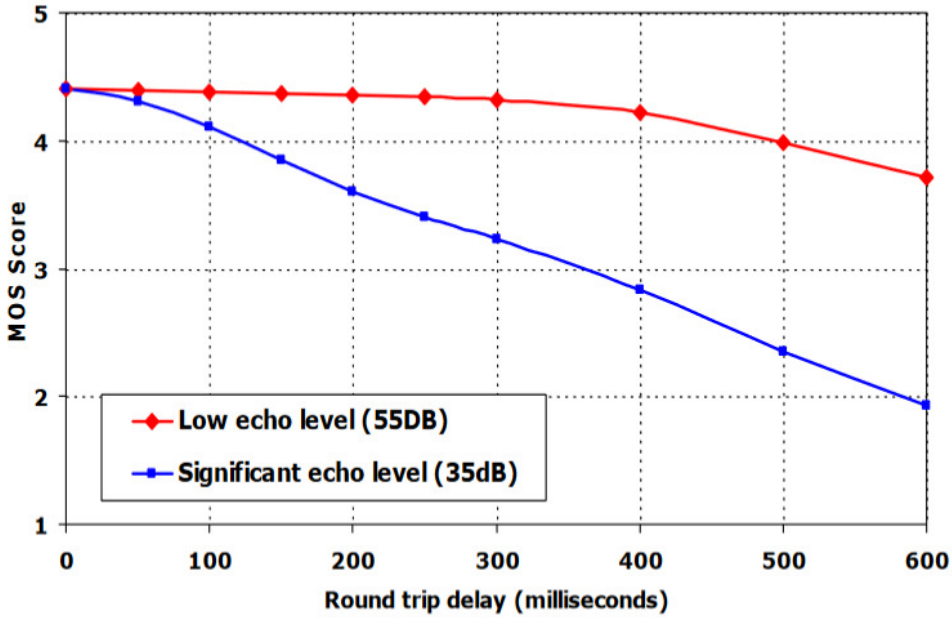


Figure 21: Effect of Delay to MOS on Conversational Quality

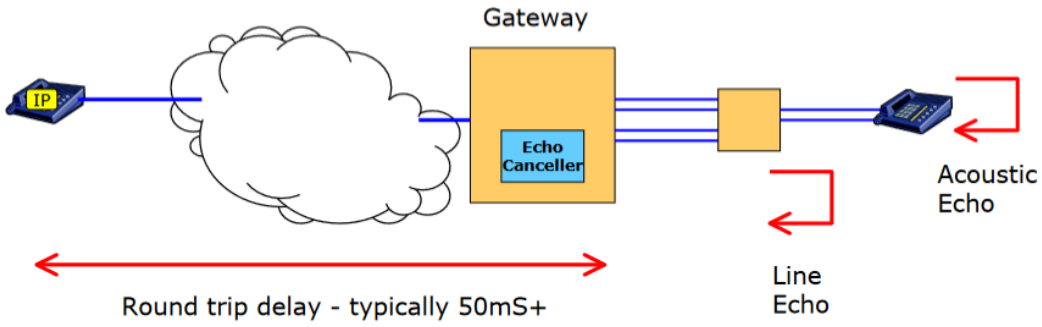


Figure 22: Causes of Echo

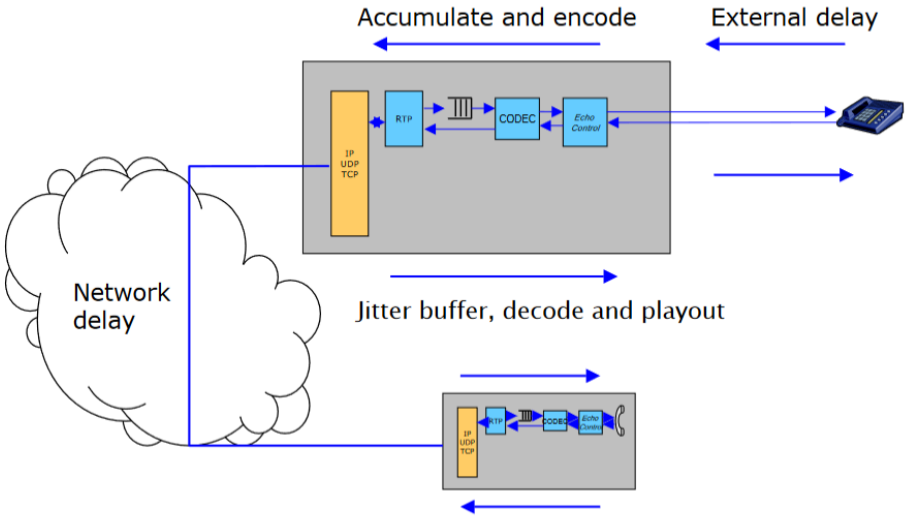


Figure 23: Causes of Delay

The problem of revising network layer delay variations to application layer loss and delay is addressed within the new ITU-T Recommendation G.1020, Packets that arrive with various impairments (delays, jitter, and errors) are processed by the applying of transforms interference into different impairments i.e. packet loss and extra delay by means of de-jitter. Packets with delay variation within the "white" range square measure accommodated, whereas packets with larger delay variation (in the "black" range) are discarded. during this method of transport layer delay variation can be mapped to application layer delay and packet loss. in order to compensate for jitter the best delay for the de-jitter buffer ought to be equal to the entire variable delay on the connection. sadly, it's not possible to find an optimal, fixed de-jitter buffer size once network conditions vary in time. Therefore, de-jitter buffers with dynamic size allocation, so called "adaptive playout buffers", are more appropriate. A nice de-jitter buffer ought to keep the buffering time as small as doable whereas minimizing the amount of voice packets that arrive too late to be played out. These 2 conflicting goals have led to numerous playout algorithms that calculate playout deadlines. A basic trade-off exists between buffering delay and packet loss. This trade-off is decided by the size of the de-jitter buffer. a bigger de-jitter buffer will accommodate packets with larger delay variation; thus fewer packets would be lost, at the expense of larger overall delay. Similarly, a smaller de-jitter buffer can produce less overall delay, however cause a bigger fraction of packets to be discarded by the terminal, therefore increasing the general loss.

Generally, a decent playout algorithm ought to be able to minimize both: buffering time and late packet loss and so improve the loss/delay trade-off.

4.2.1.4 Noise

Noise is any interfering sound in a call. Noise in the call can be due to low signal level or from bad equipment and from the proses of encoding (quantization noise). Also noise is considered environmental noise, just as a fan inside a room. As a service provider we need to distinguish between room noise that it is nothing we can do and Network/equipment/circuit noise. VoIP only solutions have very little noise compared to plain old telephone systems or hybrid solutions.

4.3 Measuring VoIP Performance

The evaluation of the quality of VoIP calls is a very difficult task. It is mostly performed by the statistical analysis of the satisfaction of real people. So because this is highly impractical we need a pseudo-subjective quality method to see the performance of VoIP telephony. To extrapolate the effects of various QoS parameters like end-to-end delay, packet loss in VoIP system as we discussed above. As we see in literature there aren't many methods to get a MOS rating and analysis of perceived voice quality in VoIP. The determination of Subjective measurement methods is a difficult thing to do. Subjective measurements of QoS are carried out by use of a group of people [65, 66]. As we can see a test phrase is recorded and then users listen to it under controlled conditions within special rooms, with background noise and other factors due to the surrounding environment, which are kept under control for test executions. Some examples of those are: hearing test, conversation assessment test, an interview and a survey test. There are disadvantages to this type of test mainly due to expense and are not implemented a lot in practice because of the large number of test subjects are needed. The obtained results are of statistical significance only when large numbers of people are participating.

The existing listening tests make possible the subjective assessment of speech. The goal of those tests is to evaluate the performance of the individual algorithms under different conditions. Some of the known hearing tests are as follows: ACR (absolute category rating) and DCR (degradation category rating) and CCR (comparison category rating).

The assessments of the quality related parameters are possible with the BYE message in the Cisco ecosystem mainly[67]. In Figure 23 for clarification we see JI = jitter, PL= Packets Loss, etc. as [67] explains.

