Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

API	Application Programming Interface
CPU	Central Processing Unit
DMTF	Dual-Tone Multi-Frequency
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
HTML5	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICE	Interactive Connectivity Establishment
IE	Internet Explorer
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IP	Internet Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MTU	Maximum Transmission Unit
NAT	Network Address Translation
PBX	Private Branch Exchange
PHP	Hypertext Preprocessor
PSTN	Public Switched Telephone Network
P2P	Peer-to-Peer
QoS	Quality of Service
RTP	Real Time Protocol
RTT	Round Trip Time
SCTP	Stream Control Transmission Protoco
SIP	Session Initiation Protocol
SMS	Short Message Service
SQL	Structured Query Language
SSL	Secure Sockets Layer
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TLS	Transport Layer Protocol
TURN	Traversal Using Relays around NAT
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WebRTC	Web Real Time Communications
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

capítulo 1

Introdução

A *World Wide Web (W3C)* ou simplesmente *web* é um meio de comunicação global que começou a ser pensado em 1945, sendo nesse ano apresentado o sistema *Memex*, as funcionalidades deste sistema era permitir seguir *links* entre documentos em microfilme. Sendo em 1960, apresentado um sistema semelhante em que as ligações eram feitas entre documentos de texto, sendo já falado em *hypertext* [1]. Todo o processo de criação da *web* de que hoje em dias dispomos foi evoluindo ao longo dos anos e continua a evoluir, sendo nos dias hoje imprescindível, nos diversos dispositivos desde o computador, ao *tablet* ou no *smartphone*. Nesse sentido, as empresas têm vindo a apostar na *web*, tendo como exemplo a *Google*, que criou uma série de serviços disponibilizados a partir de um *web browser*, tais como (*Gmail*, *Google Drive*, *Play Store*, *Chrome Apps*, etc.). Esta evolução marcou de igual forma as tecnologias que em conjunto com a *W3C* e a *The Internet Engineering Task Force (IETF)*, permitiram o desenvolvimento da tecnologia *Web Real Time Communications (WebRTC)*. Esta permite a comunicação em tempo real entre *browsers*, sem requerer a instalção de qualquer *plug-in*. *WebRTC* surgiu recentemente, mais propriamente em 2012 apresentada pela *Google*, tendo ocorrido o surgimento de várias soluções utilizado está *Application Programming Interfaces (APIs)*.

1.1 motivação e objectivos

O surgimento da tecnologia *WebRTC* e a sua normalização, têm se feito sentir devido ao contributo da *Google,W3C e o IETF*, desde o seu lançamento tem surgido diversas aplicações, sendo algumas delas apresentadas nesta dissertação.

Na Figura 1.1, é mostrado a principal motivação desta dissertação, que tem como principal objetivo construir uma solução que permita a interoperabilidade entre *WebRTC* e as redes legadas. A principal motivação deste projeto surge devido a ausência de uma forma de intercomunicar entre os diferentes ambientes, por exemplo interligar um ambiente totalmente *web*, com um ambiente *Session Initiation Protocol (SIP)*. A par do desenvolvimento desta dissertação foi desenvolvido o *Wonder* que foi um projeto europeu que tem com principal objectivo permitir a intercomunicação entre diferentes dominios, tais como dominio *web*, *IMS* e *SIP*. Outro dos objectivos deste projeto era criar uma *API*, que facilitasse o desenvolvimento de aplicações *WebRTC***wonder**

A possibilidade de interligar a tecnologia *WebRTC* com as redes legadas é um grande desafio, porque em cada caso são usadas tecnoligias muito diferentes. *WebRTC*, deixa em aberto a forma de sinalização ficando essa decisão para quem desenvolve a aplicação. Desta forma o desenvolvedor pode escolher a sinalização que prentende usar, o que torna possivel a intercomunicação entre dominios diferentes.

Neste projeto efetuou-se um estudo e uma análise das diversas librarias *SIP* existentes que permitissem a comunicação com as redes legadas, efetuou-se também um estudo sobre a tecnologia *WebRTC* e a informação trocada entre os diversos *peers* durante uma comunicação. Neste projeto foi

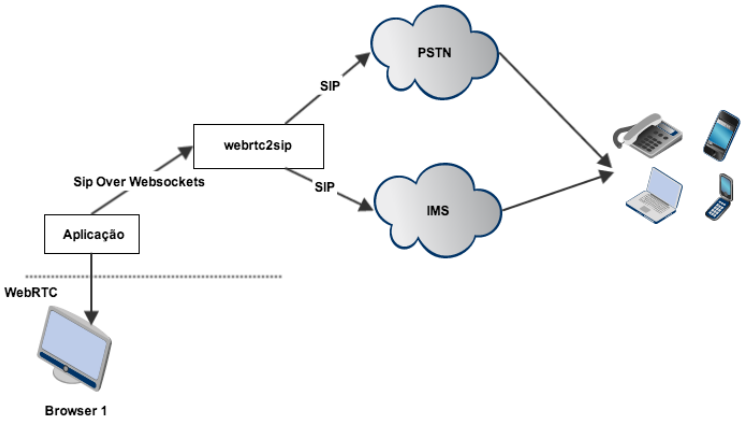

Figura 1.1: Motivação

feito um estudo as diversas librarias existentes e fazer uma analise comparativa entre ambas. Após a análise das diversas bibliotecas serão tomadas decisões sobre qual a libraria que se deverá usar para desenvolver esta aplicação que tem como objetivo permitir a comunicação entre redes legadas.

De forma a que este projeto tivesse sucesso foram definidas fases para o mesmo:

- Estudo e experimentação de diversas librarias *SIP* bem como a sua interação com a tecnologia *WebRTC*;
- Implementação e configuração do cenário a usar para desenvolvimento;
- Desenvolvimento do projeto *Wonder*;
- Desenvolvimento da aplicação e da prova de conceito;
- Teste e análise dos resultados.

De forma a provar o modelo prático desenvolvido foi desenvolvido a prova de conceito que permite provar que a solução está operacional. Sendo para isto desenvolvido uma aplicação que usa o projeto europeu *Wonde* que irá permitir provar que a comunicação entre *WebRTC* e redes legadas funciona corretamente.

1.2 ORGANIZAÇÃO DA DISSERTAÇÃO

capítulo 2

Estado de arte

O presente capítulo aborda os conceitos inerentes a este projeto bem com o seu estado de arte no momento actual. Primeiramente serão descritos os protocolos usados para trocar dados através de *data channel*. De seguida são apresentados alguns dos problemas referentes às comunicações *WebRTC*, sendo explicado o problema de *Network Address Translation (NAT)* sendo por seguinte explicado o que é o *WebRTC* e as suas *Application Programming Interfaces (APIs)*. Por fim é explicado o protocolo *SIP* que é usado na sinalização das comunicações *VoIP* sendo ainda apresentado toda a estrutura e componentes de uma arquitetura *IMS*.

2.1 procolos data channel

A tecnologia *WebRTC* para além de permitir a comunicação de *media (RTP)* entre dois *peers*, permite igualmente a partilha de dados entre dois canais de comunicação. Foi neste contexto que surgiu o conceito *data channels* da tecnologia *WebRTC*, sendo ela a troca de dados feita através de *peers connections*. Para ser efetuado a comunicação de dados entre dois utilizadores numa sessão são usados quatro procolos de para controlo e segurança da comunicação sendo eles:

- *Stream Control Transmission Protocol (SCTP)* para controlo das *streams*;
- *Datagram Transport Layer Security (DTLS)* para segurança dos dados;
- *User Datagram Protocol (UDP)* controlo da camada de transporte;
- *Interactive Connectivity Establishment (ICE)* controlo da camada de transporte.
- Meter referencias.

2.1.1 stream control transmission protocol(sctp)

SCTP é um protocolo da camada de transporte que têm como caracteristicas a garantia de entrega livre de erros, garantia que os dados não são duplicados e a fragmentação de dados de acordo com o *Maximum Transmission Unit (MTU)*. Sendo garantida a entrega sequêncial de mensagens do utilizador em multiplas *streams*, havendo de igual modo possibelidade de garantir a ordenação das mesmas. Existe ainda um mecanismo de tolerância a falhas sendo elas a nível de redes, quer a entrega de pacotes[2]. O protocolo têm um mecanismo para evitar congestionamento de dados e evita ataques de *flooding* e *masquerade* [3].

2.1.2 DATAGRAM TRANSPORT LAYER SECURITY(DTLS)

O DTLS é um procolo que garante a segurança no tráfego da rede, similar ao *"TLS over Datagram"*. Este garante a segurança do tráfego *web* e do tráfego *e-mail*, sendo usado por em protocolos como *IMAP* e *POP* [4]. O surgimento do *DTLS* deve-se ao facto de protocolos que usam *UDP* não terem qualquer mecanismo de segurança, ou seja, não é garantida a fiabilidade ou qualquer garantia da ordenação dos pacotes, desta forma o DTLS surgiu a partir do TLS sendo em parte a sua extrutura semelhante. O uso do DTLS não têm qualquer implicação quer a nivel de atraso dos pacotes quer a nivel de perda de pacotes [4].

