Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

Luis Oliveira

Interoperabilidade WebRTC com Redes Legadas IMS e PSTN

Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

Luis Oliveira

Interoperabilidade WebRTC com Redes Legadas IMS e PSTN

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Rui Aguiar, Professor Associado C/ Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho aos meus pais, aos meus tios e à minha namorada por me terem apoiado de forma incansável.

o júri

presidente

Prof. Doutor Armando Humberto Moreira Nolasco Pinto

Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais

Prof. Doutor Rui Luís Andrade Aguiar

Professor Associado C/ Agregação, do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (Orientador)

Doutor Filipe Marques da Silva Cabral Pinto

Consultor Sénior na PT Inovação e Sistemas (Arguente Principal)

agradecimentos

Quero agradecer aos meus pais, por me terem proporcionado a oportunidade de frequentar um curso superior e pelo apoio que me deram ao longo deste percurso académico.

Agradeço também à Joana e aos meus tios por me terem ajudado ao longo destes anos, pelo carinho, pelos conselhos dados e apoio nas situações difíceis durante todo o meu percurso académico.

Quero também agradecer aos meus colegas Ilan Pegoraro e Vasco Amaral por toda a compreensão ao longo deste trabalho e pela a ajuda que me deram de forma incansável a todos os níveis.

Agradeço também ao Eng. Paulo Chainho pela oportunidade dada na realização deste projeto, pelo acompanhamento e ajuda ao longo do mesmo.

Por último, mas não menos importante, quero agradecer ao professor Rui Aguiar pela orientação, coordenação e dedicação prestada durante o trabalho

Palavras Chave

WebRTC, *IMS*, *SIP*, Redes Legadas, Interoperabilidade, *Wonder*, Sinalização *on-the-fly*, *MessagingStub*, Testes, Qualidade de Serviço.

Resumo

As redes de telecomunicações têm vindo a sofrer constantes evoluções desde a sua criação. Em 2012 o surgimento da tecnologia *WebRTC* veio revolucionar as telecomunicações. *WebRTC* é uma tecnologia que permite comunicações de áudio e vídeo entre *browsers* sem necessitar da instalação de qualquer *plugin*. O presente trabalho propõe-se a estudar, criar e testar uma solução que permita interoperabilidade da tecnologia *WebRTC* com as redes legadas. Esta dissertação foi desenvolvida no âmbito de um projeto Europeu chamado *Wonder*.

O principal objetivo deste projeto é construir uma *framework* que facilite o desenvolvimento de aplicações *WebRTC*, dando um enfoque especial a interoperabilidade entre os diferentes domínios de aplicações. Neste documento é descrito todo o funcionamento da *API Wonder* e apresentadas as suas principais componentes sendo apresentado a componente principal que irá permitir a interoperabilade com as redes legadas (*IMS e PSTN*). São ainda descritos todos os testes realizados de forma a comprovar o correto funcionamento da solução, bem como as componentes utilizadas nos testes.

Para além dos testes funcionais foram ainda realizados testes de qualidade, para identificar a qualidade de áudio/vídeo na transmissão. Os resultados dos testes realizados demonstraram total interoperabilidade entre os diferentes domínios estudados e as redes *IMS* e *PSTN*, mas quanto a qualidade de áudio/vídeo notou-se alguns problemas devido às interferências da rede.

O resultado final deste trabalho é a construção e demonstração de toda uma *API* que permita a intercomunicação entre os diferentes domínios e as redes legadas.

Keywords

WebRTC, IMS, SIP, Legacy Network, Interoperability, Wonder, Signaling onthefly, MessagingStub, Tests, Quality of Service.

Abstract

The telecommunication networks have been growing since their creation. In 2012, the WebRTC technology appeared and revolutionized the telecommunications environment. This is a technology that allows audio and video communications between browsers without installing additional plugins. With this project is proposed to study the WebRTC technology, and create a solution that, with this technology, that can be able to interoperate with legacy networks. This work was developed within a European project called Wonder.

The main goal of this project is to build a framework which makes the development of WebRTC applications easier, with a view of making the interoperability, between different domains, possible. In this documento is described all the Wonder functionalities and is presented all their components and the main component that allows the interoperability between legacy networks. It is described the tests that were made to prove that the solution and all the components are working correctly.

Beyond these tests, were done tests to identify the quality of audio and video transmission. The results of this tests prove, that the solution is totally functional and allows to have interoperability between different domains, in the quality tests was noticed that exist some problems because of network interferences.

The final result of this work, is to build an API that allows interoperability between different domains and legacy networks.

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

API	Application Programming Interface
CFSCs	Call Session Control Function
HSS	Home Subscriber Server
BGCF	Breakout Gateway Control Function
MGCF	Media Gateway Controller Function
AS	Application Service
MRFC	Media Resource Function Controller
MRFP	Media Resource Function Processor
SEG	Security Gateway
SBC	Session border controller
IBCF	Interconnection Border Control Function
UE	User Equipment
UA	User Agent
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

As telecomunicações têm vindo a sofrer constantes evoluções, sendo que a sua maior evolução fez se sentir com o surgimento da eletricidade. Antigamente as comunicações eram feitas através de sinais de fumo, ou de som de tambores. Em 1800 [1] com o surgimento/aparecimento da eletricidade, as comunicações começaram a ser feitas de forma diferente, melhorando a velocidade das comunicações. Desta forma, todo o ponto de partida para as comunicações modernas tem de se referenciar ao aparecimento da eletricidade. Desde então, começaram a surgir os mais diversos meios de comunicação, desde o aparecimento de um sistema de telegrafia, até ao aparecimento do código morse, que ainda hoje é usado.

Com esta evolução, os meios de comunicação foram melhorando, tornando-se possível converter sinais elétricos em sinais sonoros, surgindo posteriormente o telégrafo e os telefones. Estes meios de comunicação necessitavam de uma rede de comunicação distribuída e fiável, no entanto os problemas de *routing* e encaminhamento de chamadas foram surgindo, sendo mais tarde solucionados por operadores humanos e por circuitos comutados, surgindo a *Public Switched Telephone Network (PSTN)*.

Com o começo da era informática todo o mundo das telecomunicações foi influenciado pelo mesmo, e o desenvolvimento da informática revolucionou o mundo das telecomunicações bem como os seus requisitos de desempenho. Em 1938 [2], surgiu uma tecnologia inovadora chamada *Pulse Code Modulation (PCM)*, que permitia a transmissão digital de um sinal de voz, incluindo a sua codificação e descodificação.

A *World Wide Web Consortium (W3C)*, ou simplesmente *web* é um meio de comunicação global que começou a ser pensado em 1945, sendo nesse ano apresentado o sistema *Memex*, em que as funcionalidades deste sistema era permitir seguir *links* entre documentos, em microfilme. Em 1960 foi apresentado um sistema semelhante em que as ligações eram feitas entre documentos de texto, com o nome de *hypertext* [3].

Todo o processo de criação da *web* que hoje em dias dispomos foi evoluindo ao longo dos anos e continua a evoluir nos mais diversos dispositivos desde computadores, a *tablets* ou *smartphones*. Recentemente a *W3C* [3] em conjunto com a *The Internet Engineering Task Force (IETF)* desenvolveram a tecnologia *Web Real Time Communications (WebRTC)*. Esta tecnologia permite, a comunicação em tempo real entre *browsers*, sem requerer a instalação de qualquer *plug-in*. Surgiu recentemente, mais propriamente em 2012, sendo apresentada pela *Google*. Desde aí têm surgido várias soluções utilizando está *Application Programming Interfaces (APIs)*.

Esta tecnologia é uma grande evolução para as telecomunicações, pois irá permitir a criação de novos meios de comunicação. Foi feita uma estimativa da evolução da tecnologia nos próximos anos, como se pode verificar na figura 1.1 [4].

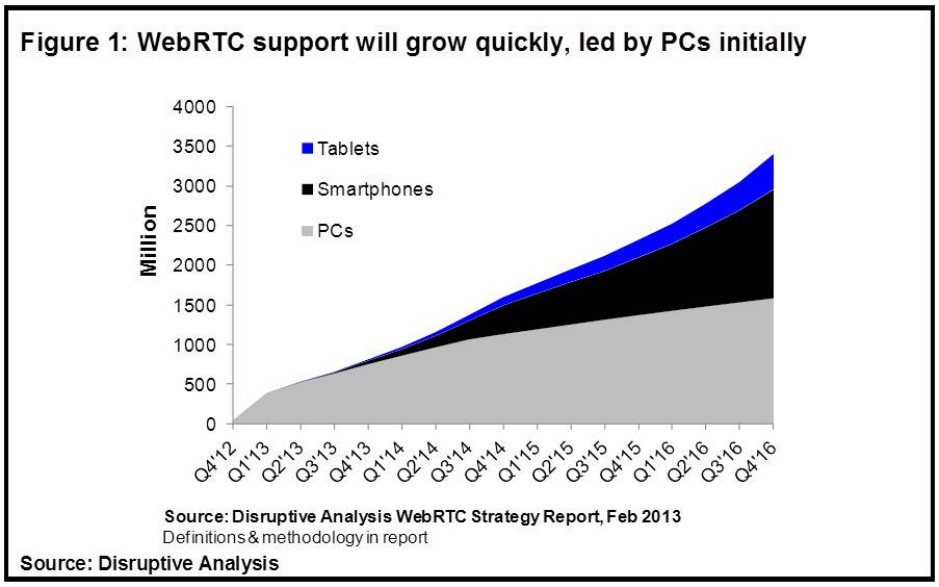


Figura 1.1: Estimativa de evolução da tecnologia *WebRTC* [4]

1.1 MOTIVAÇÃO E OBJECTIVOS

Esta dissertação surgiu no âmbito de um projeto Europeu chamado *Wonder* [5]. Este projeto tem como principal objetivo construir uma *API* que facilite o desenvolvimento de aplicações *WebRTC*, com um enfoque especial na interoperabilidade entre diferentes domínios, tais como domínios *IMS, SIP, Vertx, Node.js*. Este projeto foi desenvolvido num ambiente empresarial entre duas empresas, a PT Inovação e Sistemas e a *Deutsche Telekom*.

Nesta dissertação foi dado mais enfoque ao domínio *SIP*, que permite a interoperabilidade entre soluções *WebRTC* com redes legadas, *IMS* e *PSTN*. Na figura 1.2 é ilustrado uma das principais motivações para este projeto, que advém do facto da tecnologia *WebRTC* não normalizar um protocolo de sinalização. Desta forma é deixado para quem desenvolve a aplicação a escolha do protocolo de sinalização a usar. Torna-se usualmente impossível que um utilizador registado num ambiente totalmente *web* comunique com um utilizador registado num ambiente *SIP/IMS*. Esta dissertação tem assim como objetivo criar mecanismos que possibilitem a intercomunicação entre os diferentes domínios (*IMS, SIP, Vertx, Node.js*). Para isso foi utilizado um conceito que irá permitir ao utilizador comunicar com utilizadores de domínios diferentes: é a sinalização *on-the-fly*. Este conceito permite que os utilizadores registados em domínios diferentes comuniquem entre si, ou seja, um utilizador registado num domínio irá ser capaz de fazer o *download* e instanciar todos os métodos capazes de comunicar com um domínio diferentes do seu.

Existem, atualmente várias aplicações *Session Initiation Protocol (SIP)* no mercado, suportando vários tipos de serviços, desde *Voice over IP (VoIP)* a chamadas de vídeo até a troca de mensagens instantâneas. No entanto com o surgimento do *WebRTC* a comunicação entre os diferentes ambientes ficou limitada, devido a comunicação ser feita de diferentes maneiras.

No âmbito deste projeto foi necessário um estudo aprofundado da tecnologia *WebRTC*, e das tecnologias usadas para troca de sinalização entre *peers*, bem como de *gateways* que permitissem a interoperabilidade entre o *WebRTC* e os utilizadores *SIP/IMS*.

O principal objetivo desta dissertação é assim estudar, desenvolver e eventualmente adaptar algumas soluções existentes que permitam a interligação entre os diferentes domínios em estudo. Esta solução deverá garantir interoperabilidade entre os diferentes dominíos(*IMS, SIP, Vertx, Node.js*) e as redes legadas. Para isso irá fazer-se uma análise das diferentes plataformas *SIP* e *SIP/IMS* existentes (*stacks* e clientes) para que se possa fazer uma análise comparativa entre ambos, de forma ver qual fornece o melhor conjunto de serviços, para que seja possível cumprir os objetivos.

Para isso foram definidas algumas fases de desenvolvimento do projeto:

- Estudo da tecnologia *WebRTC* e das soluções existentes;
- Experimentação de diversas bibliotecas *SIP* bem como a sua interação com a tecnologia *WebRTC*;
- Instalação e configuração do ambiente para desenvolvimento;
- Desenvolvimento da prova de conceito;
- Teste e análise dos resultados.

Figura 1.2: Motivação para este projeto

A prova de conceito terá como principal objectivo demonstrar a interoperabilidade entre os diferentes domínios, usando a *API* do projeto (*Wonder*). Para isso foi desenvolvido uma aplicação que permita chamadas dos diferentes domínios (*web, Node.js, SIP, IMS*), para redes legadas *IMS* e *PSTN*.

No final desta dissertação são apresentados alguns testes, para verificar toda a interoperabilidade entre os diferentes domínios com as redes legadas e também verificar a qualidade de áudio e vídeo.

1.2 PRINCIPAIS CONTRIBUIÇÕES

A tecnologia *WebRTC* permite comunicações em tempo real entre *browsers*, sem requerer a instalação de *plugins*. Esta tecnologia não define um protocolo de sinalização padrão, deixando para o programador a sua escolha. Desta forma torna-se possível explorar a comunicação entre diferentes domínios, desde que exista um mecanismo que seja capaz de efetuar a sua intercomunicação.

As principais contribuições desta dissertação consistem na criação de uma base de estudo e de uma clarificação dos principais conceitos inerentes à tecnologia. De um ponto de vista prático foi

construída uma *API open source* [5] que poderá ser usada por todos os programadores. Para testes foi desenvolvida uma aplicação de forma a verificar que esta se encontra funcional.

Como fruto do desenvolvimento deste projeto/dissertação foram realizadas diversas apresentações, algumas em território nacional mais propriamente no *codebits* [6], e outras a nível internacional.

Num futuro próximo espera-se que a nível empresarial esta solução seja adoptada em projetos já existentes, e em novos projetos que surjam que usem a tecnologia *WebRTC*.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

Esta dissertação está dividida em seis capítulos fundamentais. No primeiro capítulo é apresentado uma introdução à problemática, sendo apresentado o projeto em que se insere esta dissertação.

No capítulo 2 relativo ao estado de arte, começam por ser apresentados os protocolos *DataChannel* usados na tecnologia *WebRTC*; em seguida é apresentado o problema de *NAT* existente na tecnologia, passando-se de seguida a explicar toda a tecnologia *WebRTC* e as suas componentes. Posteriormente são apresentados alguns tipos de sinalização que esta tecnologia poderá usar, onde será também apresentado o protocolo *SIP* e a arquitectura *IMS*.

No capítulo 3 são apresentadas diversas soluções relevantes existentes, quer a nível aplicacional, quer a nível de bibliotecas. São apresentadas diversas *stacks SIP/IMS* e algumas soluções *WebRTC* que existem, sendo por fim apresentadas as escolhas das diversas componentes utilizadas no projeto.

No capítulo 4 é apresentado o projeto em que está envolvida esta dissertação, bem como os seus principais conceitos e o seu funcionamento. É ilustrada também a forma como foi construída a solução que permitisse a interoperabilidade entre redes legadas.

No capítulo 5 começa-se por apresentar as diversas componentes instaladas para realização dos testes, sendo indicados os testes realizados e os seus resultados. No final é feita uma apreciação dos mesmos.

No capítulo 6 são tiradas as conclusões sobre o trabalho realizado nesta dissertação bem como o possível trabalho futuro a fazer.

CAPÍTULO 2

Estado de arte

O presente capítulo aborda os principais conceitos desta dissertação bem como o seu estado de arte no momento atual. Primeiro é explicado o que é o *WebRTC* e as suas *Application Programming Interfaces (APIs)*, bem como as diferentes formas de sinalização que poderão ser usadas. Em seguida serão descritos os protocolos usados para trocar dados através de *DataChannel*. De seguida são apresentados alguns dos problemas referentes às comunicações *WebRTC*, entre eles o problema de *Network Address Translation (NAT)*. Também é explicado o protocolo *SIP* que é usado na sinalização das comunicações *VoIP*, e por fim, é apresentada toda a estrutura e componentes de uma arquitetura *IMS*.