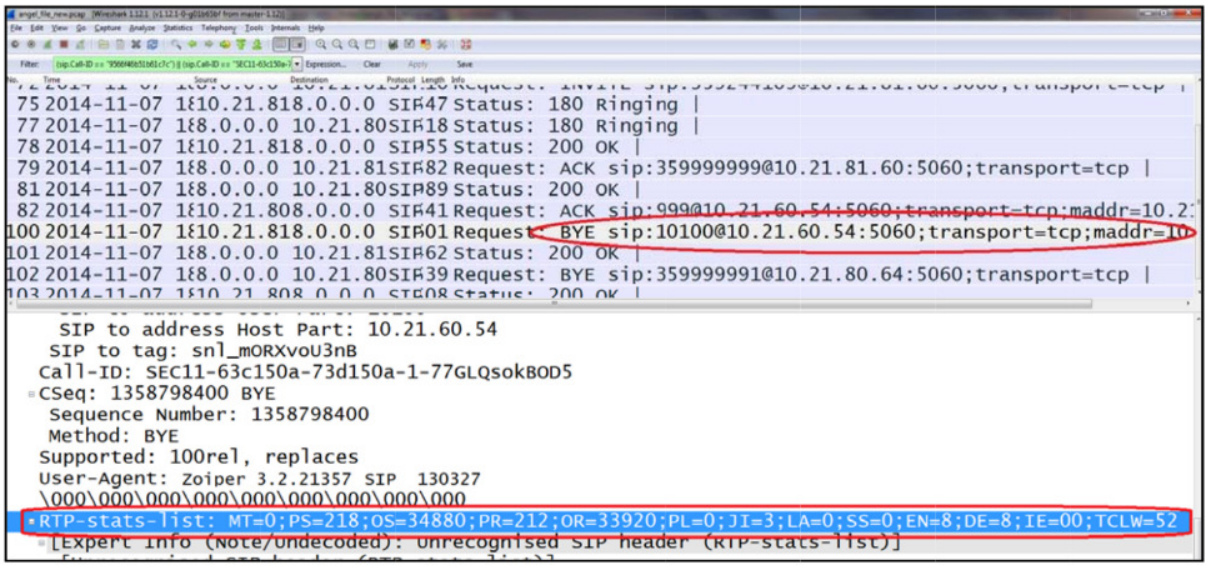


Figure 24: Cisco BYE message

4.4 WebRTC's Statistics API

WebRTC traffic is transported via best-effort IP based networks, which are by design susceptible to network congestion. A congestion in the network increases latency and packets may be dropped to mitigate the congestion, burst losses and long delays affect the quality of the media stream, and as such lowering the user experience at the receiving end. To make sure WebRTC calls can be offered at the best possible quality, the standard includes a real-time statistics API [6].

Someone can access the provided WebRTC statistics simply opening the WebRTC-internals page in the browser when making the call or using the getStats() API call. The getStats() API provides information as follows:

- Sender media capture statistics: media generation, typically frame rate, frame size, clock rate of the media source, the name of the codec, etc.
- Sender RTP statistics: media sender, typically packets sent, bytes sent, round-trip-time, etc.

- Receiver RTP statistics: media receiver, typically packets received, bytes received, packets discarded, packets lost, jitter, etc
- Receiver media: render statistics media rendering, typically frames lost, frames discarded, frames rendered, playout delay, etc.
- Datachannel metrics: messages and bytes sent and received on a particular datachannel.
- Interface metrics: metrics related to the active transport candidates. For example, if the network interface changes from a WiFi to 3G/LTE on a mobile device, or vice-versa. The active interface also carries the network related metrics: bytes, packets, sent or received on that interface, and the RTT.
- Certificate stats: shows the certificate related information, for example, the fingerprint and current algorithm.

The most influential and critical statistic is frame inter-arrival latency, or jitter: as frames are generated and sent periodically, it is reasonable to expect them to arrive periodically. Due to the presence of other traffic in the network, the packets may not only arrive out of order, but also arrive at varying intervals. In an audio call this may cause the syllables to elongate or be abruptly cut-off. If video is involved, this may result in loss of lip synchronization (audio and video are out of sync). Another very important statistic is packet losses and packet discards: packets may be lost in the network. Communication applications require fluidity; this means that the frames need to be decoded in time to preserve interactivity. Hence, frames that do not arrive in time to be decoded are often discarded even before decoding. In both cases, the decoder needs to compensate for the missing packet, either by applying concealment or just decoding as is. This may cause pixilation or a black screen for video and in audio speech may appear to skip. For example, the endpoint can parse the output of the getStats() query result for the inbound RTP statistics to get jitter, packets Loss, and packets Discarded.

There are also some security considerations such as stats identifiers may expose personally identifiable information, for example the IP addresses of the participating endpoints when a TURN relay is not used.

5 Design and Implementation

5.1 Design and Choices we made

5.1.1 The HTML and JavaScript hosting server

A Simple Apache server was used to host the HTML5 pages and JavaScript to the clients. Although we had to secure it and make an HTTPS certificate available for him.

5.1.2 Chosen JavaScript libraries

I had first tried the SIPml5 JS API with some success. But upon further testing we concluded that the ICE was not working properly and the error messages were far too cryptic for me to solve so we gave up on it. Afterwards we used JsSIP and from there on this is a main API that we used. JsSIP has a bit more community and the SIPml5 is obviously targeted to become part of a commercial program. JsSIP uses WebSocket as SIP transport [68]. By default then RTCNinja was the RTC wrapper API to deal with different browsers as transparently as possible, although it is not a promises ready wrapper as stated in W3C Working Draft 1.0 [69]. Adapter.js was used as a shim to insulate the rest of the JavaScript code from spec changes and prefix differences. For example, Chrome uses `webkitGetUserMedia()` and Firefox uses for the exact same call `mozGetUserMedia()`.

5.1.3 Environment

The following software, was used for the implementation:

- Execution Environment
    - Oracle VirtualBox v5.0.14
    - CentOS 7
    - Apache 2.4.18
    - Firefox Nightly with firefly add on
    - Google Chrome devbuild
    - PHP 7
- Mobile execution environment

    - Simulated Android phone
    - Iocean 8core 64bit device. 5.1 android version
        - Firefox beta for android
- Development Tools
    - Java Development Kit (JDK) 8
    - Netbeans 8.1
    - Notepad++
    - Putty
    - WinSCP
- JavaScript Libraries
    - Adapter.js
    - JsSIP.js
    - rtcNinja.js
    - jQuery.js

5.1.4 VM and host machine setup

The VM software that was used is an Oracle VirtualBox v5.0.14 and the allocation of resources was as high as my machine would go. The specs are:

- 10GB RAM 1800Mhz
- 4 cores Intel i5 @ 4.3Ghz over clocked, water-cooled
- Installed over and SSD 500MB read/500MB write
- Gigabit Ethernet LAN connection
- 1Mbps upload speed and 12Mbps Download WAN speeds

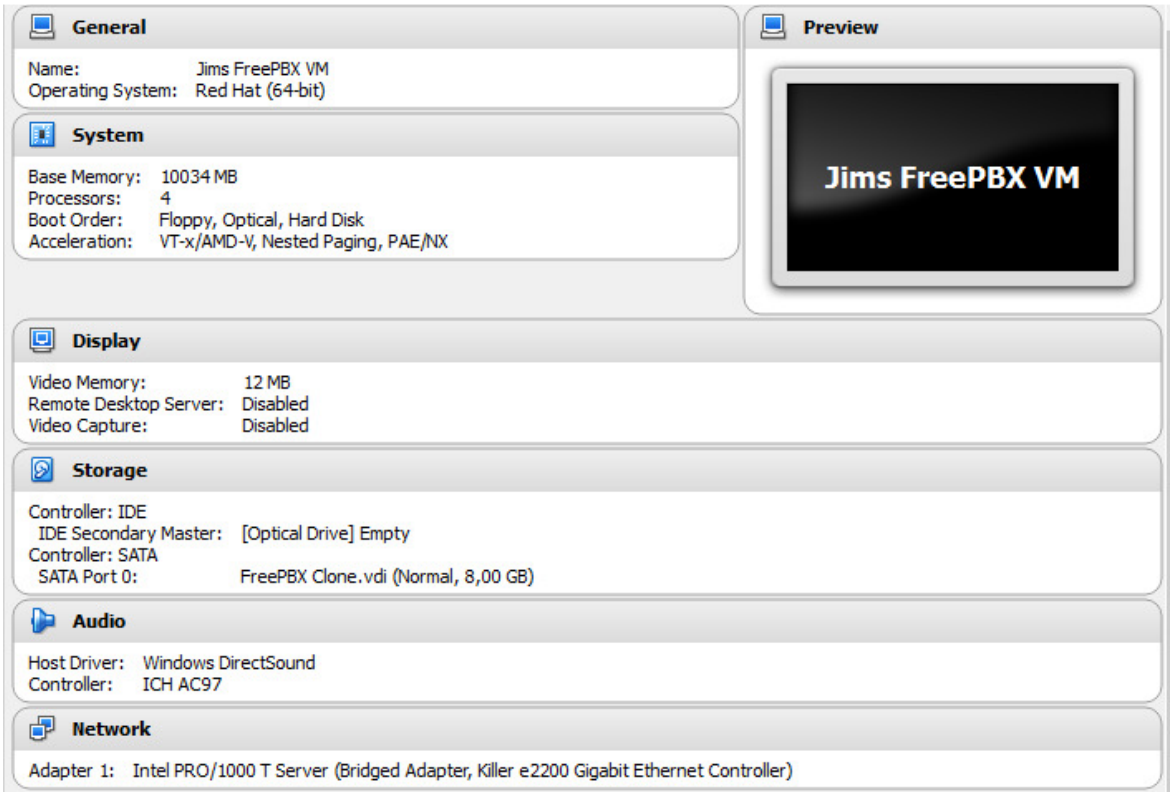


Figure 25: VM settings

5.1.5 Cryptography settings used

As RFC 5763 [21] and RFC draft [25] state, there MUST be no weak link or plain text transport for signaling and media. Thus the apache server has to offer https and the web socket server must offer Web Socket Secure transmission and Asterisk has to offer SRTP. In order to achieve this, we created my own CA so we can sign certificates. The method we used to create the CA is by invoking openSSL library. Then we generated the additional certificates for the HTTPS server the WSS server and another one for the asterisk server and also we set the settings for the SIP extensions to use cryptography and not to allow plain traffic to pass through (transport mode = WSS only and Enable Encryption option was set to SRTP only). This was also done to the web socket server. Then we went forward and installed my CA as a root authority in my machines and mobile phone so it would seem as a legitimate self-signed certificate. This is a very important step because supposedly my web apps would not work unless there where under secure conditions. In practice this was true for Chrome but not so for Firefox, but soon it will be. Also for our needs we have made an OpenVPN VPN tunnel. The home router was used in conjunction with OpenWRT in order to make the VPN work, simulating what an enterprise would do.