2.1.3 USER DATAGRAM PROTOCOL(UDP)

UDP é um protocolo da camada de transporte que permite a troca de mensagens com o mínimo *overhead*. Este não possui qualquer mecanismo para garantir que os pacotes são entregues, ou seja, não há tolerância a falhas ou atrasos. *UDP* é usado em aplicações que necessitem de transportar pequenos pacotes de dados e para trocas de *media* ou seja em sessões *RTP*. Este fato deve-se ao *UDP* garantir uma rápida entrega dos pacotes de *media*. Para controlar as falhas na comunicação é necessario criar uma série de estruturas de controlo, tais como: sistemas de redução de largura de banda, retransmissões, etc [5].

2.1.4 SECURE REAL-TIME PROTOCOL (SRTP)- DATAGRAM TRANSPORT LAYER SECURITY (DTLS)

RTP é um protocolo de redes, utilizado em aplicações de tempo real, este protocolo permite a entrega de serviços fim-a-fim em tempo real como por exemplo audio/video ou dados. O RTP é usado em diversas aplicações tais como em ligações *peer-to-peer*, como por exemplo em comunicações, conferências áudio e vídeo, *Mixers and Translators*, e *Layered Encodings*. Um dos problemas do RTP é que não garante *Quality of service (QoS)* ou a entrega dos pacotes de dados, embora consiga detetar a sua perda e efetuar uma reconstrução temporal, a segurança da comunicação também não é garantida [6]. O SRTP é um protocolo que surgiu para garantir a segurança na comunicação RTP, tal como o RTP o SRTP garante a confidencialidade e a autenticação das mensagens. O DTLS surgio como uma extenção para o SRTP, visto que o SRTP não permite uma gestão de chaves nativas, sendo necessario uma gestão de chaves e parâmetros de negociação das mesmas, bem como a troca segura dos dados. A solução SRTP-DTLS foi definida para sessões de *media* em comunicações com dois ou mais participantes. Este mecanismo de segurança é usado na tecnologia *WebRTC* de forma a garantir a encriptação dos dados RTP e a gestão da chave dos participantes [7].

2.1.5 SESSION DESCRIPTION PROTOCOL(SDP)

SDP é um protocolo, usado para descrever a sessão entre dois participantes, este protocolo descreve as caracteristicas da sessão tais como detalhes do *media* tais como (*codecs*. Ao iniciar uma chamada *Voice over IP (VoIP)* ou um *streaming* de video, são trocados diversos parâmetros de sessão tais como endereços, tamanho do video, *codecs* usados, esta negociação é feita usando o protocolo SDP, que descreve as caracteristicas daquela sessão. A descrição das caracteristicas daquela sessão é feita usando o formato de um texto, em que é descrito por uma série de pares de atributo/valor, um por linha. A descrição da sessão é feita usando um conjunto de caracteristicas tais como a versão do protocolo usado e o criador da sessão bem como o seu nome, os endereços e portos usados na sessão para que seja possivel ao utilizador ligar-se a sessão.

Na tecnologia *WebRTC* é usado o SDP para permitir a troca de informação sobre a sessão, esta informação vai codificada dentro de um objecto denominado por *RTCSessionDescription*, este objeto é usado para descrever a sessão entre dois utilizadores. O inicio de uma sessão *WebRTC* é feita usando

um mecanismo de *offer* e *answer*, em que ao querer comunicar com outro utilizador é criado uma *offer* que contém entre outros parâmetros o sdp para aquela sessão, já no caso da *answer* é a resposta a *offer* que contém a informação para aquela sessão do utilizador que recebeu a *offer*. Esta troca de informação é feita para que possa existir uma *Peer Connection* entre os dois utilizadores para que a informação possa ser trocada [8].

2.1.6 INTERACTIVE CONNECTIVITY ESTABLISHMENT (ICE)

ICE é um protocolo que usa a tecnica de *¨hole punching¨* que têm como objetivo permitir a troca de *media* entre dois utilizadores em que exista *NAT* entre ambos. Esta tecnica foi criada para que dois utilizadores possam trocar *media* através de NAT, mas nem sempre esta tecnica é fiável nesses casos o protocolo ICE recorre ao protocolo TURN. ICE é uma solução para que possam ser trocadas *streams* de *media* entre dois utilizadores que estejam entre NAT, esta tecnica usa o modelo de *offer answer* e a descrição dos dados para a sessão são enviados no sdp, tais como *ip* e portos. Na tecnologia *WebRTC* é usado este protocolo para contornar os problemas de NAT, além deste mecanimos a tecnologia *WebRTC* usa um mecaninsmo para prevenir ataques de *Denial of Service (DoS)*, isto porque não é trocado *media* sem que sejam trocados todos os candidatos [9].

2.2 NETWORK ADDRESS TRANSLATION(NAT)

Um dos problemas com que a tecnologia *WebRTC* se depara é com problemas de NAT e firewall, isto porque os endereços IP dos pacotes com origem em redes internas não iram conseguir aceder a rede pública de internet. Desta forma a tecnologia *WebRTC* usa dois protocolos (*Session Traversal Utilities for NAT - STUN, Traversal Using Relays around NAT - TURN*), que irão ser descritos a seguir. Estes protoclos são usados em simultâneo com o protocolo ICE.

2.2.1 SESSION TRAVERSAL UTILITIES FOR NAT (STUN)

STUN é um protocolo criado para possibilitar a comunicação entre dois utilizadores que tenham NAT entre si. Este protocolo permite a descoberta do *IP* e do porto em que a comunicação esta a ser feita. As principais caracteristicas desde protocolo é que permite verificar a connectividade entre dois utilizadores, bem como a retransmissão dos pacotes. A tecnologia *WebRTC* usa este protocolo de forma a estabelecer a ligação entre dois utilizadores, para tal ser possivel no inicio da comunicação é enviado pacotes de teste de forma a descobrir o caminho que se encontra até ao outro utilizador, desta forma é descoberto os *IP's* e portos mapeados.

2.2.2 TRAVERSAL USING RELAYS AROUND NAT (TURN)

TURN é um protocolo usado para quando a comunicação com o ICE falha, estas falhas ocorrem devido aos utilizadores que estejam por trás de NAT não terem um comportamento bem definido. Quando ocorre esta situação a tecnologia *WebRTC* recorre ao protocolo TURN de forma a assegurar a comunicação entre os dois utilizadores. Ao acontecer uma falha na comunicação ICE é necessario recorrer a um *relay* que se encontra na rede pública da *internet*, que têm como função retransmitir os pacotes entre os dois utilizadores que estão por trás de NAT. A tecnologia *WebRTC* usa este protocolo de forma a permitir a comunicação entre dois utilizadores, desta forma o funcionamento de uma aplicação a correr no *web browser*, irá conter um cliente TURN e um servidor TURN, em que é feito um pedido ao servidor TURN, de um endereço IP público e um porto que irá funcionar como endereço de *relay* de transporte [10].

2.3 WEBRTC

WebRTC é um projeto *open source*, grátis e normalizado que permite aos browsers realizarem comunicações de tempo real entre si, sem requerer a instalação de qualquer programa ou plugin. O principal objetivo é desenvolver aplicações em *Web browsers*, com ajuda de APIs de JAvaScript e HTML5. Este Projeto encontra-se suportado pelos *Web browsers* Chrome, Firefox e Opera[11]. Desde cedo que as comunicações de tempo real **RTP**, ganharam um grande impulso tecnologico devido a dois organismos de normalização - **IETF** e **W3C**. Estes dois organismos, que têm trabalhado na normalização definiram duas: API W3C e a IETF RTCWeb. Têm por objetivo suportar o desenvolvimento de aplicações Web que corram em qualquer dispositivo, desde *desktop* a dispositivos móveis. Para além disso devem permitir o acesso seguro as componentes de cada um dos sistemas tal como *(*webcam), microfone [12]. Na Figura 2.1 é possivel ver a arquitetura da tecnologia *WebRTC*, onde se pode observar que é constituida por duas *APIs*: *WebRTC C++* e a *Web API*. A *WebRTC C++* é utizada por programadores dos *web browsers* enquanto que a *Web API* é usada por programadores que queiram desenvolver aplicações usando a tecnologia *WebRTC*. Na figura METER REF IMAGE

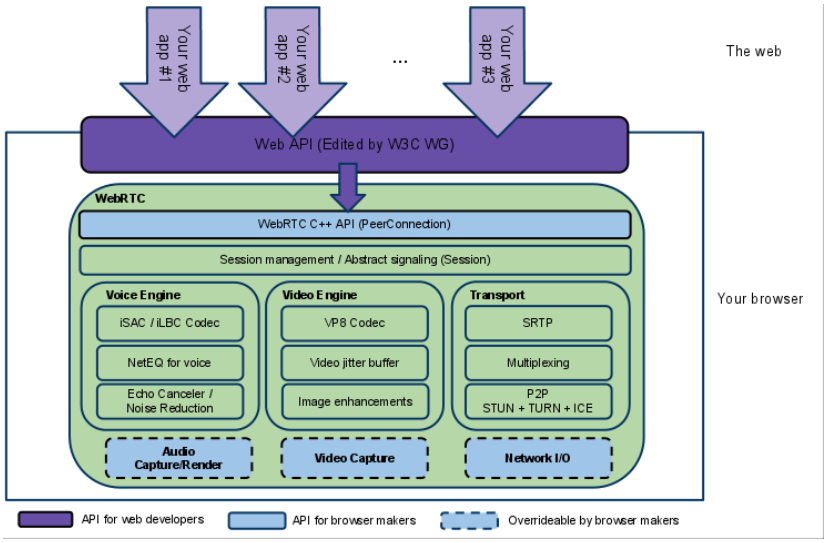

Figura 2.1: Arquitetura do WebRTC

é possivel ainda verificar outras componentes que são necessarias para a criação de aplicações *web* que usem a tecnologia *WebRTC*, essas componentes são os elementos responsavéis por a captura e transmissão do audio/video e naturalmente também existira uma componente responsavel pelo seu transporte. A tecnologia actualmente encontra-se implementada pelo *Chrome (Desktop e Android)*, *Firefox* e *Opera (Desktop e Android)* ainda não sendo suportado pelo o *IE(Microsoft)* nem pelo *Safari (Apple)* [12]. Como referido em cima a tecnologia *WebRTC* é uma tecnologia *opensource* cujo a documentação e código fonte poderá ser encontrado no svn da *Google* [13]. Como o trabalho desenvolvido incidirá sobre o desenvolvimento de uma aplicação *web* irá ser dado mais enfase a *Web API*, que serve para os programadores desenvolverem sobre essa tecnologia.