2.1 WEBRTC

As *Real Time Communication(RTC)* têm vindo a sofrer constantes evoluções devido a dois organismos de normalização, *Internet Engineering Task Force (IETF) e W3C*. Mais recentemente fruto dessa evolução é o surgimento da tecnologia *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 permitir desenvolver aplicações em *Web browsers*, com ajuda de *APIs* de *JavaScript* e *HyperText Markup Language 5 (HTML5)*. Este projeto encontra-se suportado pelos *Web browsers Chrome, Firefox* e *Opera* [7].

WebRTC tem por objetivo suportar o desenvolvimento de aplicações *web* que corram em qualquer dispositivo, desde *desktop* a dispositivos móveis. A partir desta tecnologia é possível criar aplicações, que corram em *web browsers*, usando as suas APIs de javascript e linguagem *HTML5*. Estas aplicações permitem ainda o acesso seguro às componentes dos sistemas tais como *(*webcam), microfone [8].

Na Figura 2.1 é possível ver a arquitetura da tecnologia *WebRTC*, onde se pode observar que é constituída por duas *APIs*: *WebRTC C++* e a *Web API*. A *WebRTC C++* é utilizada por programadores dos *web browsers* enquanto que a *Web API* é usada por programadores que queiram desenvolver aplicações *web* usando a tecnologia *WebRTC*.

Na figura 2.1, é possível ainda verificar outras componentes que são necessárias para a criação de aplicações *web* que usem a tecnologia *WebRTC*. Essas componentes são os elementos responsáveis pela

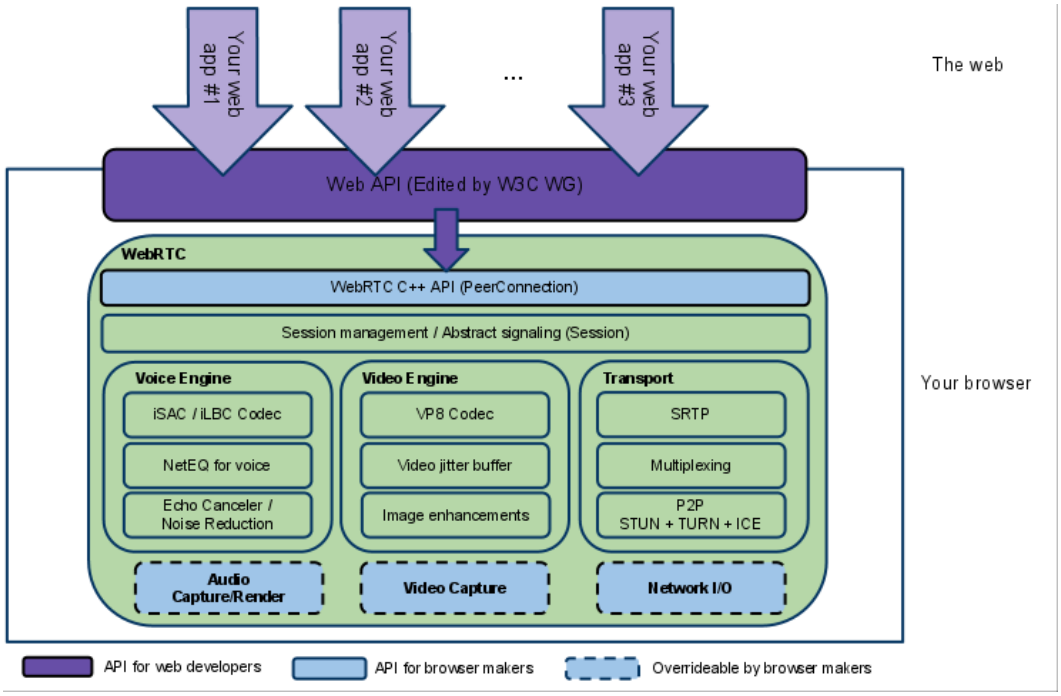


Figura 2.1: Arquitetura do *WebRTC* [9]

captura e transmissão do áudio/vídeo; naturalmente também existirá uma componente responsável pelo seu transporte.

Esta tecnologia atualmente encontra-se implementada pelo *Chrome (Desktop e Android)*, *Firefox* e *Opera (Desktop e Android)* ainda não sendo suportado pelo *IE(Microsoft)* nem pelo *Safari (Apple)* [8] (apesar de existirem *plugins* para isso).

Como referido anteriormente a tecnologia *WebRTC* é uma tecnologia *open source* cuja documentação e código fonte poderão ser encontrados no *SVN* da *Google* [10]. Como o trabalho desenvolvido incidirá sobre o desenvolvimento de uma aplicação *web* irá ser dado mais ênfase à *Web API* nesta dissertação.

De seguida irá ser explicada esta tecnologia bem como o seu estado atual. Serão também referidos alguns protocolos de sinalização que poderão ser usados para obter comunicação entre dois *peers*. Para desenvolvimento da solução vai ser usado como protocolo de sinalização o protocolo *Sip Over Websockets*.

Nos subcapítulos seguintes, será explicado o protocolo *SIP* bem como as suas componentes e as mensagens utilizadas para o estabelecimento de uma sessão *RTP*.

2.1.1 WEBRTC API

A *WebRTC API* permite o desenvolvimento de aplicações *web*. Assim para facilitar o desenvolvimento da aplicação final foi adoptado pela linguagem de programação *JavaScript*. Esta *API* é composta por três elementos fundamentais: *GetUserMedia, PeerConnection* e *DataChannel* [11] [8].

GetUserMedia é uma função do *WebRTC* que permite o acesso às *streams* de áudio/vídeo, as *streams* podem ser locais *(LocalMediaStream)*, ou remotas *(RemoteMediaStream)*. As *LocalMediaStream* representam a *stream* capturada pelo próprio sistema (câmara/microfone); as *RemoteMediaStream* representam as *streams* recebidas de outro participante. A função *GetUserMedia* permite a definição

de alguns parâmetros de entrada, tais como as *constraints* que queremos ter acesso. Desta forma é possível definir se queremos ter acesso a câmara/microfone ou só ao microfone. Para a transmissão das *mediastreams* a tecnologia faz uso dos protocolos de *SRTP*, *RTCP* e *RTP*. O protocolo *DTLS* é usado para gerar uma chave para a sessão *SRTP* [11] [8].

Desta forma as principais características da função *GetUserMedia* são:

- Acesso aos recursos *media* do utilizador, nomeadamente câmara 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 é um canal de comunicação bidirecional usado para a comunicação entre dois *web browsers*. Para existir uma *PeerConnection* entre dois *browsers* é necessário haver uma sinalização entre ambos. Esta negociação é feita através do uso do protocolo *ICE* e do uso de servidores de *STUN* e *TURN*, que irão permitir a passagem das *mediaStreams* por *NAT* [11] [8].

As principais características das *PeerConnection* são:

- Estabelecer sessões de comunicação entre *browsers*.
- *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 é um canal de dados bidirecional. A tecnologia *WebRTC* permite para além da transmissão de *media* a troca de dados, para isso faz uso dos protocolos *SCTP* encapsulado em *DTLS*. Através destes protocolos é possível a comunicação entre dois *peers*, que estejam entre *NAT*, com garantia de confidencialidade e autenticação da ligação estabelecida. Assim é possível garantir a integridade dos dados transferidos. Para ser possível a comunicação de dados é necessário criar um *dataChannel*, que pode ser criado através da função do *WebRTC CreatDataChannel()*. Este canal de comunicação é estabelecido em cima de uma *PeerConnection* [11] [8].

As principais características do *dataChannel* são:

- 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 através do protocolo *SCTP*.

2.1.2 COMPATIBILIDADE

A *WebRTC* é uma tecnologia recente e que não depende unicamente de si, uma vez que necessita de interagir com os diferentes *browsers*. É então necessário garantir que os *web browsers* suportam esta tecnologia, que tem como principal dependência a linguagem *HTML5* que é o novo *standard* da linguagem *HTML*. Esta última evolução da linguagem veio introduzir melhorias quer a nível de tratamento de erros quer a nível de redução de *plugins* instalados no *web browser*.

Com recurso à tecnologia *WebRTC* (nomeadamente *API WebRTC*) e à linguagem *HTML5*, é possível criar aplicações que reproduzam áudio/vídeo em *tags HTML5* fornecidas pela *API WebRTC* e desta forma reproduzir as *streams* [9],[8]. Na tabela 2.1 é possível verificar as versões dos *webbrowsers* suportadas pelas *APIs WebRTC (MediaStream, Peer Connection, RTC DataChannels)*

Tabela 2.1: Versões dos *web browsers* que suportam as *APIs WebRTC* [9].

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 possível verificar quais os *web browsers* mais utilizados de forma a garantir a maior interoperabilidade entre eles, bem como a percentagem de *web browsers* usados nos meses de Janeiro, Fevereiro, Março e Abril de 2014.

Tabela 2.2: Percentagem dos *web browsers* usados [12].

2014	**Internet Explorer**	**Firefox**	**Chrome**	**Safari**	**Opera**
Janeiro	10,2%	26,9%	55,7%	3,9%	1,8%
Fevereiro	9,8%	26,4%	56,4%	4,0%	1,9%
Março	9,7 %	25,6%	57,5%	3,9%	1,8%
Abril	9,4 %	25,0%	58,4%	4,0%	1,8%

2.1.3 CODECS

O *WebRTC*, ao nível dos *codecs*, requer algum cuidado quanto à sua escolha uma vez que nem todos são suportados pelos diferentes *web browsers*, nem pelas *gateways* que irão fazer a ligação com as redes legadas. Visto isto, a escolha dos mecanismos de compressão de áudio e vídeo deve ser feito de uma forma rigorosa, para que se consiga obter os melhores resultados.

O *WebRTC* definiu vários requisitos no que diz respeito aos *codecs*. Todos os clientes que usem esta tecnologia devem garantir o uso de mecanismos de compressão de áudio *Pulse Code Modulation (PCMA/PCMU)*, *telefone event*, *Opus* e *G.711* como *codecs*, sendo estes dois últimos obrigatórios [13]. Quanto aos mecanismos de compressão de vídeo, as restrições são menos significativas, apenas devem ser garantidas 10 *frames* por segundo, e uma resolução mínima de 320x240, sendo também suportadas outras resoluções. Quanto aos *codecs* de vídeo não existem restrições ficando apenas dependentes da implementação do *web browser*. Na tabela 2.3 é possível ver os diferentes *web browsers* e os *codecs* por eles suportados [13].

Mais recentemente começou a ser desenvolvido um novo *codec*, o *VP9*, com código aberto. Este *codec* é uma evolução do *VP8*, tendo com principal objetivo proporcionar a mais alta qualidade ao utilizador e a capacidade de suportar a maior variedade de dispositivos de destino [14].

Tabela 2.3: *Codecs* suportados pelos diferentes *web browsers* [13].

	Internet Explorer	**Firefox**	**Chrome**	**Opera**
Vídeo	H.264 AVC	VP8	VP8	VP8
Áudio	Opus,G.711	Opus,G.711	Opus,G.711	Opus,G.711

2.1.4 SINALIZAÇÃO WEBRTC

A sinalização é a parte mais importante de uma aplicação *WebRTC*, em geral é a parte mais importante de qualquer aplicação que use comunicações em tempo real.

Um recurso que *WebRTC* não fornece é um canal de sinalização para estabelecimento de chamadas. A sua decisão é deixada para o programador, pois como se pode verificar na figura 2.2, esta componente não é especificada. A sinalização poderá ser feita por exemplo usando *WebSockets, IMS, XMPP*. Para possibilitar a comunicação entre redes legadas a sinalização usada na solução irá ser *Sip Over WebSockets*. De seguida irão ser apresentados alguns dos protocolos mais usados para sinalização na tecnologia *WebRTC* [9].

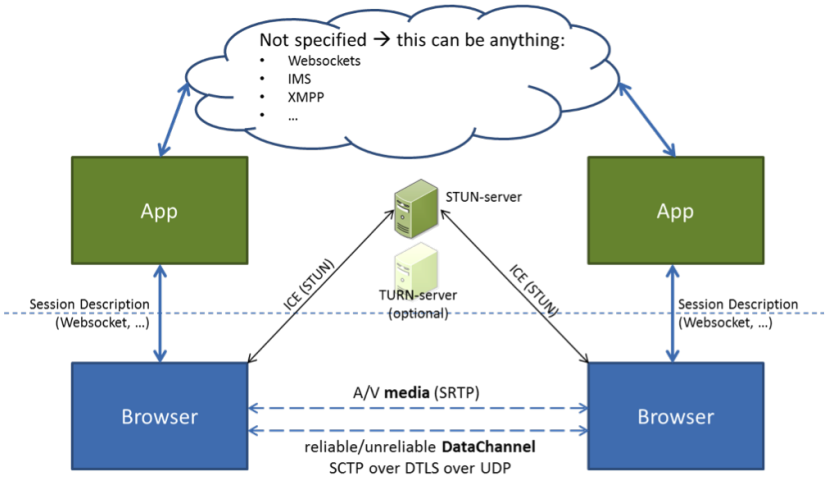


Figura 2.2: Sinalização Webrtc [9]

2.1.5 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 tem como principais problemas o abuso da comunicação *HTTP*, o que pode ter como principais problemas [15]:

- O servidor necessitar de ter 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 necessário ter um *script* para tratar das ligações ao servidor.

Com o aparecimento da tecnologia *AJAX*, as páginas tornaram-se mais dinâmicas. Porém sempre que a página fosse atualizada, teria de existir um temporizador para que só após toda a comunicação com o servidor terminar, possa ser mostrada a informação ao utilizador.

No protocolo *WebSockets*, é usada uma comunicação mais simples, em que é usado uma única conexão *TCP* em ambas as direções. O surgimento deste protocolo deve-se ao facto de substituir as comunicações bidirecionais que usam *HTTP*, sendo baseado em *sockets* e usa *HTTP* como camada de transporte. Desta forma é possível usar *WebSockets* em soluções já existentes uma vez que funcionam sobre as portas *HTTP* tais como 80 e 443, facilitando assim a sua integração[15].

2.1.6 JINGLE OVER WEBSOCKETS

Jingle over WebSockets é um protocolo que usa como base da sua comunicação o *Extensible Messaging and Presence Protocol (XMPP)*. O objetivo deste protocolo é permitir a comunicação *peer-to-peer* de *media* e dados entre utilizadores. Permite que os programadores facilmente criem aplicações que tenham a capacidade de comunicar entre diversos tipos de aplicações, tais como partilha de áudio/vídeo, *chat* e transferência de ficheiros [16].

Este protocolo implementa uma camada de segurança que garanta uma comunicação segura. Para isso precisam de ser cumpridos alguns requisitos, de forma que a comunicação entre utilizadores seja feita: deverá ser garantido o uso do *TLS* como protocolo de negociação da sessão, e para transporte de *media* deverá ser usado o *DTLS*.

Um dos problemas do uso deste protocolo para sinalização é que envolve demasiado *overhead* na comunicação, uma consequência do facto da comunicação ser baseada em *HTTP*.

Este protocolo tem um funcionamento semelhante ao *SIP over WebSockets*, pelo que toda a informação é enviada encapsulada dentro de um *websocket* [16].

2.1.7 SIP OVER WEBSOCKETS

O protocolo *Sip over WebSockets* é um sub protocolo do protocolo *WebSockets*, permitindo 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 responsáveis neste protocolo são [17]:

- *SIP WebSocket Client* - esta entidade tem 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*.

2.2 PROTOCOLOS DATA CHANNEL

A tecnologia *WebRTC* para além de permitir a comunicação de *media (RTP)* entre dois *peers*, permite a partilha de dados entre dois *peers*. Foi neste contexto que surgiu o conceito *data channels* na tecnologia *WebRTC*. A troca de dados entre dois *peers* é feita através de *peer connections*. Para ser efetuado, a comunicação de dados entre *peers* numa sessão são usados cinco protocolos de controlo e segurança da comunicação. Estes são [18] :

- *Stream Control Transmission Protocol (SCTP)* para controlo das *streams*;
- *Datagram Transport Layer Security (DTLS)* para segurança/criptografia dos dados transferidos;
- *User Datagram Protocol (UDP)* para controlo da camada de transporte e connectividade *peer-to-peer*;
- *Secure Real-Time Protocol - Datagram Transport Layer Security* que fornece multiplexação, fluxo e controle de congestionamento
- *Interactive Connectivity Establishment (ICE)* para controlo da camada de transporte.