5.2 Implementation

5.2.1 The SIP registar and PBX server

Asterisk was chosen to be the PBX server that would create the backbone of this implementation and would bring all the multiprotocol technologies required for this thesis to come together. Asterisk is an open source framework that powers IP PBX systems, VoIP gateways, conference servers and many other hybrid solutions. Asterisk abstracts some of the complexities of the protocols used and marries together the POTS (plain old telephone system) with the VoIP world. We could think no better alternative than using Asterisk. The distribution we went with is FreePBX which has a strong community and a good bug reporting feature and does not cost anything compared to other installations [70].

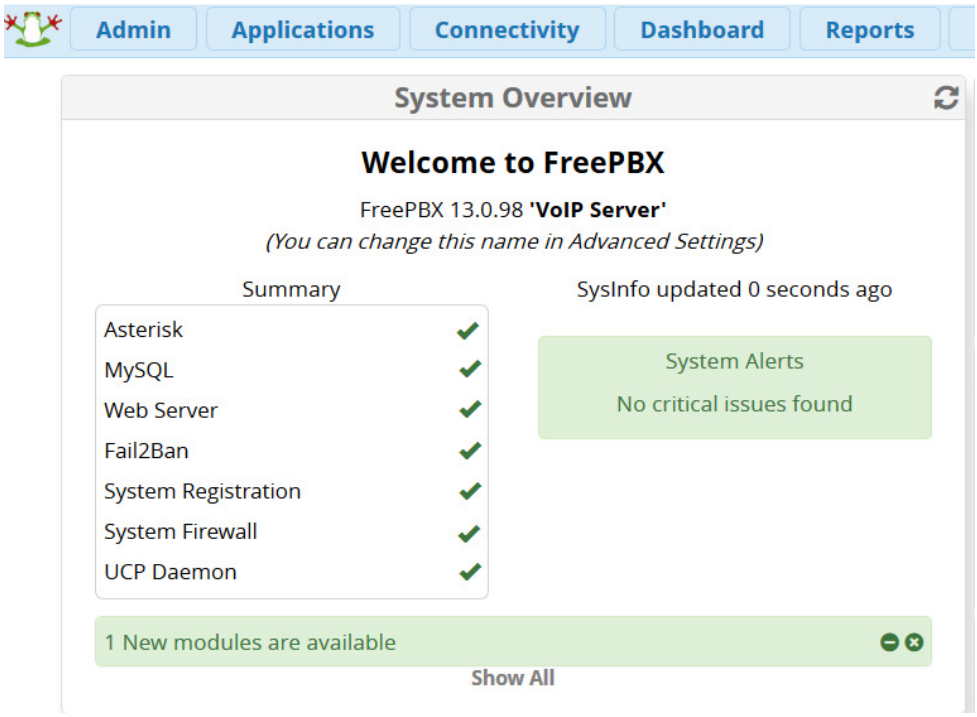


Figure 26: FreePBX main console

5.2.1.1 Connecting to the plain old telephone system

For the purpose of demo and exploration we purchased a SIP landline number 28210 8007. The emergency 122 100 166) numbers were assigned to Chania city from the company we bought the number from and we made this line an emergency line in asterisk settings.

For the outgoing settings, the dial pattern assigned to the Asterisk server for this outbound route was a rather simple NXXXXXXXXX and a ZXX where N matches any digit from 29, X matches any digit from 09 and Z matches any digit from 19. This ensured no calls were made abroad and in high toll numbers.

Calls incoming to this line where greeted from an Interactive Voice Response (IVR) asking to put the extension number or to dial a number for doing an echo test. Also the CID lookup Source was set for the web Scrapper called id we made and we have explained thoroughly in “Web Scraper Special PHP Caller ID identification” above.

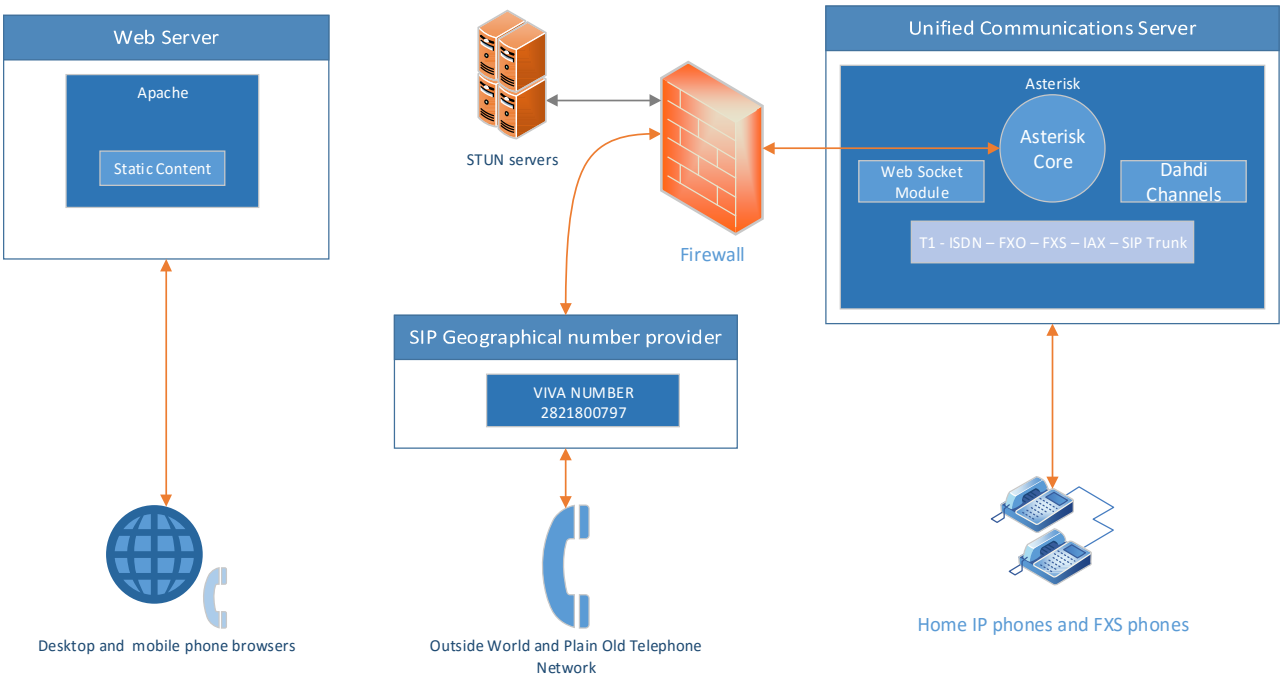


Figure 27: Thesis Servers and the connectivity to the world

5.2.1.2 Firewall and Intrusion detection

There is an expectation that when you have open ports to the outside world, people will try to hack you. We had more than 1000 SIP requests per hour after we let ports 5060 and 5061 open to the outside world. Script kiddies were attacking those ports and it was becoming hard for me to follow the logs of asterisk because of the clutter. we implemented firewall rules according to common sense and responsive firewall rules. Any incoming VoIP connection attempts that would be otherwise rejected are not blocked by the responsive model but instead allowed a very limited amount of registration attempts. If the registration attempt is successful, the remote host is then added to a 'Known Good' zone, that has permission to use that protocol, and is further more granted access to User Control Panel, if UCP is enabled. If the incoming connection attempts are shown to be invalid, then the traffic from that machine will be dropped for a short period of time. If attempts to authenticate from a particular IP continue without success, the attacking host will be blocked for 24 hours.

As intrusion detection goes Fail2ban is enabled and configured on the VM. Fail2ban screens attempts to compromise the system and logs them. If the attempts exceed the Max Retry limit, the remote IP is blocked from accessing the system for the length of Ban Time. It will also send email alerts when malicious connect attempts happen.

5.2.1.3 SIP extensions and SIP configuration

For the purposes if demo and experimentation we had made 4 sip extensions on the server. Namely 100 and 200 was the JsSIP extension and where configured as such. 300 and 400 where extensions configured to run of softphones. The important bit is extensions 100 and 200. They were configured as follows in the sip.conf file they are practically identical. The important bits are in bold letters.

Table 5: SIP peer settings example

```
[100]
deny=0.0.0.0/0.0.0.0
secret=notShownHere
dtmfmode=rfc2833
canreinvite=no
```

```
context=frominternal
host=dynamic
trustrpid=yes
sendrpid=no
type=friend
nat=yes
port=5060
qualify=yes
qualifyfreq=60
transport=wss
avpf=yes
force_avp=yes
icesupport=yes
encryption=yes
namedcallgroup=
namedpickupgroup=
dial=SIP/100
mailbox=100@default
permit=0.0.0.0/0.0.0.0
callerid=From JsSIP <100>
callcounter=yes
faxdetect=no
cc_monitor_policy=generic
dtlsenable=yes
dtlsverify=yes
dtlscertfile=/etc/asterisk/keys/default.pem
dtlscafile=/etc/asterisk/keys/ca.crt
dtlssetup=actpass
dtlsrekey=1
```

DTLS and SRTP had to be enabled and certificates issued for the WebRTC project to work correctly in the majority of the browsers (see Communications Security in page 32).