2.3.1 WEBRTC API

WebRTC API permite o desenvolvimento de aplicações *web*, desta forma para facilitar o desenvolvimento da aplicação final foi adoptado pela linguagem de programação *JavaScript*. Esta *API* é composta por três elementos fundamentais tais como: *PeerConnection, MediaStreams* e *DataChannel* [14] [12].

***GetUserMedia* permite:**

- Acesso aos recursos multimédia do utilizador, nomeadamente camara e microfone.

- Obter um *stream* que pode ser usado como fonte de um determinado elemento, por exemplo vídeo, ou ser transmitido através de uma ligação ponto-a-ponto (*PeerConnection*).

***PeerConnection* permite:**

- Estabelecer sessões de comunicação entre navegadores.
- NAT (*Network Address Translation*) usando ICE (*Internet Connectivity Establishment*) com STUN/TURN (*Session Traversal Utilities/ Traversal Using Relays around NAT*).
- Negociar codecs através de SDP (*Session Descritpion Protocol*), com um conjunto mandatório mínimo de *codecs* por forma a garantir interoperabilidade - G.711, G.722, iLBC and iSAC para áudio, e VP8 para vídeo.

***DataChannel* permite:**

- Troca de dados arbitrários (por exemplo binários) ponto-a-ponto: chat, transferência de ficheiros, etc.
- Comunicações seguras e inseguras baseadas no protocolo SCTP (*Stream Control Transmission Protocol*) e no DTLS (*Datagram Transport Layer Security*) para garantir segurança.
- Possibilidade de garantia de entrega ordenada de pacotes

2.3.2 COMPATIBILIDADE

Sendo a tecnologia *WebRTC* uma tecnologia recente e que não depende unicamente de si, uma vez que necessita de interagir com os diferentes *browsers*. É então necessario garantir que os *webbrowsers* suportam esta tecnologia, que têm como principal dependência a linguagem, *HTML5* que é o novo *standard* da linguagem *HTML*, esta evolução da linguagem veio introduzir melhorias quer a nivel de tratamento de erros quer a nivel de redução de *plugins* instalados no *webbrowser*. Com recurso a tecnologia *WebRTC* e a linguagem *HTML5* é possivel criar aplicações que reproduzam as *tags* de áudio/vídeo fornecidas pela a *API WebRTC* e desta forma reproduzir as *streams*. Na tabela 2.1 é possivel verificar as versões dos *webbrowsers* suportadas pelas *APIs WebRTC (MediaStream,Peer Connection,RTC Data Channels)*

Tabela 2.1: Versões dos *web browsers* que suportam às *APIs WebRTC* [15].

API	Internet Explorer	Firefox	Chrome	Opera
MediaStream	Sem Suporte	Suporta >v17	Suporta >v18	Suporta >v18
Peer Connection	Sem Suporte	Suporta >v22	Suporta >v20	Suporta >v18
RTC Data Channels	Sem Suporte	Suporta >v22	Suporta >v25	Suporta >v18

Na tabela 2.2 é possivel verificar quais os *web browsers* mais utilizados de forma a garantir a maior interoperabilidade entre eles, esta tabela mostra a percentagem de *web browsers* usados nos meses de Janeiro e Fevereiro de 2014.

Tabela 2.2: Percentagem dos *web browsers* usados [16].

2014	Internet Explorer	Firefox	Chrome	Opera	Safari
Janeiro	9,8%	26,4%	56,4%	4,0%	1,9%
Fevereiro	10,2%	26,9%	55,7%	3,9%	1,8%

2.3.3 CODECS

2.4 SINALIZAÇÃO

Um recurso que *WebRTC* não fornece é um canal de sinalização para estabelecimento de chamadas. Ela é deixada para o desenvolvedor, como se pode verificar na figura 2.2, esta componente não é especificada. Esta sinalização poderá ser por exemplo usando *WebSockets, IMS, XMPP*, para possibilitar a comunicação entre redes legadas a sinalização usada na solução irá ser *Sip Over WebSockets*.

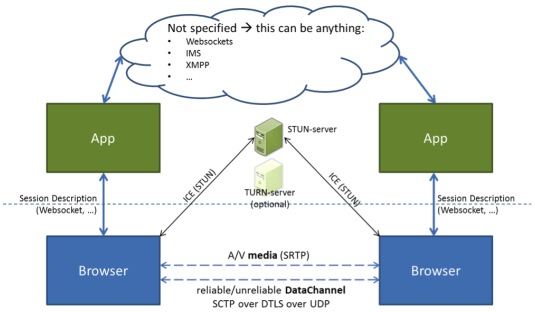

Figura 2.2: Sinalização Webrtc [15]

2.4.1 WEBSOCKETS

Desde o início do desenvolvimento da *web*, foram surgindo diversas aplicações, tendo como modelo a comunicação cliente/servidor. A comunicação neste tipo de aplicações é feita usando o mecanismo de pedido/resposta do protocolo HTTP, entre o cliente e o servidor. Este tipo de comunicação têm como principais problemas o abuso da comunicação HTTP, o que pode ter como principais problemas [17]:

- O servidor necessitar de ter um por cada cliente uma ligação TCP adjacente, uma para envio de mensagens e outra para receber;
- O protocolo usado exige uma sobrecarga do servidor, cada mensagem para o servidor tem um cabeçalho HTTP;
- Do lado do cliente é necessario ter um *script* para tratar das ligações ao servidor.

Com o aparecimento da tecnologia *AJAX*, as páginas tornaram-se mais dinâmicas. Porêm sempre que a página fosse actualizada, teria de existir um temporizador para que só após toda a comunicação com o servidor fosse terminada para mostrar a informação ao utilizador. No protocolo *WebSockets*, é usado uma comunicação mais simples, em que é usando uma única conexão TCP em ambas as direções. O surgimento deste procolo deve-se ao facto de substituir as comunicações bidirecional que usam HTTP, este protocolo é baseado em *sockets* e usa HTTP como camada de transporte. Desta forma é possivel usar *WebSockets* em soluções já existentes uma vez que funciona sobre as portas HTTP tais como 80 e 443, o que facilita a sua integração[17].

2.4.2 JAVASCRIPT SESSION ESTABLISHMENT PROTOCOL(JSEP)

2.4.3 SIP OVER WEBSOCKETS

O protoclo *Sip over WebSockets* é um sub protocolo do protocolo *webSockets* que foi definido anteriormente, que permite a troca de mensagens SIP entre cliente e servidor. *WebSockets* é um protocolo definido na camada da aplicação e é fiável, pelo que *Sip over WebSockets* também é um protocolo fiável. Concluindo *Sip Over WebSockets* é um protocolo semelhante ao *WebSockets* mas que transporta mensagens SIP encapsuladas. As principais entidades responsaveis neste protocolo são [18]:

- *SIP WebSocket Client* - esta entidade têm a capacidade de abrir ligações *WebSockets* de saída do servidor e que comuniquem com *Sip over WebSockets*;
- *SIP WebSocket Server* - esta entidade está a escuta de ligações dos clientes que comuniquem por *Sip over WebSockets*.

2.5 SESSION INITIATION PROTOCOL (SIP)

SIP é um protocolo da camada de aplicação baseado em texto, englobando elementos *Hyper-text Transfer Protocol (HTTP)* e *Simple Mail Transfer Protocol (SMTP)*. Como o próprio nome indica, foi desenvolvido para controlar sessões de comunicação, principalmente de voz e vídeo sobre IP, isto é, comunicações em tempo real. Normalmente os terminais podem ser endereçados com diferentes nomes, o que pode levar a executar comunicações de vários tipos de dados, muitas das vezes em simultâneo. O protocolo SIP permite que haja um controlo dessas sessões, ou seja, se considerarmos uma comunicação entre dois utilizadores (*user agents*), com a ajuda do protocolo SIP podem chegar a um acordo de que forma querem que a comunicação entre eles seja feita. Este protocolo é essencial para a criação, modificação e término de sessões que trabalham de forma independente da camada de transporte. Para além disso este protocolo pode trabalhar sobre *Transmission Control Protocol (TCP)*, *User Datagram Protocol (UDP)*, *Stream Control Transmission Protocol (SCTP)* e ainda *Transport Layer Security (TLS)* sobre TCP [19] [20].