Na figura 2.3, pode-se observar a pilha protocolar usada na tecnologia *WebRTC*, sendo a base de toda a comunicação a rede de *IP* de *Internet*, seguindo-se o protocolo *UDP* que é a base da comunicação em tempo real do *browser*, mas para que tal seja possível o *browser* precisa de outros protocolos associados, tais como, o *SRTP* e *SCTP* que estão encapsulados no *DTLS*. Através deste conjunto de protocolos, consegue-se ter comunicação *peer-to-peer (RTCPeerConnection)* e de canais de dados *DataChannel* [18].

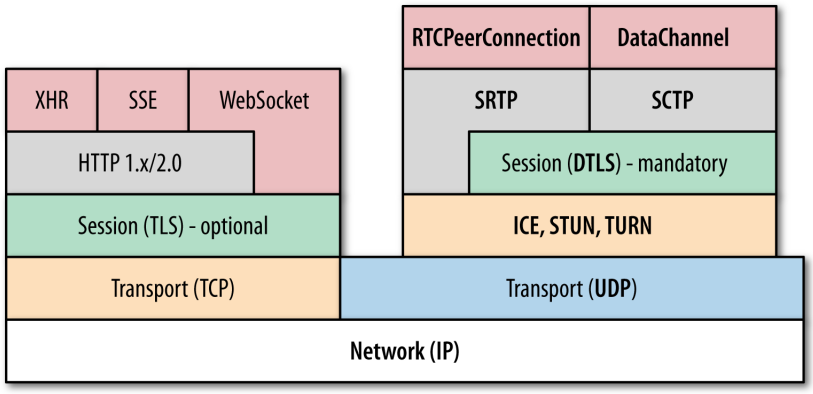


Figura 2.3: Pilha protocolar *WebRTC* [19]

Neste subcapítulo serão apresentados os protocolos usados para a comunicação usando *DataChannels*, sendo ainda apresentados os protocolos *User Datagram Protocol(UDP)*, o *Session description protocol(SDP)*. Por fim, é apresentado o protocolo que se usa na comunicação por *Datachannels* e na comunicação *WebRTC*, *Interactive Connectivity Establishment (ICE)*. O uso do protocolo *ICE* na tecnologia *WebRTC*, deve-se ao facto desta tecnologia ter problemas com *Network address translation(NAT)*. Na secção seguinte serão apresentados os problemas com *NAT* nesta tecnologia, bem como possíveis soluções para os mesmos.

2.2.1 STREAM CONTROL TRANSMISSION PROTOCOL(SCTP)

SCTP é um protocolo da camada de transporte, que tem como características, o fornecimento de serviços como 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)*. Este protocolo garante ainda entrega sequencial de mensagens do utilizador em múltiplas *streams*, podendo ainda garantir a sua ordenação. Existe ainda um mecanismo de tolerância a falhas a nível de rede, relativamente a entrega de pacotes[20] [21] [19].

SCTP é um protocolo de transporte, semelhante ao *TCP* e *UDP*. Na tabela 2.4, é feita uma comparação entre os protocolos. Pode-se tirar como conclusão desta análise que o *SCTP* oferece as melhores características do protocolo *UDP* e *TCP*.

Tabela 2.4: Comparação entre: *TCP* vs *UDP* vs *SCTP* [19].

	TCP	**UDP**	**SCTP**
Confiança	confiável	inseguro	configurável
Entrega	ordenada	desordenada	configurável
Transmissão	orientada ao byte	orientada a mensagem	orientada a mensagem
Controle de fluxo	sim	não	sim
Controle de congestionamento	sim	não	sim

2.2.2 DATAGRAM TRANSPORT LAYER SECURITY(DTLS)

O *DTLS* é um protocolo que garante a segurança no tráfego da rede, similar ao *"TLS over Datagram"*. Garante a segurança do tráfego *web* e do tráfego *e-mail*, sendo usado em protocolos como *IMAP* e *POP* [22]. O *DTLS* surgiu a partir do *TLS* sendo em parte a sua estrutura semelhante. 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. O uso do *DTLS* não tem qualquer implicação quer a nível de atraso dos pacotes quer a nível de perda de pacotes [22].

2.2.3 USER DATAGRAM PROTOCOL(UDP)

UDP é um protocolo da camada de transporte que permite a troca de mensagens com o mínimo *overhead*. Não possui qualquer mecanismo para garantir que os pacotes são entregues, ou seja, não há tolerância a falhas ou atrasos. O *UDP* é usado em aplicações que necessitem de transportar pequenos pacotes de dados e para trocas de *media*, ou seja, em sessões *RTP*. Este facto deve-se ao *UDP* garantir uma rápida entrega dos pacotes de *media*. Para controlar as falhas na comunicação é necessário criar uma série de estruturas de controlo, tais como: sistemas de redução de largura de banda, retransmissões, etc [23].

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

RTP é um protocolo de rede, utilizado em aplicações de tempo real, este protocolo permite a entrega de serviços fim a fim em tempo real como por exemplo, áudio/vídeo ou dados. O *RTP* é usado em diversas aplicações para, 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 [24].

O *SRTP* é um protocolo que surgiu para garantir a segurança na comunicação *RTP*. O *SRTP* garante a confidencialidade e a autenticação das mensagens. O *DTLS* surgiu como uma extensão para o *SRTP*, visto que o *SRTP* não permite uma gestão de chaves nativas, sendo necessário 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 [25].

2.2.5 SESSION DESCRIPTION PROTOCOL(SDP)

SDP é um protocolo usado para descrever a sessão entre dois participantes. Este protocolo descreve as características da sessão, como, detalhes do *media* de forma a identificar, quais os (*codecs*) usados naquela sessão. Ao iniciar uma chamada *Voice over IP (VoIP)* ou um *streaming* de vídeo são trocados diversos parâmetros de sessão como endereços, tamanho do vídeo e *codecs* usados. Esta negociação é feita usando o protocolo *SDP*, que descreve as características daquela sessão. A sua descrição 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 características 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 possível ao utilizador ligar-se à 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 início de uma sessão *WebRTC* é feita usando um mecanismo de *offer* e *answer*. 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 ao *offer* que contêm 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 de forma a que informação seja trocada entre eles[26].

Para a criação do *SDP* numa comunicação *WebRTC* é usado o protocolo *JavaScript Session Establishment Protocol (JSEP)*. Este define a forma como as aplicações escritas em javascript interagem com as funções *RTP*. Para o estabelecimento da uma sessão entre dois *peers* é necessário a troca de sinalização por meio de uma *offer/answer*, em que são enviadas as informações sobre as características para aquela sessão, esta informação é enviada no objecto *RTCSessionDEscription*. O protocolo *JSEP* é responsável pela forma como os clientes recolhem a informação sobre o *media*, como por exemplo, qual o *media* que suportam, os *codecs* que suportam [27].

2.2.6 INTERACTIVE CONNECTIVITY ESTABLISHMENT (ICE)

ICE é uma técnica usada na tecnologia *WebRTC*, envolvendo tradutores de endereços *NAT*. *ICE* tem como objetivo permitir a troca de *media* entre dois utilizadores em que exista *NAT* entre ambos. É um protocolo sobretudo usado em aplicações que trocam *media* entre si, os conteúdos trocados entre si têm incluídos os *IP* e portos associados a comunicação[28].

Nem sempre a técnica *ICE* é fiável e poderá falhar, nesse tipo de casos o protocolo *ICE* tira proveito das funcionalidades do protocolo *TURN* [28].

2.3 NETWORK ADDRESS TRANSLATION(NAT)

Um dos problemas com que a tecnologia *WebRTC* se depara é com problemas de *NAT*, isto porque os endereços *IP* dos pacotes com origem em redes internas não irão conseguir aceder à 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. Em conjunto com estes protocolos, é usado o protocolo anteriormente mencionado (*ICE*). Neste subcapítulo serão apresentados estes protocolos bem como o seu funcionamento de forma a possibilitar a comunicação entre *peers* usando a comunicação *WebRTC*.

2.3.1 SESSION TRAVERSAL UTILITIES FOR NAT (STUN)

Este protocolo permite a descoberta do *IP* e do porto em que a comunicação está a ser feita. As principais características deste protocolo é que permite verificar a conectividade 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 possível no início da comunicação são enviados pacotes de teste de forma a descobrir o caminho que se encontra até ao outro utilizador e a partir deste mecanismo é descoberto os *IP's* e portos mapeados [29].

2.3.2 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 necessário recorrer a um *relay* que se encontra na rede pública da *internet*, que têm como função retransmitir os pacotes entre os dois utilizadores. A tecnologia *WebRTC* usa este protocolo de forma a permitir a comunicação entre dois utilizadores. Assim 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 [30].

2.4 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* [31] [32].

Em seguida será explicado a arquitetura do *SIP*, bem como as principais mensagens trocadas. No subcapítulo seguinte irá ser apresentado a arquitetura *IMS* que usa o protocolo *SIP* como meio de comunicação para o estabelecimento de sessões entre os utilizadores.

2.4.1 ARQUITETURA

Arquitetura

Este protocolo tem uma arquitetura que é definida por várias entidades ou sistemas, que são importantes para o seu funcionamento [31] [32]:

- *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 domínio 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 redirecionamento (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*.
- *Domain Name System (DNS) Server*: este servidor tem como funcionalidade indicar qual o *IP* associado a determinado domínio. 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 domínio 2. Da mesma forma o *Proxy Server* 2 poderia identificar o Servidor *backup* para o *Proxy Server* 1, caso este fosse abaixo.

2.4.2 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 [32].

Request Messages

- *INVITE*: método para estabelecer uma sessão de *media* entre dois *User Agents*. Este tipo de mensagem pode conter informação sobre quem faz o *INVITE* e 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*.

Além destas mensagens principais temos ainda muitas outras:

- *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 é usado 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 [32]:

- *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 IP MULTIMEDIA SUBSYSTEM (IMS)

O *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, mas 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 às redes de telecomunicações, sendo necessário algumas alterações nas infra-estruturas organizacionais para poder usufruir das suas vantagens.

2.5.1 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. Estas interfaces definem pontos de referência, que determinam 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 2.4 [33] [34]:

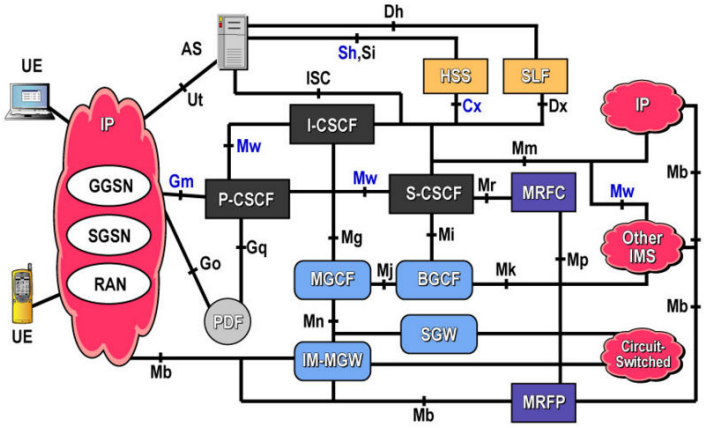


Figura 2.4: Arquitetura geral IMS e pontos de referência [35]

- 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*;
- Elementos de 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 *media*, *Signalling Gateway* (**SGW**);

- Serviços - *Application Service* (**AS** - servidor que fornece serviços aos utilziadores), *Media Resource Function Controller* (**MRFC** - controlador de *media*), *Media Resource Function Processor* (**MRFP** - elementos de *media*);
- Entidades de suporte - *Security Gateway* (**SEG** - *gateway*), *Interconnection Border Control Function* (**IBCF** - controlador de saída do domínio);

Estas categorias podem ser estruturadas por camadas principais: *Transporte Plane*; *Control Plane ou IMS Layer*; *Service or Application Plane* como é possível ver na figura 2.5.

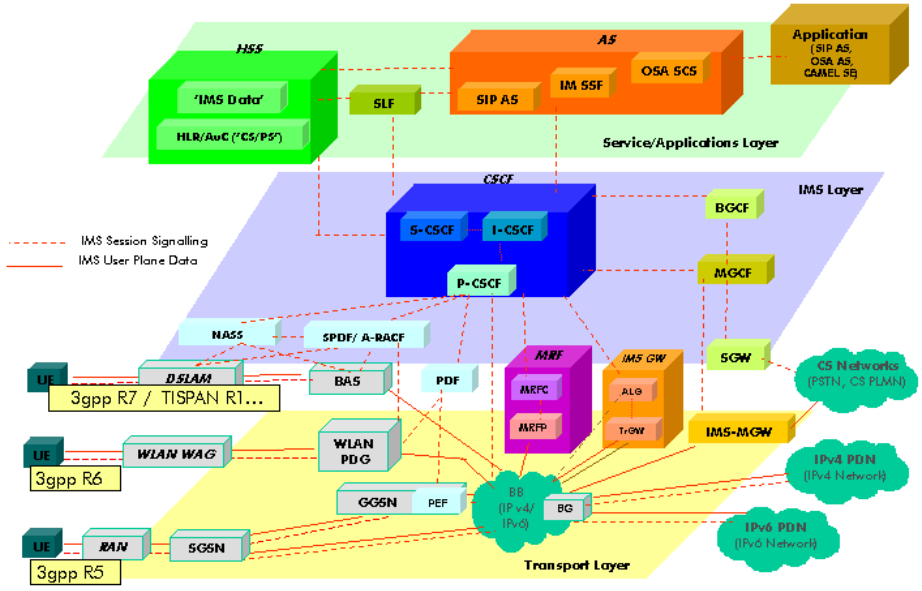


Figura 2.5: Arquitetura geral *IMS* [36]

Transport Plane

O *Transport Plane* contêm 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 [33] [34]:

- *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 [33];
- *Media Gateway (IMS-MGW)* - Interface entre *IMS media* e da *legacy network media (Public land mobile network (PLMN), PSTN)* [33].

Control Plane ou IMS Layer

Este plano é responsável por encaminhar o controlo 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 principalmente pelo *HSS* e *CSCF*, contendo também *BGCF* e *MGCF* [33], que a seguir se explica [34]:

- *Proxy-CSCF (P-CSCF)* - é um *SIP-Proxy* é o primeiro ponto de contacto com *UE*. Pode estar localizado tanto na *Home Network* como na *Visited Network*. Esta 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 primeiro 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 (S-CSCF)*: Está localizado na *home network* e é um dos pontos mais cruciais da arquitetura 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 contêm 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 contêm os *Application Servers* que suportam *SIP* e *DIAMETER*. Ele 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 [33].

2.5.2 INTERLIGAÇÃO DE WEBRTC COM IMS E PSTN

WebRTC é um conjunto de *APIs* elaborada pela *W3C* que permite comunicações de tempo real entre *web browsers* a partir de *APIs JavaScript*. Esta tecnologia veio permitir uma grande inovação, ao nível das comunicações, permitindo que quer as empresas quer os programadores comecem a integrar ou a desenvolver novas aplicações com esta tecnologia.

Pelo que se torna necessário, adpatar/integrar este novo meio de comunicação com os existentes, sendo exemplo disso as redes legadas. A rede pública de telefonia comutada ou *PSTN* como é conhecida, é a designação usada para identificar a rede de telefonia comutada por circuitos, esta rede destina-se a serviços telefónicos. No seu início foi desenhada exclusivamente para para linhas analógicas, hoje em dia já suporta outros dispositivos, como os dispositivos móveis.

Com o surgimento das redes de nova geração, ou *Next Generation Networking(NGN)*, tornou-se necessário integrar a *PSTN* com as novas arquiteturas. Desta forma a integração da *PSTN* numa rede *IMS* é feita com recurso a algumas componentes da mesma.

Para ter comunicação entre redes legadas e o *WebRTC* é necessário garantir alguns requisitos nessa comunicação, tais como, o uso do *codec VP8*, *SRTP/SRTCP* para transporte de *media*, e estabelecimento de sessão com recurso ao protocolo *ICE* para negociação de endereços e portas.

Uma das formas de comunicar entre redes legadas e *WebRTC* seria com recurso a um *Session Border Controller (SBC)*, este deverá ser capaz de fazer todo o trabalho de intercomunicação entre os diferentes ambientes.