The SIPp SIP stack was used because we had problems making WSS work with JsSIP. Although it is in the process to be deprecated we found that it works a lot more stable as well.

5.2.1.4 Asterisk support of RFC 3327

So asterisk does not implement path mechanism RFC 3327. In the sip protocol he REGISTER function is used to associate a temporary contact address with an address of a record. This contact is typically in the form of a (URI) Uniform Resource Identifier, such as Contact: <sip:alice@gmail.com> and is typically dynamic and associated with the IP address or hostname of the SIP User Agent (UA). The problem is that network topology may have one or more SIP proxies between the UA and the registrar, such that any request traveling from the user's home network to the registered UA must traverse these proxies. The REGISTER method does not give us a mechanism to discover and record this sequence of proxies in the registrar for future use.
In short this allows discovery for intermediate proxies during SIP registration and in subsequent requests. Note that RFC 3327 recommends that the registrar support S/MIME, and attach a signed S/MIME of the response, which Asterisk does not currently support.

The click to call button has to have a sip password and alias but it has to be an anonymous and for that we can sent a <random>.invalid URI to asterisk, which asterisk won't agree very well to. Thus for the purposes of this thesis we have enabled the hack-uri-to-ip option that JsSIP offers.

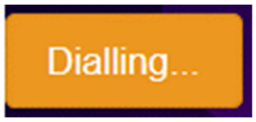
5.2.2 Click to Call Button

5.2.2.1 Button design and CSS manipulation

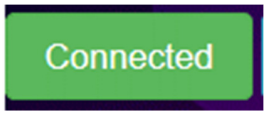
For the implementation of the click to call button the HTML design and CSS format was actually drown inspiration from similar use cases. CSS classes where used to define different colors for the click2call button. Below we see the different states that the button can be in.

- default State (WS connected or call terminated)

- dialing phase (call progress)



- answered (call accepted)



- Any type of error (call failed, WS disconnected, global error)

The CSS code that made that work is directly embedded in the js file in var js

On JsSIP events, the behavior was programmed as follows.

- The html button CSS class was changed so it would represent the current state of the call
- WS connection callback was attempted.
- Call generation callback was issued.

For example the accepted JsSIP case was programed as follows:

```
case 'progress':
jQuery(this.dom).removeClass().addClass('callbtn callbtndialing');
jQuery(this.dom).text(this.label_dialling);
```

```
        this.setOnClick('terminate', session);
        break;
```

And to reset the button after a call was terminated:

```
case 'ended':
 jQuery(this.dom).removeClass().addClass('callbtn callbtndefault');
 jQuery(this.dom).text(this.label_default);

 this.setOnClick('call', session);
 break;
```

5.2.2.2 Basic click to call button setup

The basic setup starts with the code to insert the click to call button in the web page. Since it is a button after all you enclose it in <button></button> tags and in between you enter the default label of this button, e.g. “Click to contact Sales”. Then the standard class definition for the CSS purposes, and which text to display during operations to let the user know what is happening (dialing, connecting…) and for localization purposes. Also the destination of the call is set to the “call to” parameter. After that the three must have, non-optional parameters that JsSIP asks, the web socket server location, the SIP URI assigned to the button and the password to connect to it. Because the password is plaintext (!) in asterisk we HAVE to make a custom context for that extension, in our example 100, that prohibits all activities except to call the extension(s) this button is supposed to reach. It is not a trivial think to do in an asterisk server and requires experience.

```
<button class="callme-btn callme-btn-lg" data-label-default="Call Echo Test"
data-label-dialling="Dialling..." data-label-connected="Connected"
data-call-to="*43" data-conf-ws_servers="wss://10.10.0.71:8089/ws"
data-conf-uri ="100@10.10.0.71" data-conf-password ="notShownHere" >
Call Echo Test
</button>
```

Figure 28: HTML part of the click 2 call button

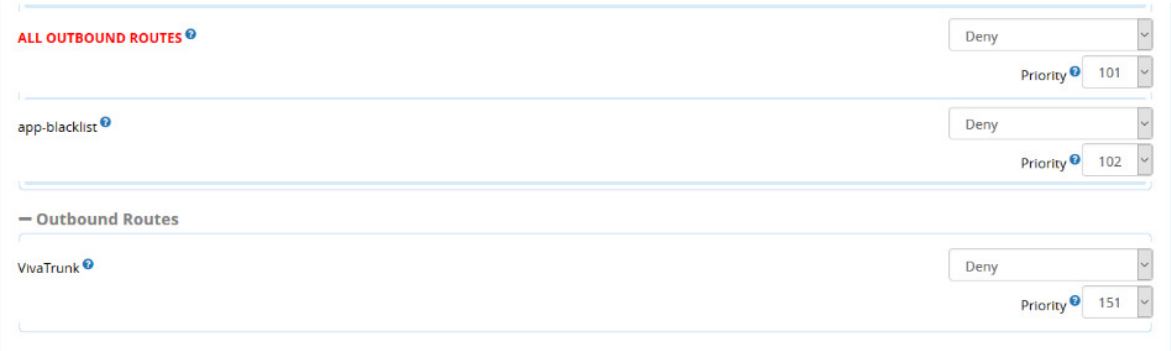


Figure 29: Part of the Custom Context created for the click to call button

5.2.2.3 Initialization of the button

As the page loads, first we enable the JsSIP debug option, in order to have console messages and a 'view' of the inner workings of this library. Then there is a check that the dowser supports A WebRTC and B WebSocket. If not the script end here. Then because we do not want to do a SIP register as soon as the web page loads but only later, if the user presses the button we disable the register parameter of JsSIP. Then we inject the config we have prepared to a UA (User Agent) which is associated to a SIP user account.

To end the initialization, the click methods for the button are bounded and the JsSIP UA event callback definitions are set. The only callbacks required for the click to call button are onConnect() and onDisconet().

5.2.3 Caller Id Scraped from the web –Reverse find of caller ID

As we were exploring the features of asterisk we noticed that there is not a similar application to the, very useful for me, GreekCallerID app we had in my android phone. It seemed that we could not do without it. This application would go to whitepages.gr xrisosodigos.gr and other places, so it would find out who is calling you in real time.

I made essentially a plugin PHP page for asterisk, a mere webpage that after asterisk would give it a phone number this page would go and scrape xo.gr and ote.gr to find the caller's name, if it existed. If it would, the PHP script would 'translate it' to greeklish so even non Unicode IP phones would have no problem to understand it. If it would not find a result it would just return a

blank page. We used the curl library in order to scrape the data of the internet. Below we see the part of the code that will search the phone number and scrape it of the ote.gr webpage.

Table 6: Part of the scrapper code that lifts the Name that it found

```
 if ($scraped_data=="")[
       $scraped_page                                                    =
curl("http://11888.ote.gr/web/guest/listnames?_wpType=number&_wpPhone=".
$_GET["phone"]);
 $scraped_data  =  scrape_between($scraped_page,  "<span  class=\"title\">",
"</span>");
 ]
```

Table 7: Part of the code that does the greeklish translation

```
function greeklish($Name)
[ $greek = array('α', 'ά', 'Ά', 'A', 'β', 'B', 'γ', 'Γ', 'δ', 'Δ', 'ε', 'έ',
'E', 'Έ', 'ζ', 'Z', 'η', 'ή', 'H', 'θ', 'Θ', 'ι', 'ί', 'ϊ', 'ΐ', 'I', 'Ι', 'κ',
'K', 'λ', 'Λ', 'μ', 'M', 'ν', 'N', …).
$english array('a', 'a', 'A', 'A', 'b', 'B', 'g', 'G', 'd', 'D', 'e', 'e', 'E',
'E', 'z', 'Z', 'i', 'i', 'I', 'th', 'Th', 'i', 'i', 'i', 'i', 'I', 'I', 'k',
'K', 'l', 'L', 'm', 'M', 'n', 'N', ….)
$string = str_replace($greek, $english, $Name);
return $string;
```

And the corresponding ‘bridge’ between my PHP code and asterisk. The local Apache server is responsible of executing the PHP code.