2.5.1 ARQUITECTURA

Arquitectura

Este protocolo têm uma arquitectura que é definida por várias entidades ou sistemas, que são importantes para o seu funcionamento:

- *Domain Name System (DNS) Server*: este servidor executa duas funções bastante importantes. Por exemplo, se imaginarmos dois domínios diferentes, onde o *User agent* do domínio 1 quer iniciar uma sessão com o *User agent* do domínio 2, o *Proxy Server* 1 irá ao servidor DNS, fazer um pedido para saber onde se encontra o *Proxy Server* 2, para reencaminhar a sessão para o dominio 2. Da mesma forma o *Proxy Server* 2 poderia identificar o Servidor *backup* para o *Proxy Server* 1, caso este fosse abaixo.
- *Proxy Server*: esta entidade funciona tanto como cliente e servidor. É responsável pelo encaminhamento de mensagens, de um *user agent* ou de outro *Proxy Server*. Normalmente encontra-se ligado a uma base de dados *Location Service*. É responsável também por autenticar o utilizador e saber qual o seu perfil. Faz pedidos (*requests*) ao servidor DNS para saber o dominio a que pertence o sistema de destino, de forma a encaminhar as mensagens para o seu *Proxy Server*.
- *Redirect Server*: este tipo de servidor apenas responde a pedidos, não os encaminha. Também usa *Location Service* para obter informação sobre o *User Agent*. Essa informação é enviada para o *User Agent* numa classe de redireccionamento (3xx).

- *Location Service*: esta entidade é usada pelo *Redirect* ou *Proxy Server* para obter informação sobre o *User Agent*, tais como registos de SIP, informação sobre presença, ou outro tipo de informação sobre a localização do *User Agent*.
- *User Agent*: é um dispositivo terminal. Um dos propósitos do protocolo SIP é estabelecer sessões entre dois *User Agents*

2.5.2 TIPO DE MENSAGENS

O protocolo SIP faz uso de diferentes tipos de mensagens para troca de informação entre os dois utilizadores desta forma este protocolo usa dois tipos de mensagens *Request Messages* e *Response Messages* que irão ser explicados a seguir [20].

Request Messages

- *INVITE*: método para estabelecer uma sessão de *media* entre dois *User Agents*. Este tipo de mensagem pode conter informação sobre a sessão de quem faz o ÏNVITEë também pode conter a lista de recursos necessários para o estabelecer uma sessão. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards*.
- *REGISTER*: método usado pelo *User Agent* para notificar a rede SIP do seu *Uniform Resource Identifier (URI)*.Os campos mandatórios do header desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards*.
- *BYE*: método que é usado para terminar uma sessão que tenha sido estabelecida. É uma mensagem fim-a-fim, por isso só é gerada por *User Agents*. Os campos mandatórios do header desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards*.
- *ACK*: é o método usado para confirmar as mensagens *INVITE* que foram enviadas. Todas as mensagens de outro tipo não são confirmadas. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards*.
- *CANCEL*: este tipo de mensagem pode ser enviado por *Proxy Servers* ou *User Agents*, para terminar tentativas de chamadas, ou *INVITES* pendentes. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards*.
- *OPTIONS*: este método é usado para saber quais as capacidades de *User Agents* ou servidores e saber qual a sua disponibilidade. Este método nunca é gerado por servidor *proxy*. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards*.
- *SUBSCRIBE*: é usado por um *User Agent* para estabelecer um tipo de subscrição, para receber noticações, sobre o método (*NOTIFY*) de um determinado evento. Se este tipo de subscrição for bem sucedida é estabelecida uma ligação entre *User Agent Client (UAC)* e o *User Agent Server (UAS)*. Os campos mandatórios do *header* desta mensagem são *Via, To, From, Call-ID, CSeq, Contact, Max-Forwards, Event, Allow-Events*.
- *NOTIFY*: método usado pelo UA para transmitir informação relativa a um determinado evento. Este tipo de mensagem é sempre enviado num diálogo entre subscritor e o notificador. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards, Event, Allow-Events, Subscription-State*.
- *PUBLISH*: é usado por um UA para enviar informação de um evento para um servidor conhecido como *Event State Compositor*. Normalmente este tipo de método é útil quando existem vários dispositivos a usar o mesmo *Address of Record (AOR)*. Assim, neste caso, seria necessário haver uma subscrição individual para cada um dos dispositivos e em vez disso o UA faz uma subscrição ao *Event State Compositor*. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards, Event, Allow-Events, Subscription-State, Expires Min-Expires*.

- *REFER*: este método éusado por um UA para fazer um pedido a outro UA para ter acesso a um recurso URI ou URL. Esse recurso é identificado pelo URI ou URL no campo *Refer-To* do *header*. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards, Refer-To*.
- *MESSAGE*: método usado para enviar mensagens instantâneas (IM) sobre SIP. Este tipo de mensagens devem ser entregues o melhor possível em tempo real. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards*.
- *INFO*: este tipo de método é usado para enviar informação de sinalização de uma chamada de um UA para outro com que tenha estabelecido uma sessão *media*. É também uma mensagem fim-a-fim nunca iniciada por um *Proxy Server*. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards, Info- Package*.
- *PRACK*: este método é usado para confirmar a recepção de respostas fiáveis provisórias transmitidas (1xx). Para as outras classes de respostas o método usado é o método ACK. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards, RAck*.
- *UPDATE*: usado para modificar o estado de uma sessão SIP,sem mudar o estado do diálogo entre dois *User Agetns*. Os campos mandatórios do *header* desta mensagem são *Via ,To, From, Call-ID, CSeq, Contact, Max-Forwards, Contact*.

Response Messages As mensagens de resposta são enviadas por UAS e servidores SIP em resposta a pedidos de um UAC. Este tipo de mensagens são divididas em seis tipos de classes diferentes, que são as classes já existentes do protocolo HTTP e uma classe diferente que foi adicionada para o protocolo SIP. Estas classes são classificadas como [20]:

- *Informational* (1xx): Indica o estado de uma chamada antes de a terminar. É também conhecida como resposta provisória.
- *Success* (2xx): Pedido bem sucedido. Caso fosse para uma mensagem *INVITE*, devia ser enviado um *ACK*, caso contrário deve ser parada a retransmissão do pedido.
- *Redirection* (3xx): Servidor retorna locais possíveis. O cliente deve enviar o pedido para outro servidor.
- *Client Error* (4xx): O pedido falhou devido a um erro do cliente. O cliente pode voltar a retransmitir, se o mesmo estiver reformulado de acordo com a resposta.
- *Server Failure* (5xx): O pedido falhou devido a erro de servidor. O cliente deve tentar reenviar o pedido para outro servidor.
- *Global Failure* (6xx): Pedido falhou. Não deve ser enviado para nenhum outro servidor.

2.5.3 INTEGRAÇÃO DO PROTOCOLO SIP NUMA ARQUITECTURA IMS

O protocolo SIP foi escolhido como o mecanismo de controlo de sessões na arquitetura IMS. Para realizar uma operação numa arquitectura IMS, é necessário que se cumpram alguns pré-requisitos, que estão representados na figura METEr REF ***IMS Service Contract*** É necessário que o administrador de uma arquitectura IMS autorize um cliente a utilizar os seus serviços. Para isto acontecer é necessário que um cliente tenha um contrato com a rede IMS do operador. Este contrato dá autorização ao cliente para receber e estabelecer chamadas sobre a rede wireless. ***Getting IP Access Connectivity*** Este processo passa por um sistema terminal IMS obter um endereçoo IP, desta forma tem acesso a uma rede IP *Connectivity Access Network (IP-CAN)*. Como existem vários tipos de redes *IP-CAN*, o registo e o acesso às mesmas também é diferente e a forma como o endereçoo de IP é obtido. Existem vários tipos de IP-CAN desde *Digital Subscriber Lines (DSL), Dial-up Lines, enterprise Local Access Networks*. Em termos de redes *Wireless*, temos redes *General Packet Radio Service (GPRS)* e *Wireless Local Access Networks (WLAN)*. Quando a *IP-CAN* não é uma rede GPRS, o protocolo utilizado para

configurar um terminal IMS é o protocolo *Dynamic Host Configuration Protocol (DHCP)* ou *DHCPv6*, dependendo se a configuração é IPv4 ou IPv6. Numa rede GPRS a configuração de um terminal IMS, passa por uma série de procedimentos necessários. Estes processos estão ilustrados na figura METER REF

Quando estes processos estiverem completos o terminal pode enviar uma mensagem do tipo Activate PDP Context message.

2.6 IP MULTIMEDIA SUBSYSTEM (IMS)

IP Multimedia Sub-system, IMS, consiste numa arquitetura aberta e normalizada defina pelo 3GPP, com o objetivo de providenciar diversos serviços de multimédia sobre uma infra-estrutura permitindo connectividade e comunicação IP entre as diversas entidades. Para isso reutiliza protocolos definidos pelo IETF, tais como SIP e o Diameter. Inicialmente foi desenhado para comunicações de redes sem fios, após serem percebidas as suas vantagens passou a ser uma peça fundamental na construção de redes de próxima geração(RPG ou NGN - *Next Generation Networks*) onde a convergência fixo-móvel vem potenciar a criação de novos serviços e modelos de negócios. Desta forma a sua adopção implica uma nova abordagem as redes de telecomunicações, sendo necessario algumas alterações nas infra-estruturas organizacionais para poder usufruir das suas vantagens.