Na figura 2.6 é possível verificar um exemplo de uma possível arquitetura de rede que permitirá a interoperabilidade entre o ambiente *WebRTC* e as redes *IMS* e *PSTN*. Desta forma torna-se visível que um utilizador poderá registar-se numa rede *IMS* a partir de uma aplicação *WebRTC*, mas para isso necessitará de uma *gateway*, que irá fazer a intercomunicação entre os dois meios. A informação sobre o utilizador está registado num servidor *SIP (SIP AS*; no âmbito deste projeto será utilizado como servidor o *Asterisk*), que irá permitir comunicar com a *PSTN* e com a rede interna da empresa. Toda a comunicação entre o ambiente *WebRTC* e as redes *IMS* e *PSTN* irá ser efetuado a partir da *gateway*, inclusive o tráfego *media*.

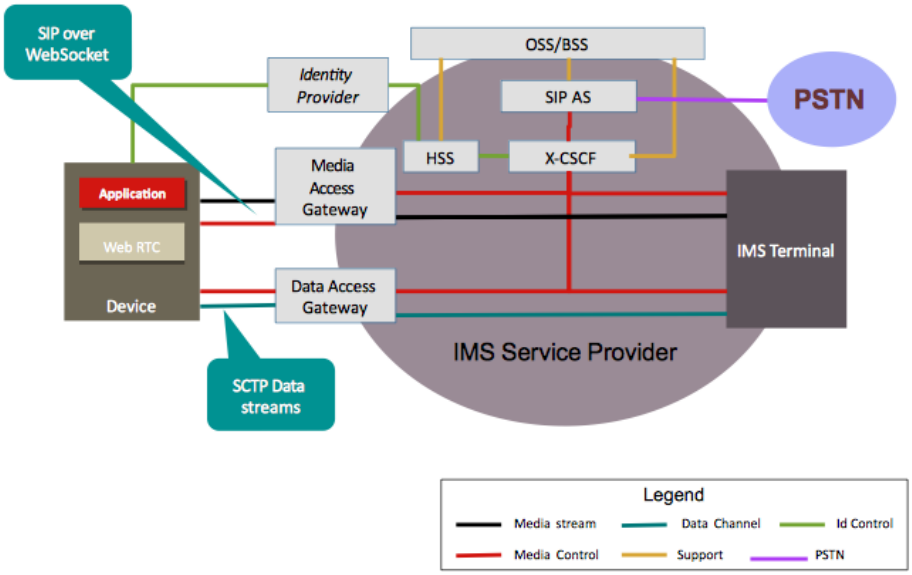


Figura 2.6: Interligação do *WebRTC* com *IMS* e *PSTN*

Um possível fluxo de interoperabilidade entre *WebRTC* e redes legadas seria:

1. A aplicação/cliente *WebRTC* liga se a uma *gateway SIP* e regista-se. O servidor *SIP (AS)*, deverá suportar *WebSockets*; caso contrário será necessário usar uma *gateway* que irá efetuar a tradução da sinalização e *media*, tal como o *webrtc2sip*;
2. O utilizador para comunicar com a *PSTN* envia um *SIP INVITE*, que irá convidar o utilizador da rede *PSTN*, este *SIP INVITE* irá encapsulado dentro de um *websocket*. Este *INVITE* contém a informação relativa para aquela sessão, tal como o *SDP* para a mesma, sendo que este terá de conter os *codecs* e os *icecandidates*;
3. O servidor *SIP* aloca uma sessão de *Media Gateway*, e envia uma resposta para o cliente remoto com o *SDP*;
4. Após os passos anteriores, a aplicação/cliente *WebRTC* tem toda a informação necessária para trocar *media* com a *Media Gateway*.
5. O servidor *SIP* serve também a sinalização a *PSTN* assim que a chamada é atendida, é estabelecido um canal *full duplex* entre os utilizadores.

2.6 SUMÁRIO

Neste capítulo foi apresentado de uma forma mais aprofundada a tecnologia *WebRTC*, bem como as componentes de que é constituída, sendo apresentada as principais *API (PeerConnection, Media Streams, RTC Data Channels)*. Sendo também referido a compatiblidade atual com os *web browsers*, sendo também apresentado os *codecs* que cada *web browser* usa.

Em seguida foram apresentados os protocolos usados na comunicação através de *Data Channel*, sendo eles o *SCTP* para controlo de *streams*, o *DTLS* para segurança da comunicação e o *UDP/ICE* que são definidos na camada do transporte. Foram também apresentados os problemas de *NAT* e *firewall* existentes na tecnologia *WebRTC* e a forma como são resolvidos com recurso aos protocolos de *STUN*, *TURN* e *ICE*.

Foi apresentado ainda o protocolo *SIP* bem como as suas componentes principais assim como as mensagens trocadas numa sessão, sendo de seguida apresentado a arquitetura de uma rede *IMS* e as suas características. No fim foi apresentado um esquema de uma arquitetura que permitiria a interoperabilidade entre *WebRTC* e as redes *IMS* e *PSTN*.

capítulo 3

Soluções existentes

Neste capítulo serão apresentadas algumas soluções existentes, começando por serem apresentadas aplicações *SIP/IMS*. Na secção seguinte é apresentado uma análise a algumas soluções *WebRTC*, mostrando a forma como as *APIs* podem ser usadas de diferentes modos. Em seguida são apresentadas 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 análise de cada componente são apresentadas as decisões que foram tomadas para realizar este projeto.

3.1 aplicações sip e sip/ims

Existem bastantes aplicações e serviços que têm de suportar a criação e a gestão de sessões entre utilizadores. Estas sessões lidam com vários tipos de tráfego, principalmente tráfego multimédia em tempo real, como dados de voz, vídeo ou mensagens de texto. O protocolo *SIP* trabalha nesse conceito [32], de conseguir descobrir quais as características de cada utilizador e o que tipo de sessão querem que seja estabelecida entre eles [32]. O conceito das redes de nova geração surgiu para atender às novas necessidades no sector das telecomunicações das operadoras. Para isso foi introduzida a arquitetura *IMS*, que tem por objetivo atender a essas necessidades, no controlo de sessões multimédia [35]. É relevante dar importância ao protocolo *SIP* e à arquitetura *IMS*, pois este projeto será baseado em ambos. Para isso analisaram-se vários clientes *SIP/IMS*. Esta análise centrou-se mais pormenorizadamente sobre os serviços que cada uma das aplicações suportava, os *codecs* que poderiam usar, tanto a nível de áudio como de vídeo.

3.1.1 linphone

É um cliente *SIP* que simula um telefone *VoIP*, *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* [37]. Em termos de segurança usa encriptação na parte de voz e de vídeo com a ajuda dos protocolos *SRTP* e *zRTP* [38]. 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, recorrendo 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* [38].

3.1.2 IMSDROID 2.X

É uma aplicação/cliente *SIP/IMS* que 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*. Suporta a seguintes funcionalidades [39], [40]:

- Chamada de voz;
- Vídeo 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 tomadas em conta. Funciona em redes móveis como *LTE, UMTS* e *EDGE Networks*. Em termos de segurança tem suporte para *TLS, SRTP* e *RTCP*. Além disso este cliente está sob a licença *GNU GPL v3*, havendo a hipótese de pedir uma licença para fins comerciais *non-GPL* [39], [40].

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* [39], [40]. 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* [39], [40].

3.1.3 IDOUBS

O *idoubs* é o primeiro cliente *3GPP IMS open source* para dispositivos *IOS(iPhone, iPod Touch e iPad)*. Este cliente foi desenvolvido pela *Doubango* e escrito em *ANSI-C*. Suporta a seguintes funcionalidades [41]:

- *Stack SIP/IMS* 7 vezes mais rápida;
- Suporte para *MAC OS X*;
- *Full HD(1080p)* vídeo;
- *NAT* usando *ICE*;
- Suporte para *TLS, SRTP* e *RTCP*;
- Suporte para *codecs* de vídeo: *VP8, H264, MP4V-ES, Theora, H.263, H.263-1998*;
- Suporte para *codecs* de áudio: *Opus, G.722, G.729AB, AMR-NB, iLBC, GSM, PCMA, PCMU, Speex-NB, Speex-WB, Speex-UWB*;
- *Chat*.

3.1.4 JITSI

É uma aplicação/cliente que suporta vários protocolos tais como *SIP*, *XMPP/Jabber*, 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 foi desenvolvida com licença *open source LGPL*. Este cliente tem como principais funcionalidades [42], [43]:

- Chamada de voz/vídeo;
- *Multi-chat*;
- Transferência de ficheiros;

- Criptografia *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 menssagens instantâneas usando *OTRv4*.

Entre outras, as características 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 e SDES* quer no protocolo *SIP* quer no *XMPP*.

3.2 STACKS SIP E SIP/IMS

SIP é um protocolo da camada da aplicação usado para estabelecer e modificar sessões multimídia. Para se efetuar comunicação entre utilizadores usando o protocolo *SIP*, é necessário ter uma *stack* para a comunicação entre ambos. De seguida irão ser apresentadas algumas *stacks SIP*, bem como as suas principais características.

3.2.1 SOFIA-SIP

Sofia-SIP é uma biblioteca *open source User-Agent* SIP e está disponível sob a licença LGPL. Pode ser usado para desenvolver clientes *SIP* para suportar serviços em tempo real tais como chamadas e mensagens instantâneas. Suporta tanto *IPv4* como *IPv6* e a sinalização pode ser encriptada com os protocolos de segurança *SSL/TLS* [44]. Esta biblioteca suporta a seguintes funcionalidades [44]:

- Presença;
- Mensagens Instantâneas;
- Respostas Provisórias;
- Preferência de chamadas;
- Autenticação de Multimédia;
- Vídeo-Conferência;
- *STUN*, método de localizar um *IP* público sob *NAT*.

Para além de todas estas características, *Sofia-SIP* suporta vários *codecs* áudio como G722, GSM, PCMA, MPA, G729, como também alguns *codecs* de vídeo MPV, MP2T, H263, H263- 1998 [44].

3.2.2 SIPML5

SipMl5 é uma *stack* desenvolvida pela *Doubango*. A *Doubango* desenvolveu a sua própria aplicação o *sipMl5* [45] que permite a realização de chamadas de áudio/vídeo para redes *IMS* e *PSTN*. Desenvolveram a sua própria *stack* de forma a possibilitar essa intercomunicação. Assim esta *stack* permite alguns parâmetros de configuração tais como os servidores de *STUN* e *TURN* e o tamanho do vídeo usado [46]. O *sipMl5* permite as seguintes funcionalidades:

- Chamadas vídeo e áudio;
- Mensagens instantâneas;
- Chamada em espera;
- Transferência de chamadas;
- *Multi-line* e *multi-account*;
- Modos de tons em *DMTF* usando *SIP INFO*;

3.2.3 RCS-E STACK IMPLEMENTATION

RCS-e Stack Implementation é uma stack *SIP/IMS* de código aberto, implementada por *Rich Communication Suite* para a plataforma *Android*. Esta *stack* está implementada sob a licença *Apache*. A *API* que se encontra implementada fornece vários tipos de serviço entre eles [47]:

- Chamada (protocolo *RTP*) 1-para-1;
- Re-encaminhamento de chamadas;
- Chamada em espera;
- Vídeo Chamada (protocolo *RTP*) 1-para-1;
- Envio e receção de chamadas;
- Mensagens instantâneas (*Chat*) 1-para-n;
- Transferência de ficheiros;
- Partilha de imagens;
- Partilha de vídeo;
- Gestão e monitorização de chamadas;
- Presença.

Estas características são as funcionalidades base que uma aplicação deste tipo deve ter. Para além destas características os *codecs* de vídeo que são compatíveis são *H.263, H.263-1998, H.263-2000, H.264, WebM (VP8)*, enquanto os de áudio são *AMR-NB, AMR-WB, G711 a- law, G711 u-law, Speex, iLBC*. Esta *stack* foi desenhada de forma a integrar qualquer novo serviço *IMS* [47].

3.2.4 QOFFEESIP

QoffeeSIP é uma *stack* completa, desenvolvida para tirar proveito das funcionalidades da tecnologia *WebRTC*. Esta *stack* não usa *JavaScript* puro, ela usa uma variante do javascript que é o *CoffeeScript* para possibilitar a sua alteração e adaptação a diferentes necessidades. Visto isto, pode ser usada para desenvolver aplicações de tempo real, áudio/vídeo, na *web*. O *QoffeeSIP* esta desenvolvido sob a licença *LGPLv3*. As principais características desta *stack* são [48]:

- Desenvolvida em *CoffeeScript*;
- Tamanho reduzido;
- Chamadas de áudio/vídeo;
- *SIP over Websockets*;
- *Open source(LGPL).*

3.2.5 SIP-JS

Sip-js é uma *stack SIP*, desenvolvida em *JavaScript* pela empresa *OnSIP*. A partir desta *stack* pode-se tirar proveito da tecnologia *WebRTC*. Esta biblioteca foi desenvolvida sob a licença *MIT* [49]. O principal objetivo desta biblioteca é poder adicionar uma pilha de sinalização *SIP* para as aplicações *WebRTC*. Esta *stack* está dividida em vários módulos, nos quais se encontram algumas funcionalidades tais como suporte de mensagens instantâneas, presença, chamadas de áudio/vídeo, entre outras. A *SIP-js* está devidamente documentada, incluindo suporte de alguns exemplos [50]. As principais características desta *stack* são [50]:

- Chamadas de áudio/vídeo;
- Modos de tons em *DMTF* usando *SIP INFO*;
- Suporta *Chrome, Firefox e Opera*;
- Partilha de ecrã;
- Mensagens Instantâneas;
- *SIP Over Websockets.*

3.2.6 PJSIP

PJSIP é uma biblioteca *open source* desenvolvida em C, com suporte para os protocolos *SIP*, *SDP*, *RTP*, *TURN* e *ICE*. Esta biblioteca combina a sinalização *SIP* com suporte para *NAT*, permitindo assim lidar com os problemas de *NAT* que existem na tecnologia *WebRTC*. As principais características desta biblioteca é o suporte de áudio/vídeo, presença, mensagens instantâneas e estar muito bem documentada. *PJSIP* possibilita também o desenvolvimento para dispositivos móveis, devido ao facto de a biblioteca se abstrair dos recursos do sistema, sendo na maioria dos casos possível usar diretamente os recursos do dispositivo. Esta *stack* foi desenvolvida centenas de programadores tendo o seu desenvolvimento começado em 2005. As principais características desta biblioteca são [51]:

- Mensagens Instantâneas;
- Chamadas de áudio/vídeo;
- Suporta os protocolos *SIP, SDP, RTP, STUN, TURN e ICE*;
- Presença.

3.3 SOLUÇÕES WEBRTC EXISTENTES

Embora a tecnologia *WebRTC* seja recente, já se encontram disponível algumas aplicações, algumas delas bastante interessantes. Em seguida serão apresentadas algumas delas bem como as suas funcionalidades.

3.3.1 SIPML5

SipMl5 é uma aplicação *web* [45] que permite a realização de chamadas de áudio/vídeo, partilha de ecrã e mensagens instantâneas. Esta aplicação permite interoperabilidade entre utilizadores de redes *IMS* e *PSTN*. Para isso disponibiliza uma interface de configurações onde é possível definir as configurações da rede a usar. *SipMl5* foi o primeiro cliente *HTML5 SIP open source* desenvolvido em *JavaScript*. Este cliente permite integração com redes sociais, tais como *Facebook, Google+, Twitter*. Estando disponível para todos os *webbrowser* sendo que no *Internet Explorer* necessita de instalação de uma extensão (*webrtc4all*) [45].

3.3.2 PHONO

Phono é um *plugin jQuery* e uma biblioteca *JavaScript* desenvolvida pela empresa *Tropo* [52]. *Phono* permite criar aplicações que sejam capazes de fazer e receber chamadas da rede *PSTN*. Permite também atuar como um cliente *XMPP*, é capaz de enviar e receber mensagens instantâneas para qualquer utilizador *XMPP/Google Talk*. Esta biblioteca tem suporte para os protocolos de segurança *SRTP* e *DTLS*, permite chamadas de áudio/vídeo e mensagens instantâneas. Inclui ainda um *SDK* que permite desenvolver aplicações para *Android* e *IOS* [52]. Para usar esta biblioteca é necessário o uso de uma *key*, que pode ser obtida a partir do site [52].

3.4 BIBLIOTECAS JAVASCRIPT

Com o aparecimento da tecnologia *WebRTC* têm surgido diversos grupos a interessar-se pelo desenvolvimento de bibliotecas que facilitem o desenvolvimento de aplicações, surgindo cada vez mais algumas empresas associadas às mesmas. Desta forma, nesta secção serão apresentadas algumas destas bibliotecas.