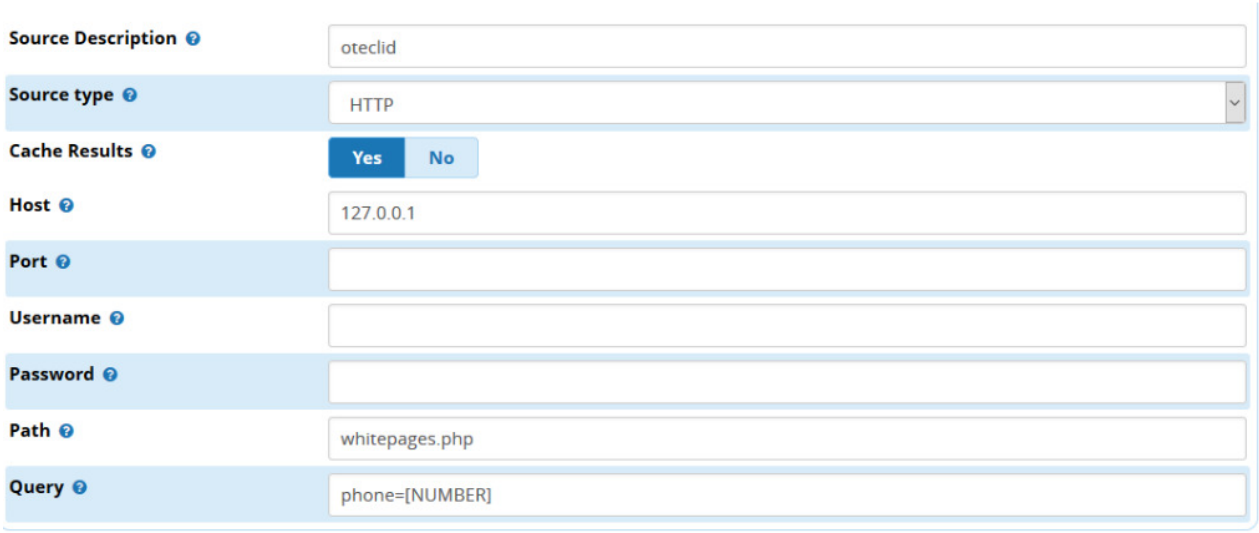


Figure 30: The asterisk CID Lookup settings page

5.2.4 WebRTC supported Web Phone

For demonstration purposes, this user interface simulates a real phone that someone would expect to have in front of him together with its functions. Additionally, we can see that there is a get statistics button that will show all the available statistics the browser has to offer. Limitations exist, such as not having the possibility to have re-invites, needed for hold function to work. GUI interface is courtesy from the JsSIP project. Additional code and functions where added in order to make it work with our implementation and also to add support for statistics gathering.

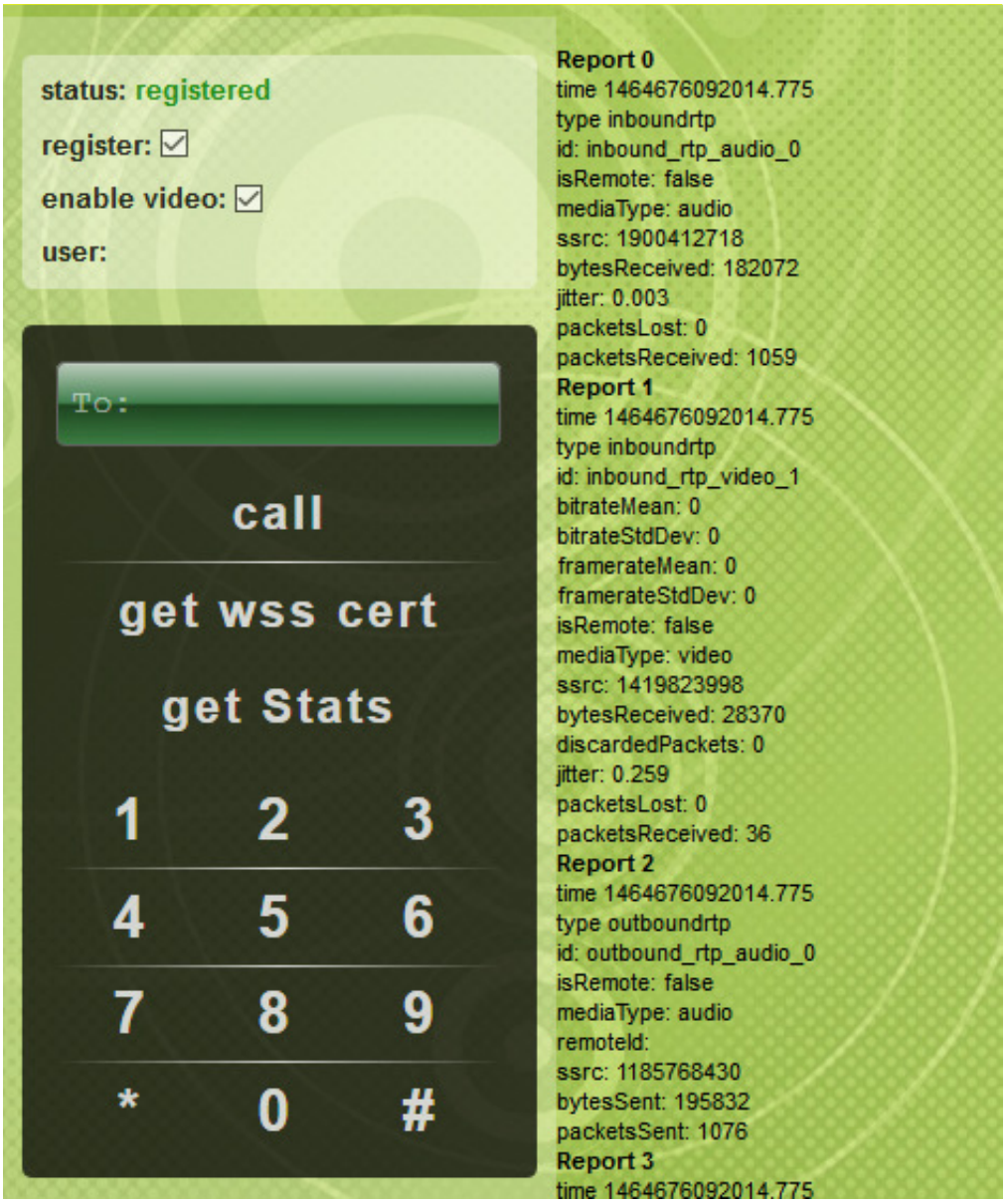


Figure 31: WebRTC demo application showing also statistics

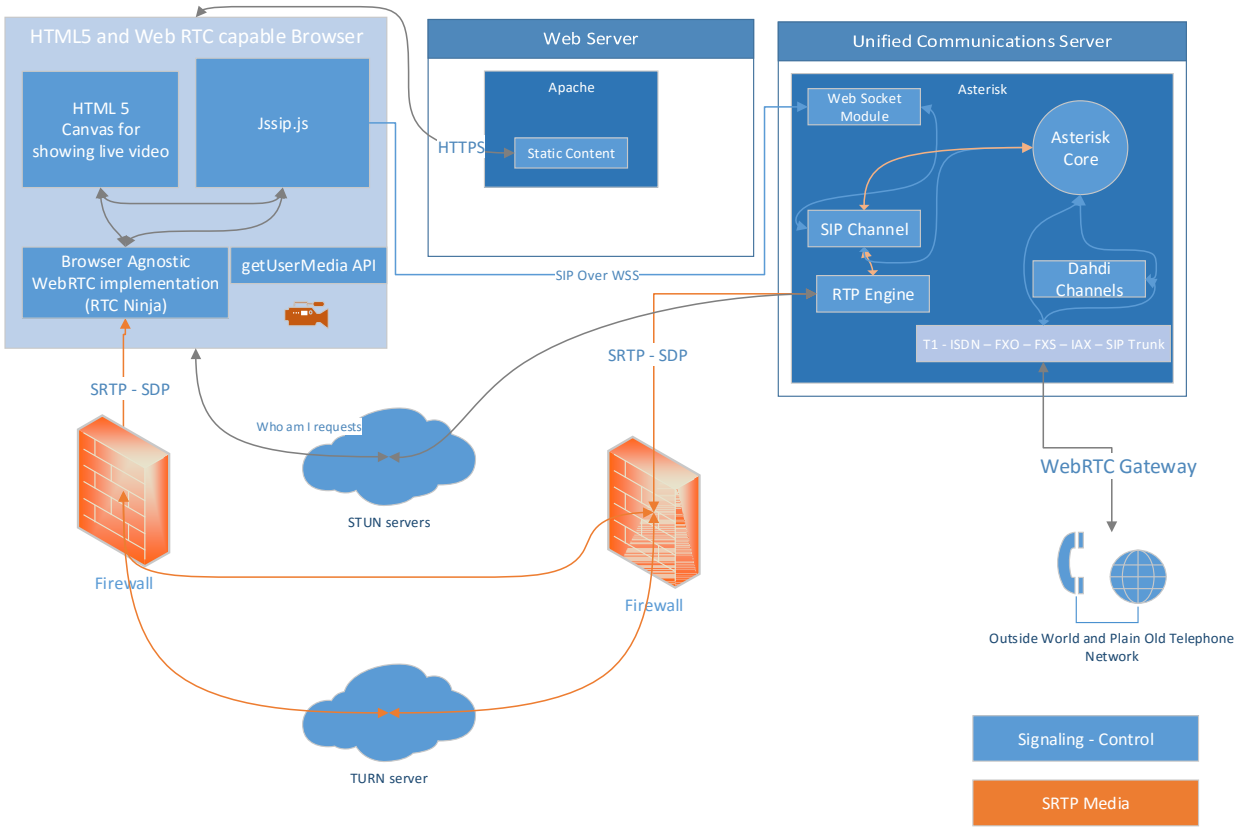


Figure 32: How the constituent parts of the implementation work together

6 Evaluation of WebRTC – to - SIP calls

As found in bibliography [47, 66, 71, 72] the major players in the quality aspect of a VOIP call are jitter and Delay, which is round trip time divided by two. Accepted values are delay up to 100ms and jitter between 0 and 20ms. In all other scenarios the quality degradation is audible by a human.