2.6.1 ARQUITETURA IMS

A arquitetura IMS foi desenvolvida para controlar e gerir redes, principalmente redes móveis, convergindo mais tarde para redes fixas. É fundamental para serviços multimédia baseados em IP. Esta arquitetura encontra-se dividida em várias funções que por sua vez estão ligadas por várias interfaces normalizadas. Sendo estas interfaces o ponto de referência, que determina em que partes da arquitetura as diferentes entidades actuam e que protocolos usam para a sua comunicação. Sendo estas entidades classificadas em seis categorias como mostra a figura 2.3 [21] [22]:

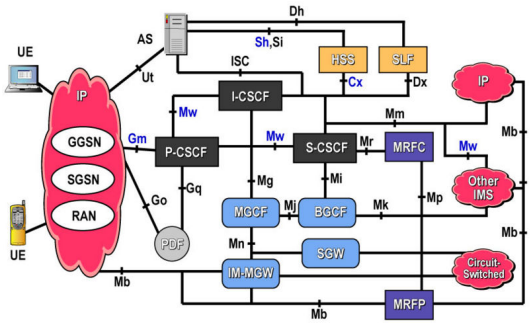

Figura 2.3: Arquitectura geral IMS e pontos de referência [23]

- Gestão de sessões e routing - *Call Session Control Function* (**CFSCs - S, I, P e E**) sevidores SIP ou encaminhadores de mensagens SIP;
- Base de dados - *Home Subscriber Server* (**HSS**) repositórios com informação, permanente e dinâmica, dos utilizadores do sistema IMS;
- Elementosde interfuncionamento *Interworking ElementsBreakout Gateway Control Function* (**BGCF** - controlador de saida do domínio do IMS), *Media Gateway Controller Function* (**MGCF** - controlador de elementos de saída do domínio IMS), *IMS-Media Gateway* **IMS-MGW** - elemento de média, *Signalling Gateway* (**SGW**);

- Serviços - *Application Service* (**AS**), *Media Resource Function Controller* (**MRFC**), *Media Resource Function Processor* (**MRFP**);
- Entidades de suporte - *Security Gateway* (**SEG**), *Interconnection Border Control Function* (**IBCF**), **LRF**;
- Charging (distribuição de endereços pelas diferentes entidades).

Estas categorias podem ser englobadas por camadas principais: *Transporte Plane*; *Control Plane ou IMS Layer*; *Service or Application Plane* como é possivel ver na figura 2.4.

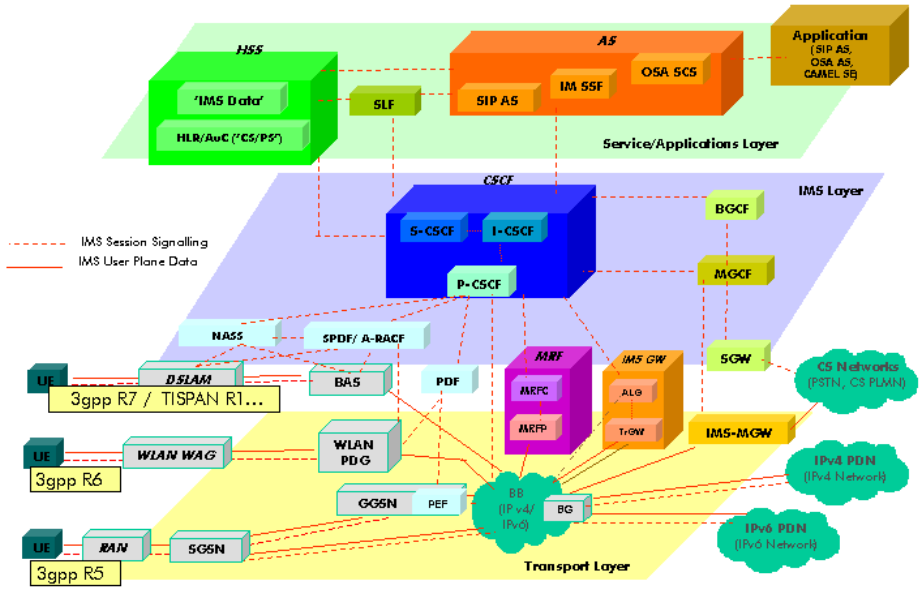

Figura 2.4: Arquitectura geral IMS [24]

Transport Plane

O Transport Plane contém funções que controlam os acessos à rede IMS e é responsável por fornecer conectividade IP aos clientes (*User Equipment*). Este plano é constituído pelas entidades das categorias de serviços e elementos de interfuncionamento, onde entram:

- Multimedia Resource Function Processor (MRFP) - Executa várias funções *media* e juntamente com MRFC é responsável pelo uso de *Media Resource Servers*. Normalmente as funções que precisam destes servidores são conversores *text-to-speech*, conferência e *interactive response (IVR)*, permitindo assim a garantia dos melhores recursos para estas funções serem executadas [21];
- Media GateWay (IMS-MGW) - Interface entre IMS *media* e da *legacy network media (Public land mobile network (PLMN), PSTN)* [21].

Control Plane ou IMS Layer

Este plano é responsável por encaminhar os sinais das chamadas de acordo com os critérios de filtragem. Para além disso é também o plano responsável por garantir o acesso aos *Application Servers*, desta forma consegue fornecer vários serviços, como presença e serviço de mensagens, aos seus clientes. Esta camada é constituída pincipalmente pelo HSS e CSCF, contendo também BGCF e MGCF [21], que a seguir se explica:

- Proxy-CSCF (P-CSCF) - é um SIP-Proxy e é o primeiro ponto de contacto com UE. Pode estar localizado tanto na *Home Network* como na *Visited Network*. Este entidade utiliza DNS lookup

para descobrir o endereço do I-CSCF que irá atender o pedido. Executa funções como autorizar determinados serviços para garantir um determinado nível de QoS;

- Interrogation-CSCF (I-CSCF): Este é o primeito ponto de contacto com o operador. Tem como função obter o endereço do próximo salto que pode ser ou o S-CSCF ou um Application Server para encaminhar as mensagens SIP;
- Serving-CSCF (P-CSCF): Está localizado na home network e é um dos pontos mais cruciais da arquitectura IMS. É responsável por fazer autenticação dos utilizadores, encaminhar os vários serviços para o destino, baseando-se nos critérios recebidos do HSS;
- Home Subscriber Server(HSS)-É a base de dados que contém informação que diz respeito ao cliente, desde a sua autenticação até aos serviços que usa;
- Media Gateway Control Function (MGCF) - Permite completar a conversão de *media* da *legacy network* e sinalização de *media* IMS;
- Breakout Gateway Control Function (BGCF) - Identifica o MGCF e MGW apropriados para suportar uma instância específica de saída do domínio IMS.

Application Plane Este plano contém os *Application Services* que suportam SIP e DIAMETER. Este plano é responsável por receber os pedidos do S-CSCF e fornecer e encaminhar serviços consoante os pedidos que foram feitos. Caso seja necessária informação adicional o servidor pode comunicar directamente com o HSS [21].

2.7 SUMÁRIO

capítulo 3

Soluções existentes

Neste capítulo serão apresentadas algumas soluções existentes. Começando por serem apresentadas aplicações SIP/IMS existentes. De seguida são apresentadas as *stacks* SIP/IMS existentes. Na secção seguinte é apresentado uma analise a algumas soluções *WebRTC*, mostrando a forma como as *APIs* podem ser usadas de diferentes modos. Em seguida é apresentado algumas bibliotecas *JavaScript* que permitem a implementação de serviços de forma mais simples. Por fim são analisadas algumas *gateways* SIP, que permitem a interoperabilidade entre ambientes *web* e ambientes SIP.

Após se efetuar uma analise de cada componente são apresentadas as decisões que foram tomadas para este projeto.

3.1 aplicações sip e sip/ims

Analisaram-se vários clientes SIP/IMS. Esta análise centrou-se mais pormenorizadamente sobre os serviços que cada uma delas suportava, os codecs que poderiam usar, tanto a nível de voz e de áudio.

3.1.1 ims communicator

3.1.2 linphone

É um cliente SIP que simula um telefone VoIP e é open-source. Este cliente encontra-se atualmente disponível para as várias plataformas desktop Linux, Windows, MacOSX e para plataformas móveis Android, iPhone e Blackberry(sem suporte de vídeo). O Linphone está disponível sob a licença GNU GPLv2 [25]. Em termos de segurança usa encriptação na parte de voz e de vídeo com a ajuda dos protocolos SRTP e zRTP [26]. Suporta a seguintes funcionalidades

- Histórico de chamadas;
- Lista de contactos;
- Notificação de uma chamada fiável ou notificação de chat (com servidor compatível);
- Estatísticas de chamadas;
- Suporte a múltiplas chamadas simultaneamente;
- Modo de Espera;
- Reencaminhamento;

- Conferência em chamada;
- Chamadas de voz e vídeo;
- Mensagens instantâneas;
- Presença.

Este cliente suporta Speex, G711, GSM, G722 que são codecs de áudio, podendo ainda suportar, devido a plugins adicionais, os AMR-NB, SILK, G729 e iLBC. No que diz respeito aos codecs de vídeo este suporta VP8 (WebM), H263, H263-1998, MPEG4, Theora e H264, garantindo resoluções de QCIF(176x144) e SVGA(800x600). Para além disso tem suporte a IPv6 [26].