3.4.1 PEERJS

PeerJS é uma biblioteca, que fornece um conjunto de funções que fornecem uma forma fácil de usar ligações *peer-to-peer*. Usa um *ID* como forma de identificar um *peer*, este *peer* depois pode criar uma ligação *peer-to-peer* e estabelecer uma ligação com outro *peer* remoto. Esta biblioteca está escrita em *JavaScript* e permite ligações de dados *peer-to-peer*, ligações de áudio/vídeo. Estando disponível para *Chrome* e *Firefox*, *PeerJS* têm um mecanismo em que após um convite para uma conversação passado 5 segundos esse convite é descartado desta forma o utilizador fica apto a rapidamente a receber novos convites de outro utilizador [53].

3.4.2 SIMPLEWEBRTC

simpleWebRTC é uma biblioteca também desenvolvida em *JavaScript* para facilitar o desenvolvimento de aplicações *WebRTC*. Nesta biblioteca é possível configurar algumas váriaveis, tais como se o acesso a câmara/microfone, é feito automaticamente ou se é pedido permissão. Esta biblioteca usa como um servidor de *socket.io*, como meio para efetuar a sinalização entre os diversos *peers*, sendo este servidor fornecido pelos programadores desta biblioteca. Permite ainda a criação de conversações com áudio/vídeo e *chat* com até seis pessoas [54].

3.4.3 WEBRTC EXPERIMENT

WebRTC Experimente é um conjunto de bibliotecas *JavaScript*, que permite de forma fácil criar aplicativos *WebRTC*. Esta biblioteca é formada por quatro subtipos de bibliotecas, a *RecordRTC.js,RTCMultiConnection.js,Translator.js,DataChannel.js,DetectRTC.js,getMediaElement.js* e *ffmpeg.js* [55].

- *RecordRTC.js* - É uma biblioteca que permite ao utilizador gravar as *streams* áudio/vídeo daquela sessão. Os ficheiros de áudio e vídeo são gravados no formato *Wav* e *WebM* respectivamente. É possível ainda gravar imagens animadas que são guardadas em ficheiros *GIF*[56];
- *RTCMultiConnection.js* - É uma biblioteca que permite criar uma abstração das funções do *WebRTC*, permitindo de forma simples e rápida a criação de aplicações, sendo ela personalizável. As principais características são: [57];
  - Chamadas de áudio/vídeo entre múltiplos utilizadores;
  - Partilha do ecrã;
  - Retirar utilizadores de uma conversação;
  - Fazer *mute/unmute* as *streams* de áudio/vídeo;
  - Detecção de presença dos utilizadores;
  - Partilha de dados;
  - Renegociação das *streams* locais e remotas;
  - Abandonar a sessão ou terminar a mesma.
- *Translator.js* - É uma biblioteca que foi desenvolvida pela *Google* que permite o reconhecimento da voz, esta suporta diversas linguagens [58];

- *DataChannel.js* - É uma biblioteca que facilita a criação de aplicações que usem a comunicação por *DataChannel*, esta permite a transferência de ficheiros ou a troca de mensagens de *chat*. As suas principais características são[59]:
  - Troca de mensagens diretas entre utilizadores;
  - Tamanho de ficheiros ilimitado;
  - Tamanho de mensagens;
  - Opção de retirar qualquer utilizador de uma conversação;
  - Abandonar a sessão ou terminar a mesma.
- *DetectRTC.js* - É uma biblioteca que tem como objectivo detectar as características dos elementos envolvidos na comunicação *WebRTC* tais como se o utilizador tem microfone ou câmara instalados [60].;
- getMediaElement.js - Esta biblioteca permite definir a forma como a sessão de *media* irá decorrer, o tamanho do vídeo, se irá ser gravada a sessão ou não, ou definir quais das *streams* pretende gravar áudio/vídeo [61].;
- *ffmpeg.js* - É uma biblioteca *JavaScript* que permite gravar sessões de áudio/vídeo em ficheiros *WAV/WebM*, é usado ainda para fazer o transcode em mp4 ou ogg/mp3 [62].;

3.4.4 ADAPTER.JS

Adapter.js é uma biblioteca escrita em *JavaScript* que têm como principal objetivo garantir a interoperabilidade entre *web browsers*. A tecnologia *WebRTC* usa funções diferentes para aceder às componentes do sistema (câmara/microfone) consoante o *browser* usado, desta forma esta biblioteca serve para unificar estas funções de forma a que uma aplicação funcione em ambos os *web browsers* [63]. Na tabela 3.1 é mostrado as diferentes funções usadas na especificação do *WebRTC* e as suas funções respectivas em cada *browser*.

Tabela 3.1: Funções *WebRTC* usadas pelos diferentes *web browsers* [64].

Funções W3C	**Chrome**	**Firefox**
getUserMedia	webkitGetUserMedia	mozGetUserMedia
RTCPeerConnection	webkitRTCPeerConnection	mozRTCPeerConnection
RTCSessionDescription	RTCSessionDescription	mozRTCSessionDescription
RTCIceCandidate	RTCIceCandidate	mozRTCIceCandidate

3.5 GATEWAYS PARA SIP

Os *Gateways* para *SIP* servem para 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 mecanismos é possível ter interoperabilidade entre diversos sistemas, o que pode ser um grande avanço nas telecomunicações. De seguida irão ser analisados algumas *gateways SIP*.

3.5.1 ASTERISK

Asterisk é um *software open source* que pode ser configurado 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* proprietários. É possível aos utilizadores criar novas funcionalidades escrevendo *scripts* que serão carregados através dos módulos [65].

Na versão 11 do *Asterisk* foi introduzido o suporte à tecnologia *WebRTC* a partir do módulo *res_http_websocket*, o que veio a permitir aos programadores 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* [65].

A partir da versão 10 do *Asterisk* foi adicionado uma biblioteca, a *libsrtp*, para garantir a segurança nas comunicações *RTP*, visto que para que haja comunicação *WebRTC* é necessário ter um canal seguro de comunicação. *Asterisk* suporta uma ampla gama de *protocolos Voice over IP*, 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* [65].

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*. Isto possibilita que uma aplicação totalmente *web* comunique com qualquer tipo de rede *PSTN e 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 responsável por fazer a conversão das *streams media*, de forma a que possa haver comunicação em que 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 diferentes dispositivos que suportem diferentes *codecs*, de forma a garantir a interoperabilidade entre os diferentes dispositivos.

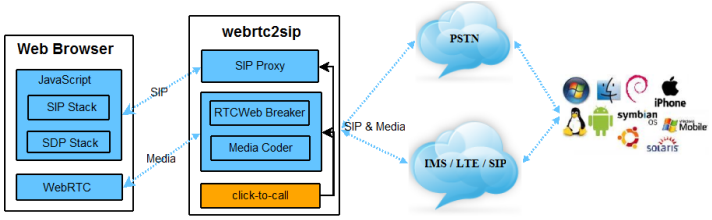


Figura 3.1: Arquitetura do sipMl5[45]

Na figura 3.2 é possível 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 ligar o terminal *SIP* com o *browser*.

Figura 3.2: Arquitetura RTCWeb Breaker[66]

A utilização de diferentes *codecs* é um problema que surge devido ao facto das diferentes aplicações usarem o *codec* mais apropriado à sua utilização. Enquanto que o *Chrome, Mozilla, Opera*, usam o *VP8*, como *codec* de vídeo a *Microsoft* usa o *H.264*. Quanto ao áudio a normalização do *RTCWeb* definiu 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*.

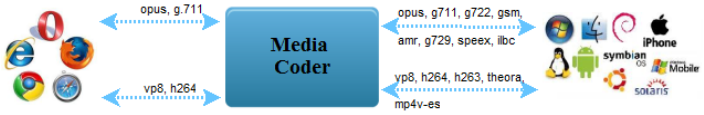


Figura 3.3: *Media Coder sipML5* [66]

3.5.3 JANUS

Janus é uma *gateway* desenvolvida para permitir a interoperabilidade entre aplicações *WebRTC*, e aplicações *SIP*. Esta *gateway* foi desenvolvida em C pela empresa *Meetecho*. A sua implementação é muito simplista, pelo que qualquer funcionalidade extra tem de ser configurada do lado do servidor com recurso a *plugins*. Ela permite a troca de mensagens *JSON* entre o ambiente *WebRTC* e o ambiente *SIP*. *Janus* tem algumas dependências; entre estas é de realçar que a aplicação tem de usar a *stack Sofia-SIP* [67].

3.6 SUMÁRIO

Neste capítulo, foram analisadas algumas soluções existentes *WebRTC* e *SIP/IMS*. No início foram analisadas algumas aplicações que permitissem a comunicação com *SIP/IMS*, sendo analisado a sua interoperabilidade com o *WebRTC* e analisado com os *codecs* que permitiam interoperabilidade.

Em seguida foram analisadas algumas *stacks SIP*, que permitissem a interoperabilidade entre os domínios diferentes, sendo analisado as suas principais características, que serão úteis para construção da solução final.

No final, foram analisadas algumas *gateways* que permitissem a interoperabilidade entre utilizadores *WebRTC* e utilizadores *SIP*, sendo feito uma análise cuidada quanto aos *codecs* suportados pelas mesmas.

capítulo 4

Suporte para Interoperabilidade entre Domínios WebRTC

O desenvolvimento desta dissertação surgiu no âmbito de um projeto Europeu, focado na interoperabilidade entre *WebRTC* e redes legadas. *Wonder* é um projeto Europeu para o desenvolvimento de uma *framework* que facilite o desenvolvimento de aplicações *WebRTC*, dando um enfoque especial à interoperabilidade e à sinalização entre diferentes domínios, tais como *IMS, SIP, Web (Vertx.io), Web (Node.js).*

Embora a tecnologia *WebRTC* seja uma tecnologia bastante recente, o surgimento de aplicações usando esta tecnologia tem vindo a crescer exponencialmente. Anteriormente foram apresentadas algumas das soluções existentes actualmente, algumas já permitem a interoperabilidade entre as redes *IMS* e *PSTN* como é o caso do *sipMl5*, outras apenas implementam comunicação entre dispositivos *WebRTC*.

O objetivo principal desta dissertação é construir uma solução que permita a interoperabilidade entre diferentes domínios, ou seja, um utilizador que está ligado a um servidor de *WebSocktes* (por exemplo *Node.js* [68]), e quer comunicar com um utilizador localizado numa rede *IMS* ou *SIP*, o consiga fazer. Actualmente não existe nenhuma solução que o permita.

Um dos requisitos tidos em conta a quando do desenvolvimento deste projeto foi a simplicidade da solução, de forma a reduzir os custos de implementação, bem como promover a sua adopção por parte dos programadores. Neste capítulo, irá ser explicado mais em pormenor todo o projeto, bem como a solução que permite a interoperabilidade entre redes legadas. Toda a especificação do projeto *Wonder* teve como referência a *wiki* do projeto presente do *GitHub* [69].

4.1 WONDER

Wonder é um projeto que permite interoperabilidade entre utilizadores de diferentes domínios, utilizando conceitos como sinalização *on-the-fly.* O desenho de toda a *API* foi um processo cuidado,

em que foi tido em conta que a solução desenvolvida deveria minimizar o acesso à rede, de forma a evitar topologias de rede do formato da figura 4.1, em que cada utilizador é controlado pelo seu próprio servidor. Para reduzir o acesso à rede, a solução ideal seria ter uma topologia do tipo da figura 4.2 em que os utilizadores são controlados pelo mesmo servidor.

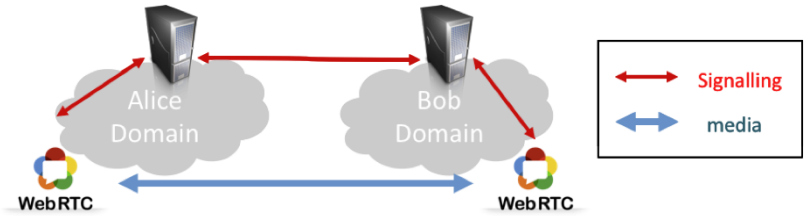


Figura 4.1: Topologia de rede trapezóide [70]

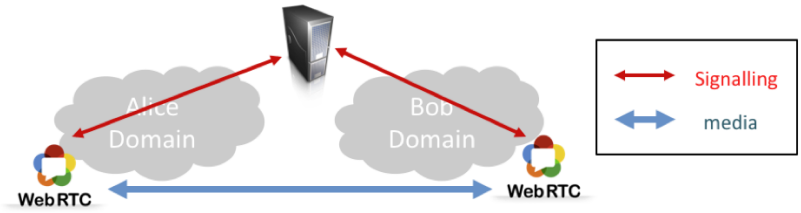


Figura 4.2: Topologia de rede triangular [70]

As principais funcionalidades da arquitetura *Wonder*, são apresentados na figura 4.3:

- *MessagingServer* - fornece todas as funcionalidades necessárias para possibilitar a comunicação interdomínio. Desta forma permite para além da troca da sinalização necessária para comunicação *WebRTC*, também a troca de mensagens (por exemplo, lista de contactos e gerir estados de presença);
- *Identity Provider* - fornece funcionalidades para gerir a informação sobre os utilizadores a partir de uma *API RESTful*;
- *Web Application Server* - fornece a funcionalidade necessária para carregar a aplicação *WebRTC*;
- *Media Gateway* - é usado para possibilitar a interoperabilidade entre utilizadores *WebRTC* com utilizadores não *WebRTC*, como por exemplo, *SIP/IMS*.

Para o desenvolvimento deste projeto foi sugerido o estudo de diversos domínios de aplicações e estudar o funcionamento de cada um. Foram sugeridos quatro domínios de aplicações:

- *PT Web Centric* - este é um domínio *Vertx.io*, que é uma *framework*, que corre em *Java Virtual Machine (JVM)*. Funciona como um servidor, é orientado a eventos e assíncrono, permitindo a criação de aplicações de forma simples e facilmente escaláveis. Esta *framework* permite criação de módulos, em que cada módulo é responsável por executar diferentes funcionalidades, podendo haver comunicação entre eles à partir de um *eventbus*, que funciona como um canal de comunicação [71]. Neste domínio foi desenvolvido todo um conjunto de processos e módulos que permite a interoperabilidade entre aplicações do mesmo domínio e de domínios diferentes;

- *PT SIP* - neste domínio, foi estudado todo um conjunto de processos e componentes que fossem utilizados de forma a permitir interoperabilidade com domínios *SIP*. Este domínio é capaz de comunicar com redes *IMS/PSTN* e aplicações que usem *SIP* como protocolo de comunicação. Este domínio comunica a partir do protocolo *Sip Over WebSockets*, sendo necessário uma *gateway* que serve como intermediário de comunicação. Esta componente é responsável pela tradução de *Sip Over WebSockets* para *SIP*. Para permitir a interoperabilidade entre as redes legadas tiveram de ser instaladas e configuradas diversas componentes, desde a *gateway* que efetua-se a comunicação entre o *WebRTC* e as redes *SIP*, foi também necessário instalar e configurar um servidor de *SIP*, o *Asterisk*. Esta dissertação incidirá mais neste domínio;
- *DT Web Centric* - este domínio *web*, usa como meio de comunicação um servidor *Node.js* que é uma plataforma orientada a eventos, que permite a criação de aplicações *web* escaláveis. Com este servidor é possível criar um canal de comunicação *WebSockets* e a partir daí efetuar toda a troca de sinalização inerente ao *WebRTC*, este servidor contem, à semelhança do *Vertx.io* um repositório com módulos com diversas funcionalidades *Node Package Manager (npm)* [68] ;
- *DT IMS* - neste domínio foi estudado, toda a interoperabilidade entre a tecnologia *WebRTC* e a rede *IMS*. A comunicação é efetuada usando um protocolo de sinalização *JSON over WebSockets*, existindo uma componente da rede *IMS* responsável pela tradução da comunicação no protocolo *SIP*.

Na figura 4.3 é possível ver um esquema de interação dos serviços da *API Wonder*.