Table 8: Voice Quality Measures

Network parameter	Good	Acceptable	Poor
Delay (ms)	0-100	101-300	>300
Jitter (ms)	0-20	21-50	>50

6.1 Testing we have done

In order to evaluate the good working order of the system we have tried many possible scenarios that could come up in a real world situation. The following calls were made to and from the PBX box. If the call was an incoming to the PBX it would be diverted to an echo application most of the time, so we could also have an audible estimation of the delay between the machine and the caller.

Landline POTS based tests

- to internal sip extension, both soft phone and WebRTC
- to echo application
- to extension thought VPN, both soft phone and WebRTC
- to extension on mobile phones browser (Firefox)

Internal based tests

- Lan extension to External WAN extension.
- to another LAN extension, both soft phone and WebRTC
- External LAN extension to Internal Lan extension
- Extensions to landline

- All the above with conference.

VPN based tests

- to internal, both soft phone and WebRTC
- to external extension, both soft phone and WebRTC

And last WAN to WAN extensions, both soft phone and WebRTC phones.

There is no need to test combined scenarios, e.g. external POTS device to External WebRTC, (even that we have made tests) because this kind of test just combines 2 successful other kind of tests.

We have concluded that the system works wonderfully and there is no significant wait for the calls as there are being established.

6.2 Statistics Gathering code

6.2.1 Real Time Gathering

In order to collect all the statistics the WebRTC has gathered through RTCP[6] packets we have to use the API that is provided from the browser. To begin we made a simple `<div>` in the HTML we needed to show all the gathered stats. That code is as follows:

```
<section id="statistics">
<div id="senderStats"></div>
<div id="receiverStats"></div>
</section>
```

This section will be the placeholder for the JavaScript to anchor the viewable statistics for demonstration purposes and it kind of simulates the 'Statisticsfor nerds" section YouTube has.

Afterwards to the .js file the variables are initialized

```
var senderStatsDiv = document.querySelector('div#senderStats');
var receiverStatsDiv = document.querySelector('div#receiverStats');
var bitrateDiv = document.querySelector('div#bitrate');
var peerDiv = document.querySelector('div#peer');
var bytesPrev;
var timestampPrev;
```

The function to be called to display the statisticsis called `invokeGetStatistics(remotePeerConnection)` and the parameter `remotePeerConnection` is the RTPConnection object that we need to get the statistics from. Then we check if the object is not null and that it has streams attached to it by doing the following `if (remotePeerConnection && remotePeerConnection.getRemoteStreams() [0])`. If this test passed we extract the raw results by:

```
remotePeerConnection.getStats(null, function(results) [
 var statsString = dumpStats(results);
```

The dumpStats function prettifies and makes the results human readable by the following Code:

```
function dumpStats(results) [
 var statsString = '';
 Object.keys(results).forEach(function(key, index) [
 var res = results[key];
 statsString += '<h3>Report ';
```

```
 statsString += index;
 statsString += '</h3>\n';
 statsString += 'time ' + res.timestamp + '<br>\n';
 statsString += 'type ' + res.type + '<br>\n';
 Object.keys(res).forEach(function(k) [
 if (k !== 'timestamp' && k !== 'type') [
 statsString += k + ': ' + res[k] + '<br>\n';
 ]
 ]);
 ]);
 return statsString;
]
```

Keep in mind that for demo purposes the types localcandidate remotecandidate googCandidatePair candidatepair **are not shown** to the HTML page due to them being too lengthy and not critical for our purpose.

Then we resume on calculating some thinks that the browser won't give us for free. That is the:

IP of the user we are trying to connect to by figuring out the candidate pair:

```
 Object.keys(results).forEach(function(result) [
 var report = results[result];
 if (report.type === 'candidatepair' && report.selected ||
 report.type === 'googCandidatePair' &&
 report.googActiveConnection === 'true') [
 activeCandidatePair = report;
 ]
 ]);
 if (activeCandidatePair && activeCandidatePair.remoteCandidateId) [
 Object.keys(results).forEach(function(result) [
 var report = results[result];
 if (report.type === 'remotecandidate' &&
 report.id === activeCandidatePair.remoteCandidateId) [
 remoteCandidate = report;
 ]
 ]);
 ]
 if (remoteCandidate && remoteCandidate.ipAddress &&
```

```
remoteCandidate.portNumber) [
peerDiv.innerHTML = '<strong>Connected to:</strong> ' +
remoteCandidate.ipAddress +
':' + remoteCandidate.portNumber;
]
```

And the bitrate of the video stream if any

```
if (report.type === 'inboundrtp' && report.mediaType === 'video') [
// firefox calculates the bitrate for us
// https://bugzilla.mozilla.org/show_bug.cgi?id=951496
bitrate = Math.floor(report.bitrateMean / 1024);
] else if (report.type === 'ssrc' && report.bytesReceived &&
report.googFrameHeightReceived) [
// chrome does not so we need to do it ourselves
var bytes = report.bytesReceived;
if (timestampPrev) [
bitrate = 8 * (bytes - bytesPrev) / (now - timestampPrev);
bitrate = Math.floor(bitrate);
]
bytesPrev = bytes;
timestampPrev = now;
]
if (bitrate) [
bitrate += ' kbits/sec';
bitrateDiv.innerHTML = '<strong>Bitrate:</strong> ' + bitrate;
]
```

Below we can see how this looks like in the browser

```
Receiver stats
Report 0
time 1460899983181.84
type inboundrtp
id: inbound_rtp_audio_0
isRemote: false
mediaType: audio
ssrc: 669552950
bytesReceived: 18748
jitter: 0
```

```
packetsLost: 0
packetsReceived: 109
Report 1
time 1460899983181.84
type inboundrtp
id: inbound_rtp_video_1
bitrateMean: 228103.6363636363
bitrateStdDev: 46229.211831626824
framerateMean: 6.787878787878789
framerateStdDev: 2.628337594023741
isRemote: false
mediaType: video
ssrc: 1263900901
bytesReceived: 1945623
discardedPackets: 0
jitter: 0.361
packetsLost: 0
packetsReceived: 2569
Report 2
time 1460899983181.84
type outboundrtp
id: outbound_rtp_video_1
```

6.2.2 Post Call Statistics Gathering

Also we have found a way to store for after call analysis many important data in the CDR directory of asterisk, per call basis. The info we were able to take was the following:

- Packets Sent
- Packets Received
- Local RX Packet Loss
- Local TX Packet Loss
- Local RX Jitter
- Local TX Jitter
- Local Jitter Max

- Local Jitter Min
- Local Jitter Norm Dev
- Local Jitter Std Dev

App	Destination	Disposition	Duration	Userfield
BackGround	*43	ANSWERED	00:05	Packets Sent: 223 Packets Received: 0 Local RX Packet Loss: 0 Local TX Packet Loss: 0 Local RX Jitter: 0.000000 Local TX Jitter: 0.000000 Local Jitter Max: 0.000000 Local Jitter Min: 0.000000 Local Jitter Norm Dev: 0.000000 Local Jitter Std Dev: 0.000000

Figure 33: Screenshot of the CDR report we made for accessing post call stats

6.3 MOS measuring code

6.3.1 Real time measurement

The hard work done by the statistics gathering code and the collected data has to be converted to useful information for the end users, e.g. for assessing the subjective quality of calls during the call time. To this end, we have made use of a simplified E-model measuring system. As previously mentioned at “Evaluation of WebRTC – to - SIP calls” in chapter 6 drawing from literature, we have extrapolated a simplified way of determining a live MOS. In Figure 34 we can see a color coded code snippet that calculates the simplified MOS in real time using simply the measured RTT of a call. Figure 35 illustrates a call with the associated MOS calculation.

```
function eModel(){

      if (now.type == "outboundrtp") {

        rttMeasures.append(now.roundTripTime);
            var avgRTT = average(rttMeasures);

        var emodel = 0;
        if (avgRtt/2 >= 500)
           emodel = 1;
        else if (avgRtt/2 >= 400)
           emodel = 2;
        else if (avgRtt/2 >= 300)
           emodel = 3;
        else if (avgRtt/2 >= 200)
           emodel = 4;
        else if (avgRtt/2 < 200)
           emodel = 5;
        console.log ("e-model: "+str(emodel));
      }

}

function average (values) {
  var sumValues = values.reduce(function(sum, value){
    return sum + value;
  }, 0);
    return (sumValues / values.length);
}
});
```