3.1.3 SOFIA-SIP

3.1.4 IMSDROID 2.X

É uma aplicação cliente SIP/IMS e usa *Doubango Framework*. Esta é uma framework *3GPP IMS/LTE* que foi desenvolvida pela *Doubango Telecom* que é uma empresa focada em projetos *open source*. Este cliente têm as seguintes funcionalidades:

- Chamada de voz;
- Video-Chamada;
- Chamada em espera;
- Reencaminhamento de chamadas;
- Mensagens SIP (*Short Instant Message*);
- MSRP Chat (Chat 1-para-n);
- Histórico de Mensagens e Chamadas;
- Short Message Service (SMS);
- Transferência de ficheiros;
- Monitorização de chamadas;
- STUN.

Para além destas funcionalidades, este cliente tem outras características que devem ser levadas em conta. Funciona em redes móveis como LTE, UMTS e EDGE Networks, em termos de segurança tem suporte para TLS, SRTP e RTCP, além disto este cliente está sob a licença GNU GPL v3, havendo a hipótese de pedir uma licença para fins comerciais non-GPL **imssdroid imssdroid2** Este cliente suporta alguns tipos de codecs como H264, Theora, H.263, H.263-1998, H.261, VP8. Neste caso o utilizador pode escolher o tipo de qualidade que quer para uma ligação entre *Low, Medium, High* a nível aplicacional. Desta forma é possível obter uma melhor qualidade de vídeo, com menos atrasos e menos utilização de CPU **imssdroid imssdroid2** Em relação às funcionalidades que dizem respeito a áudio, o utilizador também pode escolher o tipo de qualidade que quer para uma ligação áudio, pois este cliente oferece essa opção, suportando assim vários codecs de áudio como AMR-NB, GSM, PCMA, PCMU e Speex-NB **imssdroid imssdroid2**

3.1.5 IDOUBS

3.1.6 JITSI

É uma aplicação cliente que suporta vários protocolos e redes, tais como SIP, XMPP/Jabber, AIM/ICQ, Windows Live, Yahoo!, esta aplicação suporta *codecs* de áudio Opus, SILK, Speex, G.722,

PCMU/PCMA (G.711), iLBC, GSM, G.729, e em vídeo suporta H.264, H.263-1998 / H.263+ e VP8(a ser desenvolvido neste momento). Esta aplicação esta desenvolvida com licença *open source* com licença LGPL. Este cliente têm como principais funcionalidades [27] [28]:

- Chamada de voz/vídeo;
- Multi-chat;
- Transferência de ficheiros;
- Encriptação OTR;
- Notificação de chamadas perdidadas;
- *Drad and Drop* para transferência de ficheiros;
- Integração com *Microsoft Outlook* e *Apple Address Book*;
- Suporta contactos do *Google*;
- *IPV6* em SIP e XMPP;
- Pesquisa de directórios rodando sobre TCP/IP, usando *Lightweight Directory Access Protocol (LDAP)*;
- Encriptação de chamadas usando SRTP, ZRTP e SDES em XMPP e SIP;
- Encriptação de messagens instantáneas usando OTRv4.

Entre outras, as caracteristicas principais desta aplicação residem na capacidade de encriptar chamadas quer de áudio/vídeo quer em conversações de chat. Isto devido ao suporte de SRTP, ZRTP, SDES quer no protocolo SIP quer no XMPP.

3.1.7 PHONO

3.2 STACKS SIP E SIP/IMS

3.2.1 SIPML5

3.2.2 QOFFEESIP

3.2.3 SIP-JS

3.2.4 PJSIP

3.3 SOLUÇÕES WEBRTC EXISTENTES

3.4 BIBLIOTECAS JAVASCRIPT

3.5 GATEWAYS PARA SIP

Gateways para SIP servem com o próprio nome indica, permitir a comunicação de clientes *WebRTC* com clientes *SIP*, ou seja, permite a interação entre dois ambientes totalmente diferentes um ambiente totalmente Web com um ambiente totalmente SIP e vice-versa. A partir destes mecanismo é possivel ter interoperabilidade entre diversos sistemas o que pode ser um grande avanço nas telecomunicações. De seguida ira ser analisado algumas *gateways* SIP.

3.5.1 ASTERISK

Asterisk é um *software opensource* que pode ser utilizado de diversas formas tais como *Media gateway, Media Server*, correio de voz e *IP Private Branch Exchange (PBX)*. Este *software* inclui diversos recursos disponíveis em sistemas *PBX* proprietarios. Sendo possivel aos usuários criar novas funcionalidades escrevendo *scripts* que serão carregados através dos módulos [29].

Na versão 11 do *Asterisk* foi intruduzido o suporte a tecnologia *WebRTC* a partir do módulo *res_http_websocket*, o que veio a permitir aos desenvolvedores desenvolver soluções que interajam com *WebRTC* e o servidor *Asterisk* a partir de *WebSockets*. Foram também adicionadas funcionalidades para suportar os protocolos de *ICE,STUN* e *TURN* para lidar com os problemas de *NAT* [29].

A partir da versão 10 do *Asterisk* foi adicionado uma biblioteca, a *libsrtp* foi adicionada para garantir a segurança nas comunicações *RTP*, visto que para que haja comunicação *WebRTC* é necessario ter um canal seguro de comunicação. *Asterisk* suporta uma ampla gama de *Voice over IP protocolos*, incluindo o *Session Initiation Protocol (SIP), o Media Gateway Control Protocol (MGCP) e H.323*. O *Asterisk* pode interoperar com a maioria dos telefones *SIP*, atuando tanto como servidor como uma gateway entre telefones *IP* e da rede *PSTN* [29].

3.5.2 WEBRTC2SIP

Webrtc2sip é uma gateway para SIP desenvolvida pela *Doubango* que permite a interoperabilidade entre a comunicação *web* e a comunicação SIP. Desta forma o *web browser* poderá comunicar com qualquer tipo de aplicações ou dispositivos que usem comunicação SIP. O que possibilita que uma aplicação totalmente *web* comunique com qualquer tipo de rede *PSTN, SIP*.

Como se pode verificar na figura 3.1 o *webbrowser* usa dois tipos de *stack*, a *SIP Stack* e a *SDP Stack* que irão comunicar através de *SIP* com o *SIP Proxy*. Quanto ao *WebRTC* irá receber e enviar o *media* através do *RTCWeb Breaker* que é o responsavél por fazer a conversão das *streams media*, de forma a que possa haver comunicação em que os dispositivos finais não suportem as características dos protocolos *ICE* e *DTLS/SRTP*. O *WebRTC* pode usar ainda outra componente do *webrtc2sip*, que é o *media coder*, que permite a comunicação entre diferente dispositivos que suportem diferentes codecs, de forma a garantir a interoperabilidade entre os diferentes dispositivos.

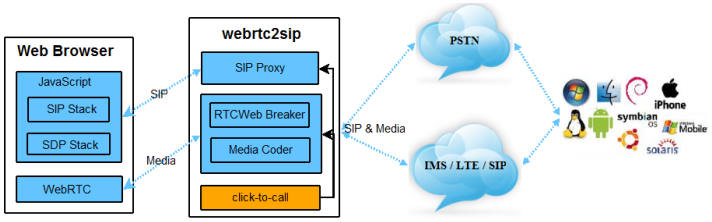

Figura 3.1: Arquitetura do sipML5[30]

Na figura 3.2 é possivel ver o funcionamento do *RTCWeb Breaker*, que irá fazer a conversão de *streams* de *media* desta forma esta componente negoceia e converte o fluxo de *media* para permitir a interoperabilidade. Se um servidor não suportar *ICE*, isto significa que o *RTCWeb Breaker* deverá ser capaz de conectar o terminal *SIP* com o *browser*.

Figura 3.2: Arquitetura RTCWeb Breaker[31]

A utilização de diferentes *codecs*, é um problema que surge devido ao facto das diferentes aplicações usarem o *codec* mais apropriado a sua utilização. Enquanto que o *Chrome, Mozilla, Opera*, usam o

VP8, como *codec* de vídeo a *Microsoft* irá usar o *H.264*. Quanto ao áudio a normalização do *RTCWeb* definiou dois *Mandarory To Implement (MTI)* que foram o *Opus* e *G.711*. Na figura 3.3 é possível ver o funcionamento do *Media Coder* que irá possibilitar a comunicação de áudio/vídeo entre dispositivos que usem diferentes *codecs*.

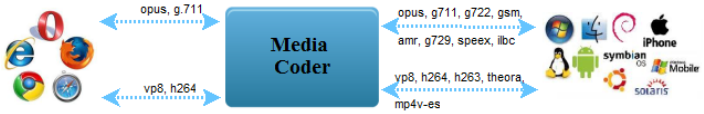

Figura 3.3: Media Coder sipML5 [31]

3.6 EXPERIMENTAÇÃO E SELEÇÃO DE SOLUÇÕES EXISTENTES

3.7 SUMÁRIO

capítulo 4

Interoparabilidade entre WebRTC e redes legadas

A par do desenvolvimento da solução que permiti-se interoperabilidade entre *WebRTC* e redes legadas, foi desenvolvido um projeto denominado *Wonder*. Este era um projeto europeu que tinha como objectivo o desenvolvimento de uma *framework* que facilita-se o desenvolvimento de aplicações *WebRTC*, dando um enfoque especial a interoperabilidade e sinalização entre diferentes dominios, tais como domínios *IMS, Web-centric, SIP, Vertx, Node.js*.