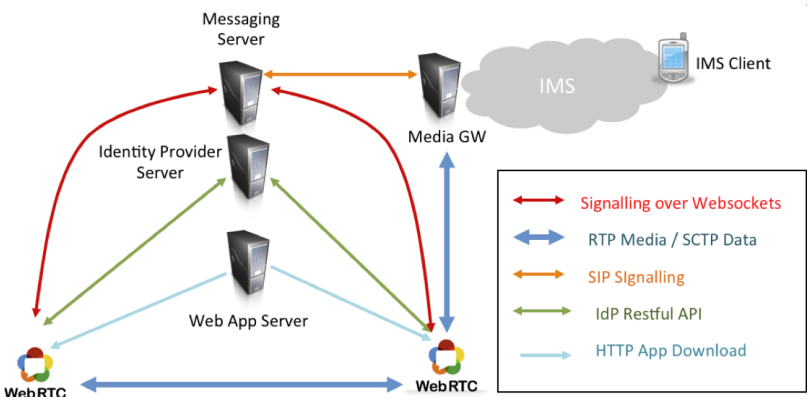


Figura 4.3: Arquitetura funcional *Wonder* [70]

Este projeto é constituído por diversas componentes que permitem a interoperabilidade entre os diferentes domínios. A componente central deste projeto é o *MessagingStub*, que é o responsável pela troca de mensagens com os diferentes domínios, é neste ponto que incide todo o processo de interoperabilidade entre o *WebRTC* e as redes legadas. De seguida irá ser explicado cada uma das componentes bem como as suas principais características.

4.1.1 IDENTITY PROVIDER

O *Identity Provider (IDP)* é uma componente do sistema que armazena a informação relativa aos dados dos utilizadores. Esta informação é guardada numa base de dados não relacional *open source*,

MongoDB[72]. O *MongoDB* é uma base de dados com suporte direto aos documentos *Javascript Object Notation (JSON)* [73]. *JSON* permite um mecanismo de troca de informação entre ambientes *web* e outros. A informação sobre os utilizadores (identidades) pode ser acedida/modificada/apagada diretamente por meio de pedidos *HTTP REST request*.

4.1.2 MESSAGING STUB

O *MessagingStub* é uma das componentes mais importantes desta *API*. Esta componente é responsável por fazer a ligação com os diferentes domínios, ou seja, é responsável por efetuar a troca de mensagens entre a aplicação e o seu domínio, bem como entre diferentes domínios e desligar essas mesmas ligações com o *MessagingServer*. O principal conceito desta componente é que ele apenas lida com a troca de mensagens deixando a parte da sinalização para a restante *API*.

Ele é constituído por três métodos principais, que são: *connect, disconnect, sendMessage*.

No método **connect** é implementada toda a lógica necessária, para ligar a aplicação ao servidor de mensagens *(MessagingServer)*. Isto depende do protocolo de sinalização usado no domínio, tendo como primeiro parâmetro uma *rtcIdentity* que é um *URI*, que indica a identidade do utilizador que se quer ligar na aplicação. No caso de ser necessário uma ligação mais segura o método *connect* poderá ainda receber as credenciais do utilizador.

O método **sendMessage** tem um papel central pois todas as mensagens enviadas devem ser traduzidas para o seu protocolo específico.

O método **disconnect** serve para fechar e limpar o canal de comunicação que foi estabelecido no método *connect*.

4.1.3 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 estático, esperando que algo aconteça. Desta forma todo o processo de funcionamento da biblioteca desenvolvida funciona com base neste conceito. Na figura 4.4 é possível ver o modo de funcionamento de toda a *API*, (nesta figura é possível ver o processo de ligação entre dois *peers*).

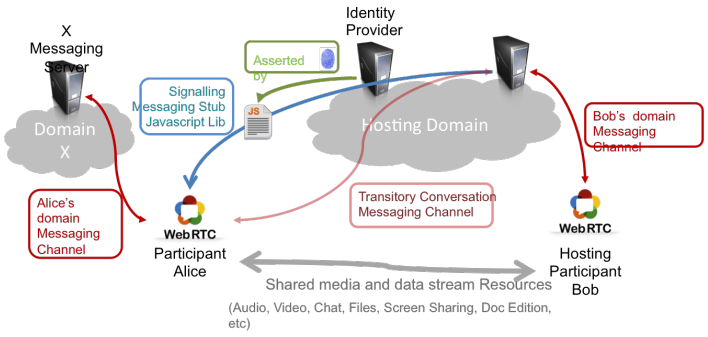


Figura 4.4: Sinalização *on-the-fly* [70]

Uma das vantagens do *Wonder* é o uso de linguagem de *scripting (JavaScript)*, o que permite implementar *stack* de protocolos para comunicar entre *peers*. Ou seja, o *MessagingServer* e a *stack*

de protocolos associada pode ser descarregada e instanciada durante a execução da aplicação. Esta característica permite que os protocolos usados na conversação *WebRTC*, possam ser selecionados consoante o tipo de conversação que se queira estabelecer. A este mecanismo dá-se o nome de sinalização *on-the-fly*. Antes de descrever melhor o conceito, é importante conhecer alguns termos:

- Canal do Domínio *(Domain Channel)* - canal de sinalização que é estabelecido com o servidor de mensagens *(MessagingServer)* do domínio, assim que o utilizador esta registado;
- Canal transitório *(Transient Channel)* - canal de sinalização que é estabelecido, tipicamente com outro servidor de mensagens *(MessagingServer)*, ou seja, a partir de um outro domínio no âmbito de uma conversação.
- *(Messaging Stub)* - *script* que contém a pilha de protocolos e toda a lógica necessária para estabelecer um canal de comunicação com um determinado servidor de mensagens;
- *Host* da conversação *(Conversation Host)*- é o servidor que é usado para apoiar todas as mensagens de conversação, trocados entre os *peers* pertencentes aos diferentes domínios. O utilizador que é o *host* da conversação utiliza canais do domínio e os restantes participantes na conversação usam canais transitórios.

Na figura 4.5, são ilustradas algumas das principais mensagens deste conceito. De forma a perceber melhor este conceito, é apresentado uma lista de funções. Para isso usa-se o exemplo clássico da Alice e Bob, assume-se que estão registados em domínios diferentes. No caso da Alice querer falar com o Bob usando o domínio do Bob, por exemplo *bob@domain.com*, serão realizadas as seguintes etapas:

1. A informação sobre a identidade do Bob, incluindo o *MessagingStub*, é fornecido por um serviço *REST* denominado *Identity Provider*.
2. A Alice faz *download* e instância o *MessagingStub* do Bob no seu *browser*, para configurar um canal transitório com mensagens do domínio do servidor do Bob.
3. Assim que o canal transitório é estabelecido, a Alice pode enviar uma mensagem de convite *(invite)* para o Bob, que irá conter a *offer* com o respetivo *SDP*.
4. Como o Bob está ligado ao mesmo *MessagingServer* através do canal do domínio, ele irá receber o convite *(offer)* no seu *browser*. Caso o Bob aceite o convite, ele irá enviar a sua resposta *answer* através de uma mensagem de aceite *(accepted)* para a Alice, contendo o *SDP* da sua sessão.
5. Assim que o *browser* da Alice recebe a mensagem de aceite com o *SDP* do Bob, a sessão de *media* ou as *streams* podem ser trocadas directamente entre os dois *browsers*.

Este cenário implica, que quem recebe o convite para a conversação está a gastar mais recursos na conversação do que quem envia o convite, uma vez que ambos estão ligados ao servidor de quem recebe o convite. Caso esta solução não seja aceitável é possível que quem envia o convite seja o *hosting* da conversação. As principais diferenças são:

1. Um serviço *restful* de notificação é usado para enviar o convite *(invite)* para o Bob.
2. A identidade da Alice incluindo o *MessagingStub URI* é fornecido e certificado pelo *IDP* da Alice.

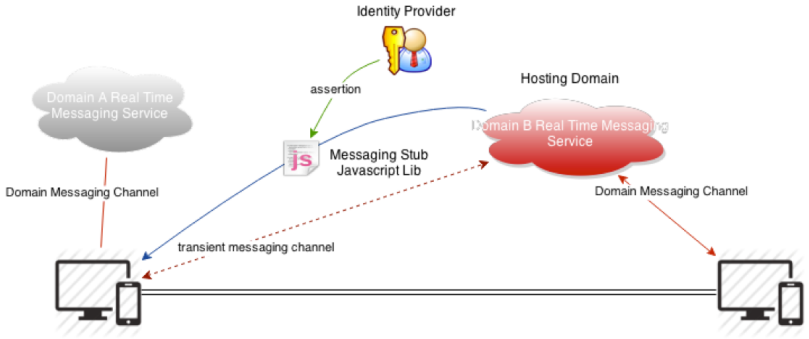


Figura 4.5: Sinalização *on-the-fly* principais mensagens [70]

3. No caso do Bob aceitar o convite *(invite)*, o Bob irá efectuar o *download* e instanciar o *MessagingStub* da Alice no seu navegador, e configurar um canal transitório com o *MessagingServer* da Alice.
4. Após o canal transitório estar estabelecido, o Bob pode enviar uma mensagem de aceite *(accepted)* contendo a sua *answer* com o seu SDP.
5. Como a Alice está ligada ao mesmo *MessagingServer* ela receberá o *SDP* do Bob através do canal do domínio. Depois de receber o *SDP* do Bob, a comunicação e os fluxos de dados podem ser estabelecidos directamente entre os dois *browsers*.

Na figura 4.6, é possível observar todo o fluxo de mensagens trocadas entre as diversas componentes desta *API*. O conceito de sinalização *on-the-fly* pode ser também usado para suportar a interoperabilidade com redes legadas, por exemplo *PSTN*, *IMS*, usando uma *gateway* que ira converter o protocolo de sinalização do dispositivo *WebRTC* para o protocolo de sinalização usado na rede legada, (como se pode observar na figura 4.7).

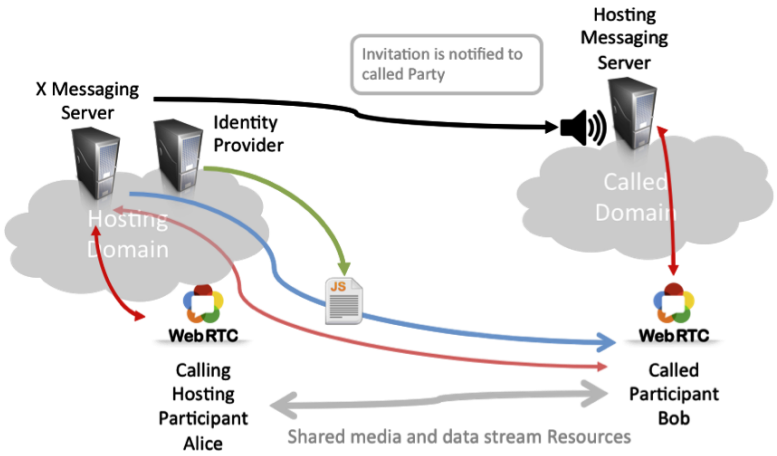


Figura 4.6: Conversação hospedada por quem envia o convite *invite* [70]

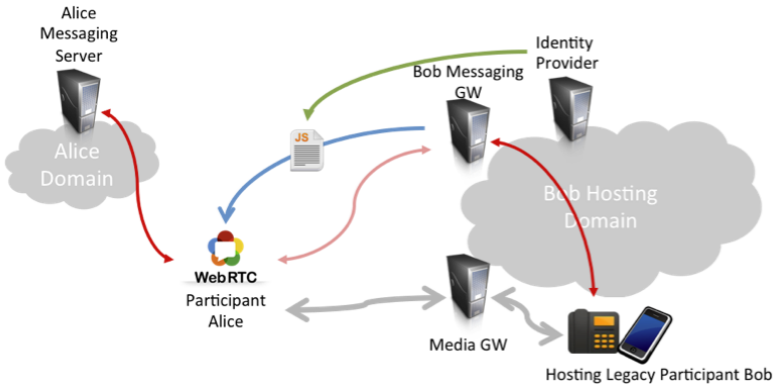


Figura 4.7: Interoperabilidade com redes legadas, por exemplo *IMS*

4.1.4 CODECS ON-THE-FLY

Neste projeto foi dado também um enfoque especial aos *codecs*, de forma a possibilitar que utilizadores que usem o mesmo *codecs* possam comunicar entre si. Na figura 4.8 é possível ver o esquema de funcionamento relativo a estes. Numa conversação entre dois *peers* (Alice e Bob), ambos têm de usar o mesmo *codec* para que a partir do *DataChannel* possam enviar mensagens de texto ou ficheiros. Todo este processo de saber qual o *codec* a usar é executado durante a troca de sinalização.

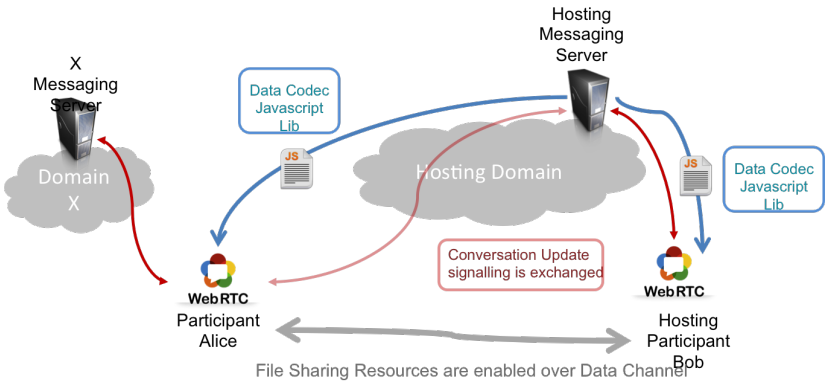


Figura 4.8: *Codecs on-the-fly* [70]

Na figura 4.8 é possível verificar o funcionamento do processo. A Alice comunica com o *Messaging-Server* para saber qual o *codec* que o Bob utiliza para a comunicação de dados, e após esta informação estar trocada a Alice sabe qual a biblioteca que irá ter que descarregar para comunicar com o Bob utilizando o canal de *DataChannel*.

4.1.5 SINALIZAÇÃO MULTI-PARTY

Uma das características desta *API* é possibilitar comunicações *multi-party*. As conversações em *multi-party* também funcionam com o conceito de sinalização *on-the-fly*. Podem ser usadas diferentes topologias da rede.

1. Topologia de rede hospedada *(hosting)* por quem convida, todos os pares estão ligados a um servidor de mensagens e são trocados todos os fluxos de dados directamente entre utilizadores,

desta forma todos os pares de utilizadores têm um canal de sinalização estabelecido com o mesmo *MessagingServer* (figura 4.9).

2. Neste tipo de rede não existe nenhum *hosting*, todos os pares de utilizadores têm meios diretos de fluxos de dados estabelecidos com os outros pares de utilizadores, ou seja, cada par tem n-1 canais de sinalização estabelecidos com o *MessagingServer*, onde n é o número de pares envolvidos na conversação (figura 4.10).

3. Rede baseada num *Multipoint control unit(MCU)* em que todo o fluxo de dados passa por uma *gateway*. Neste caso todos os pares têm meios de comunicação e fluxos de dados estabelecidos com um servidor central de *mídia*, ou seja todos os pares têm um canal de sinalização estabelecido com o mesmo *MessagingServer* (figura 4.11).

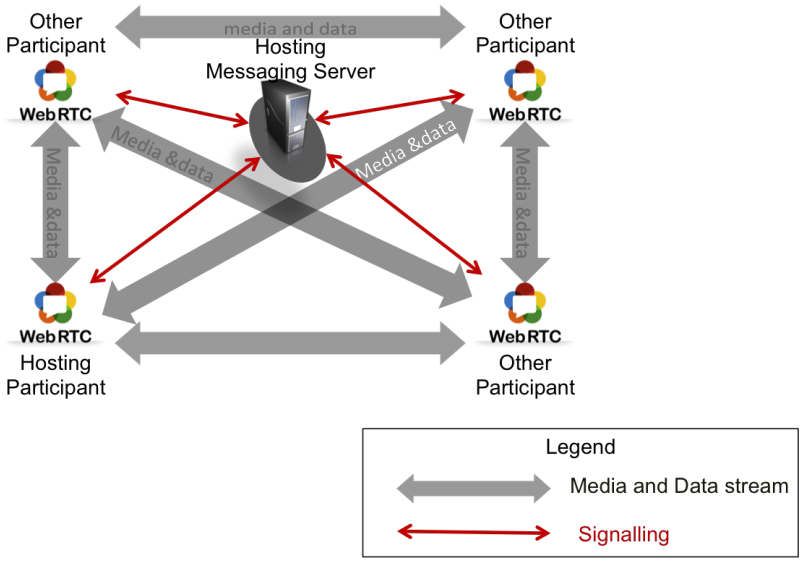


Figura 4.9: Rede *Multi-Party* com *hosting* [70]

De forma a perceber melhor o funcionamento de toda a *API*, bem como todas as mensagens trocadas, é apresentada um diagrama de sequência (figure 4.12) para tornar mais claro essa interação.