Figure 34: MOS measuring code

Figure 35: Screenshot of the live MOS score

6.3.2 Post call measurements

Saved in the CDR we can calculate the MOS by accessing the data stored in there as we discussed in 6.3.2 “Post call measurements”. Using these data, we can apply many derivatives of the E-model and calculate the end MOS score for the whole call. This approach gives us currently more options than the real time calculation we saw above. We set as quality rules: Packet loss should be below 0.5% and jitter < 5 ms, and RTT (measurement for latency) < 200 ms or less. The code that does this calculation follows. Figure 36 depicts the resulted calculation for a call.

```
exten    =>    s,n,Set(LOST_LOCAL_TOT=${MATH(${CUT(RTPAUDIOQOSLOSS,\;,1):5]    /
${CUT(RTPAUDIOQOSLOSS,\;,2):9],float)])
exten  =>  s,n,Set(LOST_REMOTE_TOT=${MATH(${CUT(RTPAUDIOQOSLOSSBRIDGED,\;,1):5]
/  ${CUT(RTPAUDIOQOSLOSSBRIDGED,\;,2):9],float)])
```

and the stats reported by other end:

```
exten => s,n,Set(JITTER_REP_LOCAL_AVG=${MATH(${CUT(RTPAUDIOQOSJITTER,\;,7):19]
/ 1000)])
exten                                                                          =>
s,n,Set(JITTER_REP_REMOTE_AVG=${MATH(${CUT(RTPAUDIOQOSJITTERBRIDGED,\;,7):19]
/ 1000)])
exten                                                                          =>
s,n,Set(CDR(userfield)=${CDR(userfield)]&lost_remote:${LOST_REMOTE_TOT]&lost_
local:${LOST_LOCAL_TOT]&format_native=${FORMAT_NATIVE])
exten                                                                          =>
s,n,Set(CDR(userfield)=${CDR(userfield)]&jitter_remote:${JITTER_RX_REMOTE_AVG
]&jitter_local:${JITTER_RX_LOCAL_AVG])
exten                                                                          =>
s,n,Set(CDR(userfield)=${CDR(userfield)]&jitter_rep_remote:${JITTER_REP_REMOT
E_AVG]&jitter_rep_local:${JITTER_REP_LOCAL_AVG])
exten                                                                          =>
s,n,Set(CDR(userfield)=${CDR(userfield)]&rtt_remote:${RTT_REMOTE_AVG]&rtt_loc
al:${RTT_LOCAL_AVG])
exten                                                                          =>
s,n,Set(LOST_REMOTE=${MATH(${CHANNEL(rtpqos,audio,remote_lostpackets)]       /
${CHANNEL(rtpqos,audio,remote_count)],float)])
exten  =>  s,n,Set(LOST_LOCAL=${MATH(${CHANNEL(rtpqos,audio,local_lostpackets)]
/  ${CHANNEL(rtpqos,audio,local_count)],float)])
```

```
exten => s,n,Set(JITTER_REMOTE=${CHANNEL(rtpqos,audio,remote_jitter)])
exten => s,n,Set(JITTER_LOCAL=${CHANNEL(rtpqos,audio,local_jitter)])

exten => s,n,Set(CDR(userfield)=${CDR(userfield)]&lost_remote
 ${LOST_REMOTE]&lost_local:${LOST_LOCAL]&format_native=${FORMAT_NATIVE])

exten => s,n,Set(CDR(userfield)=${CDR(userfield)]&jitter_remote

${JITTER_REMOTE]&jitter_local:${JITTER_LOCAL]&rtt:${CHANNEL(rtpqos,audio,rtt)
])
```

Disposition	Duration	Userfield
ANSWERED	00:17	Packets Sent: 831 Packets Received: 816 Local RX Packet Loss: 0 Local TX Packet Loss: 0 Local RX Jitter: 0.000000 Local TX Jitter: 0.003094 Local Jitter Max: 0.008030 Local Jitter Min: 0.000226 Local Jitter Norm Dev: 0.002249 Local Jitter Std Dev: inf E-Model Calc: 5

Figure 36: Post call statistics in the asterisk CDR

6.4 Security related evaluation

Due to the tests we were making, we had left ports 5060 – 5061 exposed to the internet. This has the result that script kiddies where trying to brute force various extensions. They did not once try to attack an actual existing extension (e.g. 100@myip) but seemingly random sip extension, like 9999 or 1234 with passwords that resembled classic brute attacks (aaaa then going aaab etc). Fortunately, the fail to ban application installed in the distribution took care of them, as we discussed above and banned them for a period of time after 3 failed login attempts. Nonetheless my inbox got flooded with banned IP reports. For the duration of the attack (about 1 day) we received more than 200 mails. The system was not compromised and thus we experienced why there is a need to hide a PBX behind a SIP proxy.

7 Conclusion and Possible use cases

7.1 Possible use cases

- Click to call for support from customer facing website
    - A user is using his/her mobile device and browses a company's Website when they come across a button to request customer services. As for example it could be an online shopping Website which offers an online shop assistant to help with products or the ordering-payment process.
    - To elaborate the customer clicks the customer services link which establishes a video or audio call with the customer services person at the facility. In this case, the mobile device uses WebRTC -based video call and the call center has a WebRTC client on a desktop computer. During this video call, the customer service person can send technical details and specifications for a product to the caller, or even demonstrate how the product is used.
    - Alternatively, consider the case on which someone is involved in vehicular accident and they call report the incident to their insurance company. They call the insurance company through their VoLTE client, but the Insurance Company has WebRTC clients deployed on tablets for each call center agent. In the process of this call, the insurance company wants to view the damage done, and the user switch to a video call and thus resolve problems faster.
- Click to call from enterprise directory web page
- Social Networking integration
    - In this scenario, some friends are using a social site and communicate through a feature enabling RTC based on WebRTC. The various friends of this call are using a number of different devices capable of supporting a Web browser, such as a Smart TV, tablet, smartphone and laptop.
- Click to call me" URL in email signature
- Desktop sharing, collaboration
    - TeamViewer like
- Existing Collaboration Apps
    - Backboard

- Multiparty video calls
    - In this scenario the users can use plugin-less conference call facilities leveraging WebRTC technology. Customers can access, via a WebRTC client, a conference service without downloading a plugin. Also consider some customers may prefer to access the conference from their mobile devices which support communication services like WebRTC.
- Bit Torrent like apps directly from the browser
- interactive TV
- Web Notification All the necessary application notification will be display through the browser web notification message

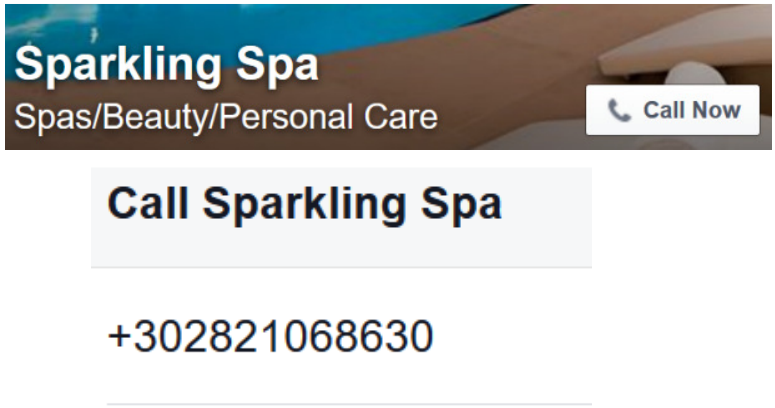


Figure 37: Example from social networking site that would benefit from a Click to Call button

7.2 Conclusions

WebRTC is becoming a pervasive and disruptive technology, for sure. Any device with a Chrome, Firefox, Opera are already WebRTC available and in working order. Many mobile applications developers are also using the WebRTC to voice and video enable their applications and enhance interoperability with existing services. The WebRTC service ecosystem can be fully integrated it with existing video, voice, and texting services.

It is important to understand in the meantime that while the WebRTC standard is still being tweaked and that not everyone agrees on the codecs WebRTC should use, the audio part of the WebRTC standard is solid and stable. For that reason alone, organizations can start creating

WebRTC-enabled applications with confidence. In environments where the enterprises have control over what type of browser the user has, video applications can also work very well as well. All around this is a very promising technology, waiting to change a lot about the way we do communications as we stated in the introduction.

Concerning the thesis, we conclude that the goals have been met and the journey of this thesis