Na figura 4.1 é possivel ver um esquema de interação dos serviços *WebRTC*. Num cenário de comunicação entre dois *peers* em que um deles está ligado a uma rede legada,

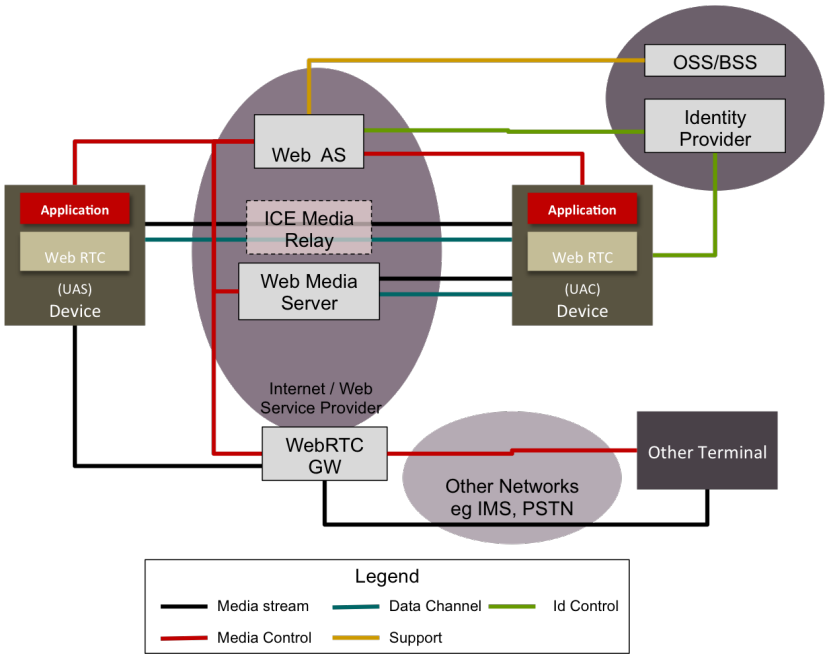

Figura 4.1: Serviços base WebRTC

Na figura 4.2 ....

De seguida irá ser explicada este projeto mais especificadamente bem como as suas componentes, e a sua utilização na construção da solução que permitisse a interoperabilidade entre redes legadas.

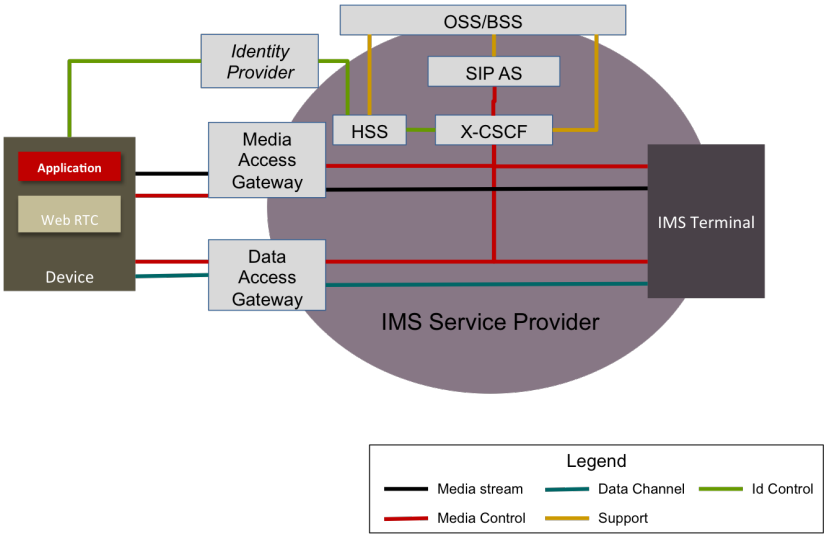

Figura 4.2: Serviços base WebRTC numa arquitetura IMS

4.1 WONDER

Wonder é uma *framework*, que facilita o desenvolvimento de aplicações *WebRTC*, na figura 4.3, é possivel observar as diferentes classes por que é constituida, que serão explicadas de seguida.

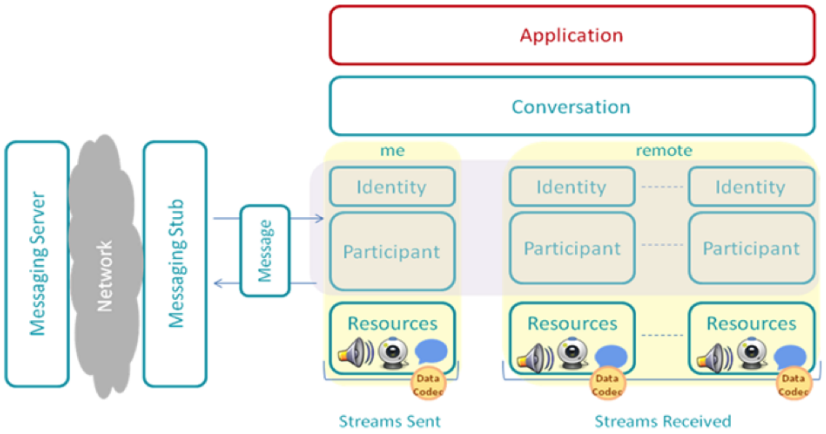

Figura 4.3: Principais classes wonder

A classe ***Identity*** representa um usuário e contêm todas as informações necessárias para apoiar serviços de conversação, incluindo o terminal de serviço a pilha de protocolos (*Messaging Stub*) que será utilizada para estabelecer um canal de sinalização como, um servidor de mensagens do domínio *Identity*. A entidade *Identity* estende o conceito de identidade atual definida na especificação *WebRTC* [32], para apoiar a interoperabilidade contínua usando a sinalização *on-the-fly*. O *MessagingStub* implementa a pilha de protocolos usado para este mecanismo de comunicação.

A classe ***Message*** é usado para trocar todos os dados necessários para a utilização do midia e dados de conexão entre pares através do servidor de mensagens. Pode ser usado ainda para outros fins, por exemplo, gestão de informações de presença.

A classe ***Conversation*** gerencia todos os participantes, incluindo a gestão do mídia e as conexões de dados.

A classe ***Participant*** processa todas as operações necessárias para gerenciar a participação de uma identidade(usuário) em uma conversa incluindo as funcionalidades *PeerConnection* do *WebRTC*. O participante local esta associado a uma *identity* local, enquanto que o participante remote esta associado a outra *identity* remota.

A classe ***Resource*** representa os ativos digitais que são compartilhados entre os diferentes participantes numa conversação, incluindo voz dos participantes, video, fotos, videos, etc. Esses activos são geralmente gerenciados pelo participante que os possui. Outro tipo de recursos como chat não são gerenciados pelo participante mas sim pela conversa.

A classe ***Data Codec*** é usado por recursos que são compartilhados a partir do *Data Channel*, como por exemplo chat. O *Codec* pode ser também transferido *on-the-fly* pelos participantes na conversação.

4.2 SINALIZAÇÃO *on-the-fly*

O termo *on-the-fly* é um termo que define a forma de funcionamento dinâmico de uma aplicação e não algo estatico esperando que algo aconteça. Desta forma todo o processo de funcionamento desta biblioteca funciona com base neste conceito. Na figura 4.4 é possivel ver o modo de funcionamento de toda a *API*, nesta figura é possivel ver o processo de ligação entre dois peers.

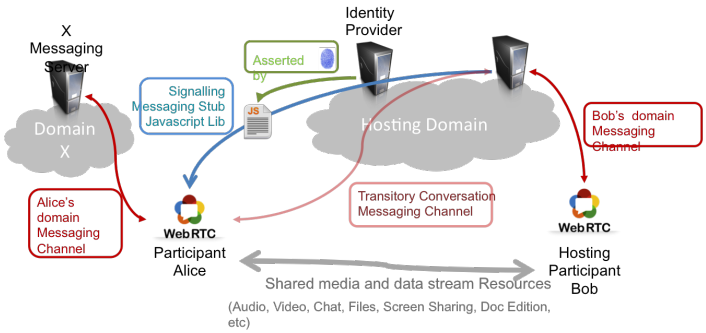

Figura 4.4: Sinalização *on-the-fly*

Numa conversação entre dois peers, Alice e Bob, em que ambos têm dominios diferentes. A Alice para comunicar com o Bob têm de interrogar o provedor de identidades que não é mais que uma *API REST*, que vai fornecer toda a informação associada aos utilizadores inclusive os seus domínios associados, esta informação está toda guardada numa base de dados. Após contactar o provedor de indentidades a Alice irá saber qual o domínio associado ao Bob e irá descarregar o seu *MessagingStub* associado e assim possibilitar toda a comunicação de sinalização entre ambos. Depois de trocada toda a informação de sinalização a Alice e Bob irão poder comunicar entre si por ligação de *PeerConnection*. Este processo é todo feito dinamicamente dai o uso do termo *on-the-fly*.

4.3 *codecs on-the-fly*

Neste projeto foi dado também um enfoque especial aos *codecs* de forma a possibilitar que utilizadores que usem diferentes *codecs* possam comunicar entre si. Na figura 4.5 é possivel ver o esquema de funcionamento relativo aos mesmos, em que numa conversação entre dois *peers*, Alice e Bob, após estar toda a sinalização feita entre ambos, para que seja possivel a comunicação entre ambos por um canal de *DataChannel* eles terão de usar o mesmo *codec* para comunicar entre si. Sendo assim este mecanismo funciona com base no conceito *on-the-fly* que irá permitir que toda a troca de informação para que seja possivel a comunicação por um canal de dados entre dois *peers* seja feita automaticamente.