Convite e notificação

(1) Alice organiza uma conversação com vários utilizadores, e convida o Bob e a Carol, que estão registados em diferentes domínios.

(2-3) O *MessagingServer* da Alice invoca o *Identity Provider* e obtém informação sobre a identidade do Bob e da Carol, depois envia uma notificação com o convite *(invite)* para a conversação.

Bob aceita o convite

(4-5) Bob faz *download* e instância o *MessagingStub* no seu *browser*, e configura o canal transitório com o *MessagingServer* da Alice. Após este canal estar estabelecido, o Bob envia uma mensagem de aceite/*accepted(answer)* contendo o seu *SDP*.

(6) Alice recebe a mensagem de aceite do Bob.

Alice faz um *publish* a dizer que o Bob aceitou o convite

(7-8) Quando a Alice recebe uma mensagem do Bob, a dizer que ele aceitou o convite para a conversação, verifica que foi ela é o *hosting* para a conversação e envia uma mensagem para os outros participantes sem *SDP*, e acrescenta um campo à mensagem, a dizer quem é que já esta ligado a àquela conversação. Desta forma, é possível os participantes da conversação identificarem se já estão ligados a

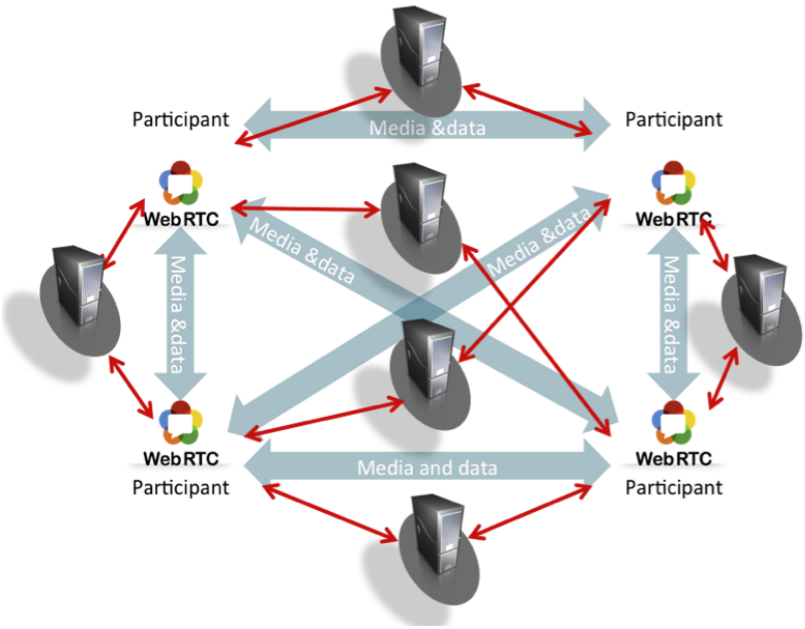


Figura 4.10: Rede *Multi-Party* sem *hosting* [70]

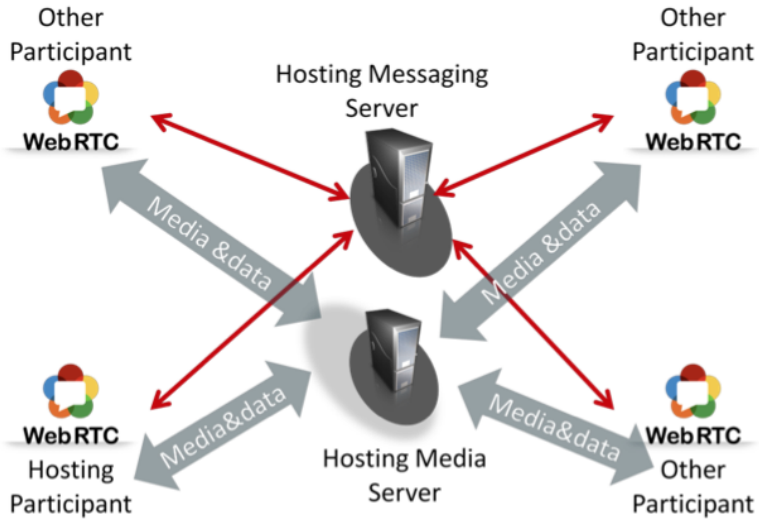


Figura 4.11: *Multi-Party* com fluxo de *media* com topologia estrela [70]

todos os outros participantes, caso não estejam, eles irão ter de enviar um convite *(invitation)* para os outros participantes de forma a ficarem ligados entre si. A mensagem de aceite do Bob irá ser guardada em *cache* no *MessagingServer* de forma a estar disponível para novos participantes.

Carol aceita o convite

(9-10) Carol faz *download* e instância o *MessagingStub* no seu *browser*, e configura o canal transitório com o *MessagingServer* da Alice. Após este canal estar estabelecido o Bob envia uma mensagem de aceite/*accepted(answer)* contendo o seu *SDP*.

(11) Carol recebe a mensagem de *accepted* a dizer que Bob aceitou o convite, esta mensagem foi publicada *published* pela Alice (passo 7).

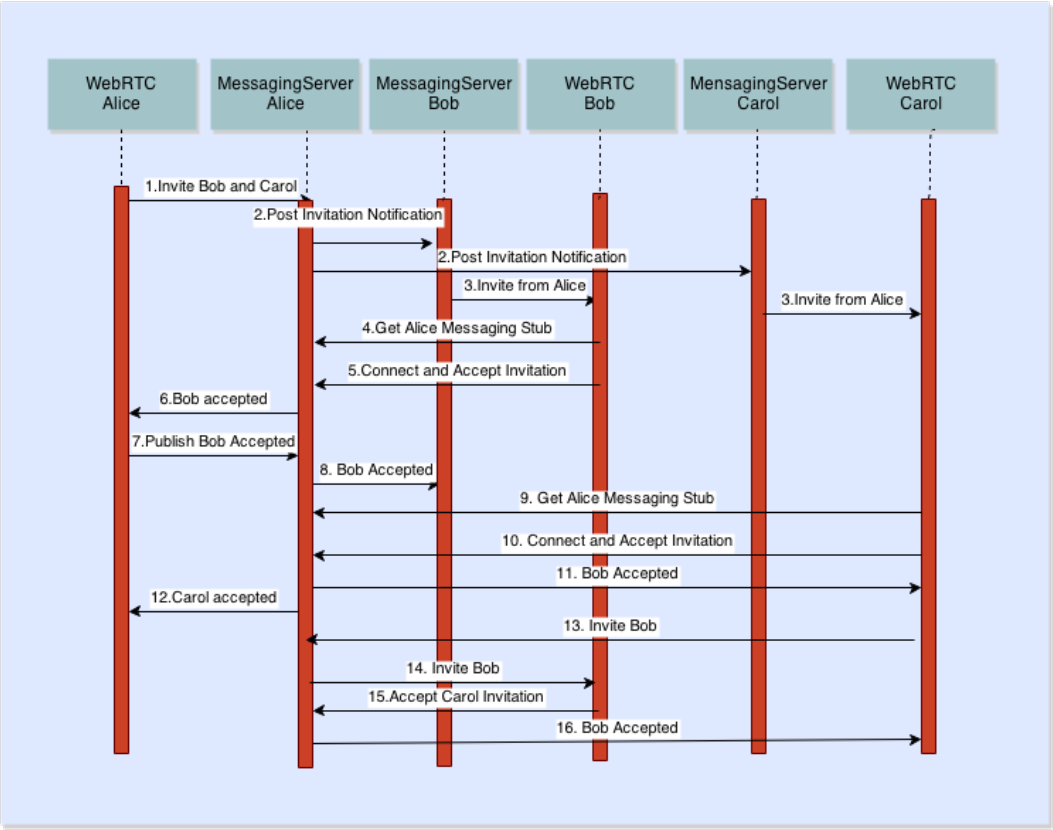


Figura 4.12: Diagrama de sequência da troca de mensagens

(12) Alice recebe a mensagem de aceite da Carol, e faz um *publish* dessa mensagem sem o SDP, e com a informação de quem já esta ligado, ou seja, Bob e Carol, para todos os participantes. Bob recebe a mensagem a dizer que a Carol aceitou o convite, e verifica que esta na lista e ignora.

Carol é o último parcipante a juntar se a conversação e convida os outros participantes ligados(Bob)

(13-14) Assim que a Carol é informada que o Bob entrou na conversação (passo 11), ela verifica que ele não está na lista de participantes a quem esta ligado e envia um convite para o Bob. A mensagem de convite com o *contextID* da conversação criada pela Alice. Desta forma é estabelecida um canal de ligação *PeerConnection* entre a Carol e o Bob permitindo assim a comunicação entre ambos.

(15-16) Bob aceita automaticamente o convite da Carol e responde com o seu *SDP*, após este passo está estabelecida a comunicação entre ambos a partir de uma *PeerConnection*

4.2 TIPOS DE MENSAGENS TROCADAS

A comunicação entre os diferentes participantes é feita usando vários tipos de mensagens. A partir desta comunicação é possível estabelecer ligações entre os diferentes participantes e adaptá-los as novas características da conversação. As mensagens trocadas entre os participantes podem ser:

- Convite *(Invitation)* - mensagem de convite *(offer)* serve para convidar um peer para uma conversação;

- Aceite *(Accepted)* - esta mensagem corresponde à resposta a um convite *answer*, de forma a indicar que o participante aceitou participar naquela conversação;
- Candidato *(Connectivity Candidate)* - mensagem que contém um candidato (*ICE*) ;
- Não aceite *(Not Accepted)* - mensagem a indicar que o participante não aceitou o convite para a conversação;
- Cancel - mensagem a indicar que foi cancelada aquela conversação;
- Adeus *(Bye)* - mensagem a indicar que um participante deixou aquela conversação, caso seja o *owner* da conversação termina a conversação;
- Atualizar *(Update)* - esta mensagem indica que irá ser adicionado um novo *resource* à conversação;
- Atualizado *(Updated)* - mensagem que é a resposta ao atualizar a indicar que a conversação foi atualizada.

As mensagens têm determinadas características, assim, no *header* da mensagem, deverá existir a seguinte estrutura:

- *Id* - identificação da mensagem;
- *type* - tipo de mensagem enviada;
- *from* - *rtcIdentity* de quem envia a mensagem;
- *to* - lista de *rtcIdentity* a que se destina a mensagem;
- *context-id* - identificação da conversação associada;

No caso do *body* da mensagem também é seguida uma estrutura. O *body* da mensagem deverá conter: o *ConnectionDescription (SDP)* associado; quem é o *hosting* da conversação; quais os *codecs* associados à conversação.

4.2.1 CLASSES WONDER

Wonder é uma *framework* que facilita o desenvolvimento de aplicações *WebRTC*. Na figura 4.13, é possível observar as diferentes classes por que é constituída. No anexo 6.2 pode-se observar o diagrama de classes desta *API*.

A classe ***Identity*** representa um utilizador 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* [74], 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 *media* e dados da ligação entre *peers* através do servidor de mensagens. Pode ser usado ainda para outros fins, como por exemplo, gestão de informações de presença.

A classe ***Conversation*** faz a gestão de todos os participantes, incluindo a gestão do *media* e das ligações de dados.

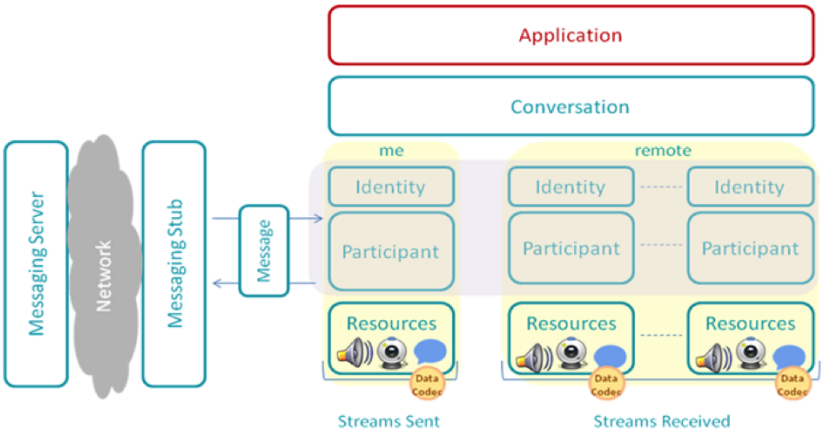


Figura 4.13: Principais classes *Wonder* [70]

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

A classe ***Resource*** representa o *media* que são partilhados entre os diferentes participantes numa conversação, incluindo voz dos participantes, vídeo, áudio, fotos, etc. Estes recursos são geralmente administrados pelo participante que os possui. Outro tipo de recursos como *chat* não são administrados pelo participante mas sim pela conversa.

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

4.3 DESENVOLVIMENTO DA PROVA DE CONCEITO

Wonder tem como um dos seus objetivos construir toda uma *API*, que permita aos programadores criar aplicações que usem a tecnologia *WebRTC* de forma simples, e que suporte grande parte dos serviços disponibilizados pela tecnologia. Outro dos objetivos do *Wonder*, é permitir a interoperabilidade entre os diferentes domínios de referência *(PT Web (vertx.io), DT Web (node.js), PT SIP, DT IMS)*. Esta dissertação inseriu-se neste projeto pelo estudo e realização de um *MessagingStub* que permitisse a comunicação com redes legadas. A construção do *MessagingStub* que permitisse a comunicação com as redes legadas, foi um processo que envolveu todo um estudo de diversas *gateways*, como das diversas *bibliotecas SIP*.

Na construção do *MessagingStub* (*PT SIP*) foi usado como base a aplicação *sipml5* [45], que usa a biblioteca do *sipml5* [75]. Foi feita uma análise a mesma, identificando as principais funções que poderiam ser reutilizadas na construção do *MessagingStub*. Todo o processo de desenvolvimento do *MessagingStub* foi contínuo sendo acompanhado de testes para garantir o funcionamento do mesmo.

Foi necessário o uso de uma *gateway*. Neste caso foi usada a *webrtc2sip* [66], que irá ser responsável por efetuar o *encoding* e *decoding* de todo o tráfego entre o domínio *web* e o domínio *SIP*.

Na figura 4.15, é possível observar um esquema de todas as mensagens trocadas para o estabelecimento de uma sessão entre um utilizador registado numa rede legada com um utilizador registado num *web browser*. Sendo a comunicação do lado do *web browser* efetuada por meio de *Sip Over Websockets*,

depois a sua tradução é efetuada na *gateway*. Após a tradução, os pacotes *SIP* são encaminhados para a rede legada.

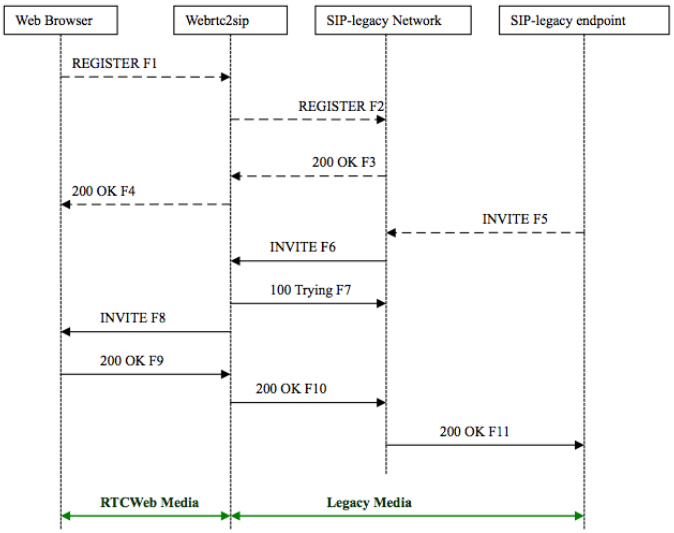


Figura 4.14: Exemplo de fluxo de mensagens trocado para estabelecimento de uma sessão *RTP* [66] [70]

Em seguida será apresentado todo o processo de criação do *MessagingStub PT SIP* bem como a aplicação desenvolvida a partir da *API Wonder*.

4.4 EXPERIMENTAÇÃO E SELEÇÃO DE TECNOLOGIAS EXISTENTES

Para o desenvolvimento deste projeto/dissertação foram estudadas e experimentadas algumas tecnologias e componentes para o desenvolvimento do mesmo, para que seja possível criar uma solução de forma organizada.