References

1. Hickson, I., *The HTML5 Draft Standard. Web Hypertext Application Technology Working Group*. 2010.
2. Bergkvist, A., et al., *WebRTC 1.0: Real-time Communication Between Browsers. W3C Editor's Draft, W3C*. 2012. https://www.w3.org/TR/webrtc/
3. Handley, M., *RFC 4566 Session Description Protocol (SDP)*. http://www.ietf.org/rfc/rfc4566.txt
4. Engan, M., et al., *RFC 2509: IP header compression over PPP*. 1999, RFC 2509, February, Tech. Rep.
5. Schulzrinne, H., *RFC 2326 Real time streaming protocol (RTSP)*. 1998. http://tools.ietf.org/html/rfc2326.txt
6. Harald Alvestrand, G., *W3C Identifiers for WebRTC's Statistics API.* October 2015.
7. Rosenberg, J., et al., *RFC 3261: SIP: session initiation protocol*. 2003, IETF, Tech. Rep., 2002.[Online]. Available: www. ietf. org/rfc/rfc3261. txt.
8. Berners-Lee, T., R. Fielding, and L. Masinter, *RFC 2396: Uniform resource identifiers (URI): Generic syntax.* Status: DRAFT STANDARD, 1998.
9. Conroy, S.P.a.L., *RFC 2848: The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services.* June 2000.
10. Handley, M. and V. Jacobson, *RFC 2327.* SDP: session description protocol, 1998. **10**.
11. Burnett, D. and A. Narayanan, *Media capture and streams.* World Wide Web Consortium WD WD-mediacapturestreams-20120628, 2012.
12. Rosenberg, J., *RFC 5245 Interactive Connectivity Establishment (ICE)*. https://tools.ietf.org/html/rfc5245
13. Holmberg, C., H. Alvestrand, and C. Jennings, *Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers.* draft-ietf-mmusic-sdp-bundle-negotiation-00 (work in progress), 2012.
14. J. Uberti, C.J., Google, Cisco, *Javascript Session Establishment Protocol (JSEP) Internet-Draft.* 2013.
15. Saint-Andre, P., *Jingle: Jabber does multimedia.* IEEE MultiMedia, 2007. **14**(1): p. 90-94.
16. Jennings, C., *RTCWeb Offer/Answer Protocol (ROAP) draft-jennings-rtcweb-signaling-01.* Network Group, Internet Draft expired on May, 2012. **2**.
17. Hickson, I., *The WebSocket API- W3C Working Draft.* May 2012.
18. Loreto, S., *RFC 6202 nown Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP*. April 2011. https://tools.ietf.org/html/rfc6202
19. Fette, I., *RFC 6455 The WebSocket Protocol.*
20. Rescorla, E., *RFC 2818 HTTP Over TLS. Updated by RFC 5785 Defining Well-Known Uniform Resource Identifiers (URIs).*
21. Rescorla, E., *Security Considerations for WebRTC*. https://tools.ietf.org/html/draft-ietf-rtcweb-security-08
22. Barth, A., *The Web Origin Concept*. https://www.ietf.org/rfc/rfc6454.txt
23. Baugher, M., *RFC 3711 The Secure Real-time Transport Protocol (SRTP)*. https://tools.ietf.org/html/rfc3711
24. Rescorla, E., *RFC 4347 Datagram Transport Layer Security (DTLS)*. https://tools.ietf.org/html/rfc4347

25. Fischl, J., *RFC 5763 Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)*. https://tools.ietf.org/html/rfc5763
26. Farrell, S. and H. Tschofenig, *Pervasive monitoring is an attack.* 2014.
27. IESG, I., *IETF Policy on Wiretapping*. 2000, RFC 2804, May.
28. Wing, D., et al., *RFC 5479 Requirements and analysis of media security management protocols.* 2009.
29. Rosenberg, J., *RFC 6544 TCP Candidates with Interactive Connectivity Establishment (ICE)*. https://tools.ietf.org/html/rfc6544
30. Rosenberg, J., *RFC 5389 Session Traversal Utilities for NAT (STUN)*. https://tools.ietf.org/html/rfc5389
31. Mahy, R., *RFC 5766 Traversal Using Relays around NAT (TURN)*. https://tools.ietf.org/html/rfc5766
32. Dorgham, S., *Next-Gen Open Service Solutions over IP (N-GOSSIP)—Area for SIP Enhancements.* Internet Citation, XP002235858.
33. Singh, K. and V. Krishnaswamy, *A case for SIP in Javascript.* IEEE communications magazine, 2013. **51**(4): p. 28-33.
34. Bertin, E., et al. *WebRTC, the day after*. in *Procs. 17th International Conference on Intelligence in Next Generation Networks, ICIN*. 2013.
35. *sipMPL5 - HTML5 SIP client*. Available from: https://www.doubango.org/sipml5/index.html.
36. *jsSIP - The Javascript SIP library*. Available from: http://jssip.net/.
37. caniuse.com. *Can I use Web Sockets?* 2016; Available from: http://caniuse.com/#feat=websockets
38. Labs, A., *Avaya IP Voice Quality Network Requirement*. 2009. http://downloads.avaya.com/css/P8/documents/100018203
39. Mahdi, A.E. and D. Picovici, *Advances in voice quality measurement in modern telecommunications.* Digital Signal Processing, 2009. **19**(1): p. 79-103.
40. Narbutt, M. and M. Davis, *Assessing the quality of VoIP transmission affected by playout buffer scheme.* 2005.
41. Ding, L., et al., *Non-intrusive single-ended speech quality assessment in VoIP.* Speech communication, 2007. **49**(6): p. 477-489.
42. Goudarzi, M., *Evaluation of voice quality in 3G mobile networks*. 2008, University of Plymouth.
43. Al-Akhras, M., et al., *Non-intrusive speech quality prediction in VoIP networks using a neural network approach.* Neurocomputing, 2009. **72**(10): p. 2595-2608.
44. Daengsi, T., et al. *A study of VoIP quality evaluation: User perception of voice quality from G. 729, G. 711 and G. 722*. in *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*. 2012. IEEE.
45. Union, I.T., *ITU-T Recommendation P. 800: Methods for subjective determination of transmission quality*. 1996, August.
46. Rec, I., *ITU-T Recommendation P.805 Subjective evaluation of conversational quality.* Intl. Telecom. Union, 2007.
47. Dymarski, P., S. Kula, and T.N. Huy, *QoS conditions for VoIP and VoD.* Journal of Telecommunications and Information Technology, 2011: p. 29-37.

48. Johannesson, N.O., *The ETSI computation model: a tool for transmission planning of telephone networks.* Communications Magazine, IEEE, 1997. **35**(1): p. 70-79.
49. Bergstra, J. and C. Middelburg, *G.107 The E-model, a Computational Model for Use in Transmission Planning.* ITU-T Recommendation G, 2006. **107**.
50. Association, T.I., *Voice quality recommendations for IP telephony.* TIA/EIA Telecommunications Systems Bulletin, TSB116, 2001.
51. Time, O.-W.T., *ITU-T Recommendation G. 114.* ITU-T May, 2000.
52. Rec, I., *G. 131,".* Talker echo and its control, 2003.
53. Daengsi, T., et al., *Comparison of perceptual voice quality of VoIP provided by G. 711 and G. 729 using conversation-opinion tests.* IJCIM, 2012. **20**(1): p. 21-26.
54. Xplore, I. *E-model Seach result count*. 2106; Available from: http://ieeexplore.ieee.org/search/searchresult.jsp?queryText%3DE-model&addRange=2010_2016_Publication_Year&pageNumber=1&resultAction=REFINE.
55. Karapantazis, S. and F.-N. Pavlidou, *VoIP: A comprehensive survey on a promising technology.* Computer Networks, 2009. **53**(12): p. 2050-2090.
56. Rec, I., *P. 800.1, Mean opinion score (MOS) terminology.* International Telecommunication Union, Geneva, 2006.
57. Voznak, M., *E-model modification for case of cascade codecs arrangement.* system, 2011. **1**: p. 10.
58. Vozňák, M., *Non-intrusive speech quality assessment in simplified E-Model.* 2012.
59. Assem, H., et al. *Monitoring VoIP call quality using improved simplified E-model*. in *Computing, networking and communications (ICNC), 2013 international conference on*. 2013. IEEE.
60. Ren, J., et al. *Enhancement to E-model on standard deviation of packet delay*. in *Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on*. 2010. IEEE.
61. Botta, A., A. Pescapé, and G. Ventre. *On the statistics of qos parameters over heterogeneous networks*. in *IFIP Networking 2006 Workshop Towards the QoS Internet (To-QoS'2006)*. 2006.
62. Assem, H., et al. *Online estimation of VVoIP Quality-of-Experience via network emulation*. in *Signals and Systems Conference (ISSC 2013), 24th IET Irish*. 2013. IET.
63. Ott, J., I. Curcio, and V. Singh, *RTP Control Protocol (RTCP) Extended Report (XR) for RLE of Discarded Packets.* 2014.
64. Hines, A., et al. *Perceived Audio Quality for Streaming Stereo Music*. in *Proceedings of the ACM International Conference on Multimedia*. 2014. ACM.
65. Lakaniemi, A., J. Rosti, and V.I. Räisänen. *Subjective VoIP speech quality evaluation based on network measurements*. in *Communications, 2001. ICC 2001. IEEE International Conference on*. 2001. IEEE.
66. Koumaras, H., F. Liberal, and L. Sun, *PQoS assessment methods for multimedia services.* Chapter contribution in" Wireless Multimedia: Quality of Service and Solutions", Editors Dr. Nikki Cranley, Dr. Liam Murphy, IGI Global Pub.(Under Publication), 2008.
67. Cisco, *Reporting End-of-Call Statistics in SIP BYE Message*. 2008. http://www.cisco.com/c/en/us/td/docs/ios/ios_xe/voice_cube_-_ent/configuration/guide/vb_11099_xe.pdf

68. Castillo, I.B., *RFC 7118 The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)*. https://tools.ietf.org/html/rfc7118
69. *WebRTC 1.0: Real-time Communication Between Browsers - Interface Definition section*. 10 February 2015; Available from: https://www.w3.org/TR/2015/WD-webrtc-20150210/#interface-definition.
70. Gareiss, R. *The True Cost of Voice Over IP* 2009; Available from: http://www.webtorials.com/main/resource/papers/avaya/paper14/True_Cost_VoIP.pdf.
71. Carvalho, L., et al. *An E-model implementation for speech quality evaluation in VoIP systems*. in *Proceedings of the 10th IEEE Symposium on Computers and Communications*. 2005. IEEE Computer Society.
72. Sun, L. and E.C. Ifeachor. *Perceived speech quality prediction for voice over IP-based networks*. in *Communications, 2002. ICC 2002. IEEE International Conference on*. 2002. IEEE.