Na figura a cima, é possivel verificar o funcionamento do mesmo, em que a Alice comunica com o *MessagingServer* para saber qual o *codec* que o Bob utiliza para a comunicação de dados, após esta informação estar trocada a Alice sabe qual a libraria que irá ter que descarregar para comunicar com o Bob utilizado o canal de *DataChannel*.

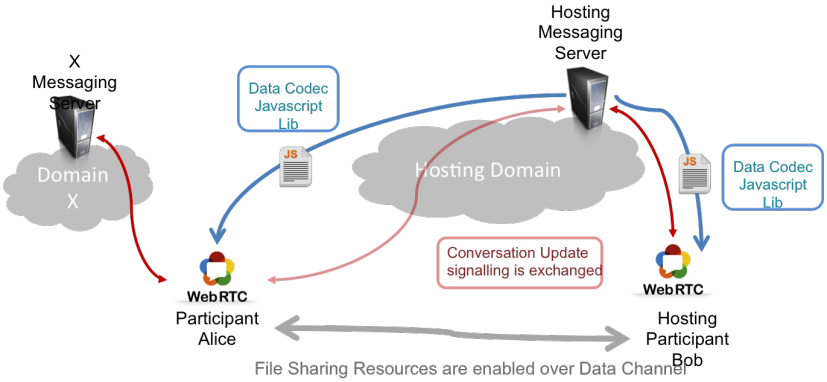

Figura 4.5: *Codecs on-the-fly*

4.4 SINALIZAÇÃO *multi-party*

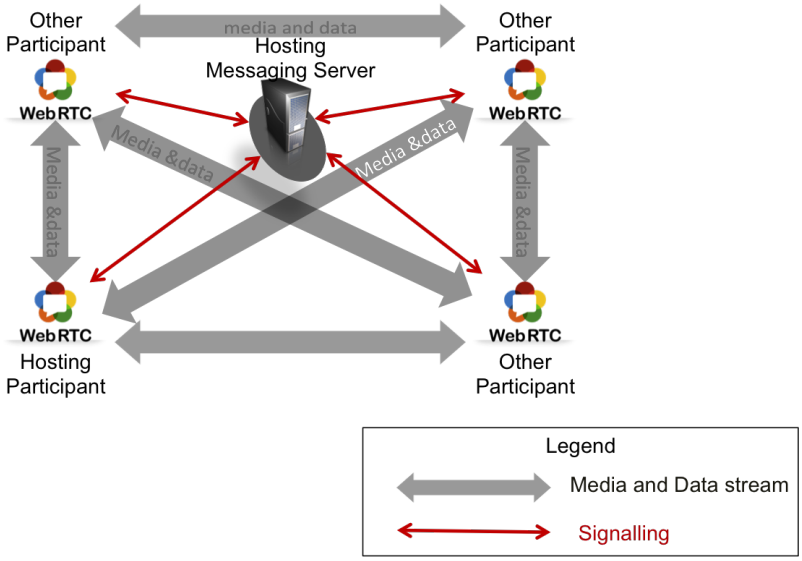

Figura 4.6: Sinalização *Multi-Party*

4.5 SUMÁRIO

capítulo 5

Cenários de teste e resultados obtidos

capítulo 6

Conclusões e trabalho futuro

Referências

[1] W3C, *A little history of the world wide web.* [Online; acedido Fevereiro-Março 2013], 2012. endereço: http://www.w3.org/History.html.

[2] *An introduction to the stream control transmission protocol (sctp)*, http://tools.ietf.org/html/rfc3286, May 2002. endereço: http://tools.ietf.org/html/rfc3286.

[3] *Stream control transmission protocol*, http://www.ietf.org/rfc/rfc2960.txt, October 2000. endereço: http://www.ietf.org/rfc/rfc2960.txt.

[4] *Datagram transport layer security*, https://tools.ietf.org/html/rfc4347, April 2006. endereço: https://tools.ietf.org/html/rfc4347.

[5] *User datagram protocol*, https://tools.ietf.org/html/rfc768, 28 August 1980. endereço: https://tools.ietf.org/html/rfc768.

[6] *Rtp: a transport protocol for real-time applications*, http://tools.ietf.org/html/rfc3550, July 2003. endereço: http://tools.ietf.org/html/rfc3550.

[7] *Datagram transport layer security (dtls) extension to establish keys for the secure real-time transport protocol (srtp)*, http://tools.ietf.org/html/rfc5764, May 2010. endereço: http://tools.ietf.org/html/rfc5764.

[8] *Sdp: session description protocol*, https://tools.ietf.org/html/rfc4566, July 2006. endereço: https://tools.ietf.org/html/rfc4566.

[9] I. E. T. F. (IETF), *Interactive connectivity establishment (ice)*, https://tools.ietf.org/html/rfc5245, [Online; acedido Janeiro/Fevereiro/Março 2014], April 2010. endereço: https://tools.ietf.org/html/rfc5245.

[10] *Traversal using relays around nat (turn): relay extensions to session traversal utilities for nat (stun)*, http://tools.ietf.org/html/rfc5766, April 2010. endereço: http://tools.ietf.org/html/rfc5766.

[11] Google, *Webrtc.* http://www.webrtc.org/, [Online; acedido Fevereiro-Março 2014], 2012. endereço: http://www.webrtc.org/.

[12] *Webrtc 1.0: real-time communication between browsers*, http://dev.w3.org/2011/webrtc/editor/webrtc.html, 10 April 2014. endereço: http://dev.w3.org/2011/webrtc/editor/webrtc.html.

[13] *Web-based real-time communication*, https://code.google.com/p/webrtc/source/checkout. endereço: https://code.google.com/p/webrtc/source/checkout.

[14] A. Johnston e D. Burnett, *Webrtc: APIs and Rtcweb Protocols of the Html5 Real-Time Web*, 1ª ed. USA: Digital Codex LLC, 2012, pp. 1–170, ISBN: 0985978805, 9780985978808.

[15] *Getting started with webrtc*, http://www.html5rocks.com/en/tutorials/webrtc/basics/, 23 July 2012. endereço: http://www.html5rocks.com/en/tutorials/webrtc/basics/.

[16] *Browser statistics*, http://www.w3schools.com/browsers/browsers_stats.asp. endereço: http://www.w3schools.com/browsers/browsers_stats.asp.

[17] *The websocket protocol*, http://tools.ietf.org/html/rfc6455, December 2011. endereço: http://tools.ietf.org/html/rfc6455.

[18] *The websocket protocol as a transport for the session initiation protocol (sip)*, http://tools.ietf.org/html/draft-ietf-sipcore-sip-websocket-10, November 2013. endereço: http://tools.ietf.org/html/draft-ietf-sipcore-sip-websocket-10.

[19] A. B. Johnston, *SIP: Understanding the Session Initiation Protocol*, 2ª ed. USA: Artech House: Norwood, MA, 2003, pp. 1–90, ISBN: 0985978805, 9780985978808.

[20] *Sip: session initiation protocol*, http://www.ietf.org/rfc/rfc3261.txt, June 2002. endereço: http://www.ietf.org/rfc/rfc3261.txt.

[21] M.Poikselkä e G.Mayer, *The IMS:IP Multimedia Concepts and Services*, 3ª ed. Wiley, 2009.

[22] A. S. Ahson e M. Ilyas, *IP Multimedia Subsystem (IMS) Handbook*, 3ª ed. CRC Press - Taylor & Francis, 2008.

[23] *Arquitetura ims.* http://www.hill2dot0.com/wiki/index.php?title=Image:IMS_Architecture_RefPts.jpg, Marth 2014. endereço: http://www.hill2dot0.com/wiki/index.php?title=Image:IMS_Architecture_RefPts.jpg.

[24] *Bluezy*, http://commons.wikimedia.org/wiki/File:Ims_overview.png, 28 May 2005. endereço: http://commons.wikimedia.org/wiki/File:Ims_overview.png.

[25] *Linphone-open source video sip phone for desktop/mobile.* http://www.linphone.org/, 2010. endereço: http://www.linphone.org/.

[26] *Linphone features*, http://www.linphone.org/eng/features/, 2010. endereço: http://www.linphone.org/eng/features/.

[27] *Jitsi features*, https://jitsi.org/Main/Features, January 2014. endereço: https://jitsi.org/Main/Features.

[28] *Jitsi - open source video calls and chat*, https://jitsi.org/, January 2014. endereço: https://jitsi.org/.

[29] Digium, *Asterisk*, http://www.asterisk.org/, [Online; acedido Janeiro/Fevereiro/Março 2014], 2014. endereço: http://www.asterisk.org/.

[30] Doubango, *World's first html5 sip client.* http://sipml5.org/, [Online; acedido Fevereiro-Março 2014], Marth 2014. endereço: http://sipml5.org/.

[31] Dougango, *Doubango telecom. smart sip and media gateway to connect webrtc endpoints.* http://www.webrtc2sip.org/, [Online; acedido Fevereiro-Março 2014], 2014. endereço: http://www.webrtc2sip.org/.

[32] *Identity provider selection*, http://dev.w3.org/2011/webrtc/editor/webrtc.html/identity, November 2012. endereço: http://dev.w3.org/2011/webrtc/editor/webrtc.html#identity.