O objetivo principal deste projeto/dissertação é construir uma solução que permita interoperabilidade entre diferentes domínios, incidindo principalmente esta dissertação na interoperabilidade com os domínios *IMS* e *PSTN*. Desta forma foi desenvolvida uma aplicação *web*, em que foi usado como linguagem de programação o *JavaScript*, que permitisse a interoperabilidade com as redes legadas. Para construção desta mesma solução foi usado uma biblioteca *WebRTC* existente que é o *Adapter.js*, que tem como principal objetivo garantir a interoperabilidade entre *browsers*. De seguida irão ser apresentadas as escolhas efetuadas para a construção da solução.

Gateways

Após uma análise detalhada e cuidada das *gateways* existentes no momento de realização, a escolha feito foi para o uso da *gateway webrtc2sip* da *Doubango*. Foram realizadas diversas experiências usando diversos cenários, em que consistia estabelecer chamadas de áudio/vídeo entre um cliente *WebRTC* e um cliente *SIP* por exemplo *Softphone*. Usando o *Asterisk* como *gateway*, foi possível estabelecer chamadas apenas de áudio, notando-se alguns problemas em relação ao vídeo. Quanto a *gateway Janus*, devido a sua implementação ainda ser minimalista, o seu uso foi descartado logo ao início.

No caso da *gateway webrtc2sip* foram realizados diversos testes, de forma a garantir que era possível estabelecer chamadas entre um cliente *WebRTC* e um cliente *SIP*. Os resultados dessas experiências foram positivos, pois foi possível ter uma comunicação de *áudio/vídeo* em ambos os sentidos, sendo possível estabelecer chamadas com diversos *Softphones(X lite,Linphone, Zoiper,etc)*.

Servidores SIP

A escolha do servidor *SIP* mais adequado a solução, foi um processo que teve de ter em conta a solução actual e já implementada na rede empresarial e que permite a interoperabilidade entre a rede *PSTN*. Na rede empresarial é usado o *Asterisk* ligado a um *Private Automatic Branch Exchange (PABX)*. Este sistema permite a comutação de chamadas automaticamente. Este sistema permite não só lidar com tráfego de telefonia da rede interna da empresa como a comunicação do tráfego para a rede empresarial.

A solução optada para servidor *SIP* foi a do *Asterisk*, uma vez que a rede empresarial já faz uso do mesmo, por outro lado para não correr alguns riscos foi feita uma instalação a parte da existente de forma a garantir o funcionamento da rede empresarial. Para possibilitar a nova instalação de comunicar com a rede *PSTN* foi configurado um *SIP Trunking*, entre ambos, que é uma forma simples de encaminhar o tráfego entre os servidores.

4.4.1 MESSAGINGSTUB - PT SIP

Para o desenvolvimento do *MessagingStub PT SIP*, foi utilizado como base a aplicação do *sipml5*, sendo todo o trabalho efetuado em torno da adaptação à *API Wonder* [75] [45]. Os principais métodos do *MessagingStub PT SIP* são o *connect, sendMessage e o handleMessages*.

Connect

No método *connect* é preciso estabelecer um canal de comunicação com o servidor de mensagens *(MessagingServer)* nesta caso será usado o *Asterisk* que irá permitir a intercomunicação com a *PSTN*. O canal de comunicação é estabelecido usando como protocolo de comunicação o *Sip Over WebSockets*, este canal de comunicação é feito recorrendo a uma ligação *WebSockets*. Nesta ligação vão estar definidos métodos para receber/enviar e tratar de erros das mensagens (*websocket.onmessage(), websocket.onerror(), websocket.onopen()*).

Ao ligar-se ao servidor de mensagens é necessário verificar se o utilizador é do domínio *PT SIP*. Caso não seja é necessário usar uma extensão *"_friend"* a indicar que é um utilizador de um domínio diferente. A partir deste mecanismo o utilizador conseguirá ligar-se ao servidor de mensagens do domínio *PT SIP* e comunicar com utilizadores desse domínio. A partir deste mecanismo é possível que utilizadores de domínios diferentes comuniquem com o domínio *PT SIP*.

No método *connect* é feito uma série de passos que irão permitir a comunicação com o ambiente *SIP*:

1. Um utilizador ao registar-se na aplicação tem associado a si uma identidade que é representada pela forma *user@domain*. Ao iniciar a sessão, a aplicação irá descarregar e instanciar o *MessagingStub* associado ao seu domínio (conceito sinalização *on-the-fly*);

2. De seguida ao ser descarregado o *MessagingStub*, o utilizador invoca a função *connect* do *MessagingStub*, onde se irá verificar se o utilizador é do domínio *PT SIP* (*imsserver.ece.upatras.gr*). Caso o utilizador seja de um domínio diferente é adicionado o sufixo *"_friend"*, e assim poderá

ligar-se ao servidor de mensagens *PT SIP* (*Asterisk*). De notar que as contas terão de ser aprovisionadas previamente no *Asterisk*, para que a aplicação funcione corretamente;

3. Em seguida é criada uma *stack SIP* que irá permitir ao utilizador comunicar com o servidor de mensagens *(Asterisk)*. A criação da *stack* é feita recorrendo a biblioteca do *sipml5* [46]. Esta aceita algumas configurações tais como:
   - Indicar quais os *IP's* em que se encontra o servidor *SIP* (*outbound_proxy_url*);
   - Indicar qual o endereço usado pela *gateway* (*webrtc2sip*) (*websocket_proxy_url*);
   - Podendo ser configurado a transcodificação ou não da comunicação, isto é, caso os dois utilizadores não usem os mesmos *codecs* fazer a transcodificação do tráfego *RTP* entre ambos, isto é possível através da flag *enable_rtcweb_breaker*;
   - Adicionar um *listener* para receber as mensagens associadas a *stack* (*events_listener*);
   - Indicar quais os servidores de *STUN* e *TURN* que irão ser usados;
   - Quais as credenciais associadas ao utilizador que cria a *stack* (*realm* - domínio, *impi* - *identity*, *impu* - endereço *SIP*, *password* - password, *display_name* - nome a apresentar).
4. Após o registo estar efetuado com sucesso é possível o utilizador fazer uma chamada entre utilizadores do domínio *PT SIP*. Para isso terá de usar o método *sendMessage* e enviar um *INVITE*.

No trecho de código abaixo, pode-se observar parte do código presente na função *connect*. Depois do utilizador estar ligado ao servidor de mensagens toda a troca de sinalização poderá ser efetuada, para isso é usado o método *sendMessage* que será explicado de seguida.

```
var name = ownRtcIdentity.split("@")[0];

if(ownRtcIdentity.split("@")[1] != "imsserver.ece.upatras.gr"){
    name = name + "_friend";
}

o_stack = new SIPml.Stack({
            realm: 'imsserver.ece.upatras.gr',
            impi: name,
            impu: 'sip:' + name + '@imsserver.ece.upatras.gr',
            password: pass,
            display_name: ownRtcIdentity,
            websocket_proxy_url: 'ws://10.112.67.66:10060',
            outbound_proxy_url: 'udp://10.112.67.66:5060',
            enable_rtcweb_breaker: true,
          ice_servers: "[{ url: 'stun:150.140.184.242:3478'}, {
             url:'turn:root@150.140.184.242:3478',
             credential:'w0nd3r'}]",
            events_listener: { events: '*', listener:
               onSipEventStack },
            enable_early_ims:  true, // Must be true unless you're
               using a real IMS network
```

```
            sip_headers: [
                { name: 'User-Agent', value: 'IM-client/OMA1.0
                    wonder-v0.2013.12.10B' },
                { name: 'Organization', value: 'Wonder PTIN' }
            ]
        }
    );
```

No anexo 6.2 é possível observar as mensagens de registo e estabelecimento de uma chamada entre *WebRTC* e a *PSTN*.

sendMessage

Após a ligação ao servidor de mensagens *(MessagingServer)* estar estabelecida, é preciso trocar a sinalização com o outro utilizador para que a sessão *RTP* fique estabelecida. Uma das vantagens da *API Wonder* é que trata de toda a parte de sinalização deixando apenas a troca de mensagens para o *MessagingStub*. Desta forma o método *sendMessage* irá receber todas as mensagens provenientes da *API Wonder* e será responsável pela sua troca com o servidor de mensagens.

No trecho de código abaixo, é possível ver parte da implementação deste método. Ao receber as mensagens da *API Wonder*, irão ser feitas algumas comparações para identificar cada tipo de mensagem para serem tratadas de forma diferente. Caso a mensagem seja uma mensagem de *bye*, será terminada a sessão a partir de uma função (*hangup*) da biblioteca do *sipml5* [75]. A *API Wonder*, usa um mecanismo em que ao mesmo tempo que é feito o *gathering* dos candidatos eles são enviados ao outro utilizador. Ou seja, é enviado uma mensagem com o *INVITE*, e depois seguem-se mensagens de *ConnectivityCandidate*, este mecanismo torna a comunicação entre utilizadores mais rápida. Mas na comunicação *SIP* o mesmo não acontece é necessário esperar que seja feito o *gathering* dos candidatos e só depois é enviado o *INVITE* contendo todos os candidatos no *SDP*.

De seguida é verificado se a mensagem recebida é uma mensagem referente ao último candidato. Caso seja é enviado um *INVITE* a partir da função *callSIP* para se estabelecer uma ligação com outro utilizador. No caso de ser uma resposta a um convite (*INVITE*) será enviado um *OK*, o *OK* será enviado a partir do *INVITE* recebido sendo necessário trocar os campos *from* e *to* da mensagem, bem como o respetivo *SDP*.

No caso de ser uma mensagem de *INVITE*, é guardado para quem se pretende fazer a ligacação *calle*, e o *contextID* associado a conversação, que corresponderá ao *Call-ID* na mensagem *SIP*. Se for uma mensagem de *Accepted*, é modificado o valor de uma variável auxiliar que irá permitir identificar que foi recebido um convite para uma chamada.

```
if(message.type == MessageType.BYE){
    callSession.hangup({events_listener: { events: '*', listener:
        onSipEventSession }});
}

if(message.body.lastCandidate){
    sdp =message.body.connectionDescription;
    if(called == true){
        thate.send_response(thate.o_last_iInvite, 200, "OK", true);
```

```
    }else{
        this.callSIP();
    }
}else{
    if(message.type == MessageType.INVITATION){
        calle= message.to.rtcIdentity;
        callID = message.contextId;
    }
    if(message.type == MessageType.ACCEPTED){
        called = true;
    }
}
```

No anexo 6.2 pode-se observar um exemplo de um *SDP* trocado no estabelecimento de uma chamada para a *PSTN*.

handleMessages

Outro dos métodos principais do *MessagingStub* é o *handleMessages*. Neste método são tratadas todas as mensagens que são recebidas (*INVITE,OK*). As mensagens *SIP* chegam a aplicação encapsuladas nos pacotes *WebSockets*, sendo a sua recepção e descodificação feita pelas funções reutilizadas da aplicação do *sipml5*. Na função *___tsip_transport_ws_onmessage* são recebidas as mensagens *WebSockets*, sendo necessário retirar os campos (*from,to*) das mensagens para identificar quais utilizadores a que se destina a mensagem.

A partir da descodificação dos campos *from,to* é criado as mensagens de *(invitation/accepted)* e são enviadas para a *API Wonder* com toda a informação incluíndo o *SDP*. A *API Wonder* ao receber as mensagens irá tratar da sinalização inerente a tecnologia *WebRTC* e enviar as mensagens de resposta as mesmas. Na comunicação com a *PSTN* são necessários alguns cuidados pois é preciso garantir que as *constraints* são unicamente para acesso ao micro. Daí na função *handleMessages* ser feito a verificação se a chamada é proveniente de um número da *PSTN*, nesse caso é garantido que o campo das *constraints* é enviado para a *API* só com as *constraints audioMic*. Em seguida é verificado se o utilizador remoto já tem associado o *IDP*; caso não tenha é instanciado um, isto porque o *IDP* é um *Singleton*, sendo posteriormente enviadas as mensagens para a *API Wonder* (*baseStub.deliverMessage(message)*).

Outra das funções utilizadas da aplicação *sipml5* é a função que faz o *parser* do *SDP* (*tmedia_session_jsep01.prototype.___set_ro*). No trecho de código a baixo é possível observar parte da implementação da função *handleMessages*.

```
    var c = s.split('@')[0];
    if(/^-?[\d.]+(?:e-?\d+)?$/.test(c)){
        full_message.body.constraints = [{
        constraints: "",
        type: ResourceType.AUDIO_MIC,
        direction: "in_out"
        }];
    }
```

```
var message = full_message;

Idp.getInstance().createIdentity(message.from,
    function(identity) {
     message.from = identity;

     if(message.to instanceof Array && message.to[0] ===
        null){
         that.baseStub.sendOtherMessages(message);
     }
     else{
         // createIdentities can take array or single
            rtcIdentity and always returns an Array in the
            callback result
         Idp.getInstance().createIdentity(message.to[0],
            function(identityArr) {
             message.to = identityArr;
             // Filter the listeners to redirect the message
             that.baseStub.deliverMessage(message);
         });
     }

});
```

4.4.2 DESENVOLVIMENTO DA APLICAÇÃO

De forma a demonstrar a interoperabilidade com as redes legadas foram usados dois cenários de testes, isto porque na rede *IMS* usada para testes no projeto *Wonder* não era possível haver comunicação com a *PSTN*. Por este facto foram usadas duas aplicações distintas, uma delas desenvolvida de raiz e outra que tinha sido criada anteriormente pela empresa.

APLICAÇÃO IMS

Na realização de testes na rede *IMS* foi utilizada a aplicação da figura 4.15 [76]. Esta aplicação foi desenvolvida de raiz, sendo a sua implementação muito simplista. Esta aplicação permite interoperabilidade entre os diferentes domínios (*DT IMS, DT Web, PT Web, PT SIP*). O utilizador para se registar terá de ter uma conta aprovisionada na base de dados para que tal seja possível.

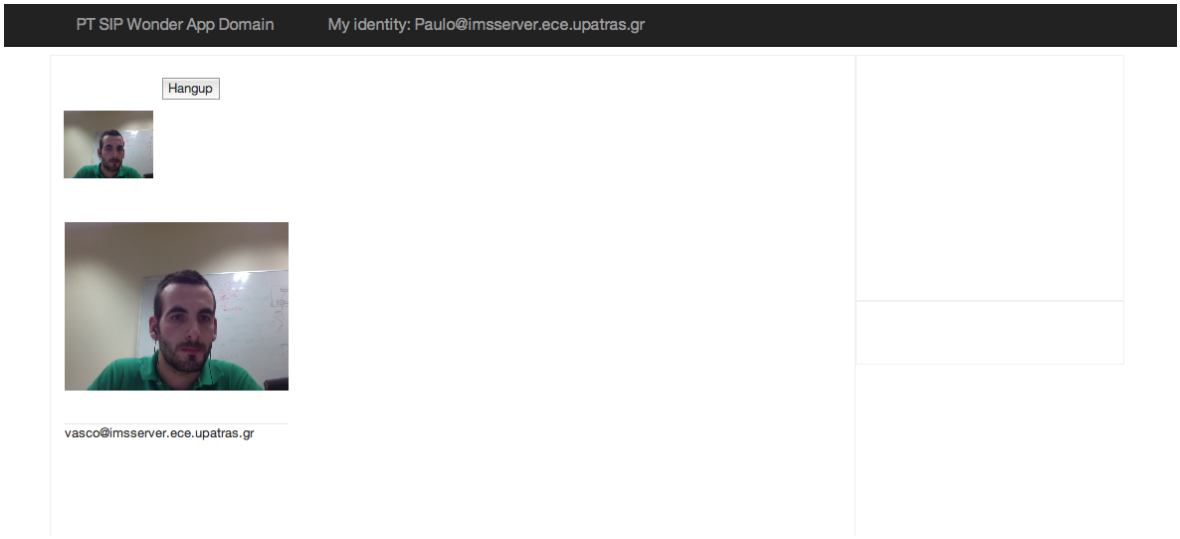


Figura 4.15: Aplicação desenvolvida para testes na rede *IMS*.

APLICAÇÃO PSTN

Para realização de testes com a rede *PSTN* foi reformulada uma aplicação anteriormente desenvolvida pela empresa (PT Inovação e Sistemas). A aplicação da figura 4.16 foi reformulada de forma a usar a *API Wonder*. Esta aplicação tem como principais características:

- Registo de utilizadores;
- Pesquisa de utilizadores;
- Lista de contactos (adicionar/remover amigos);
- Estados de presença dos contactos;

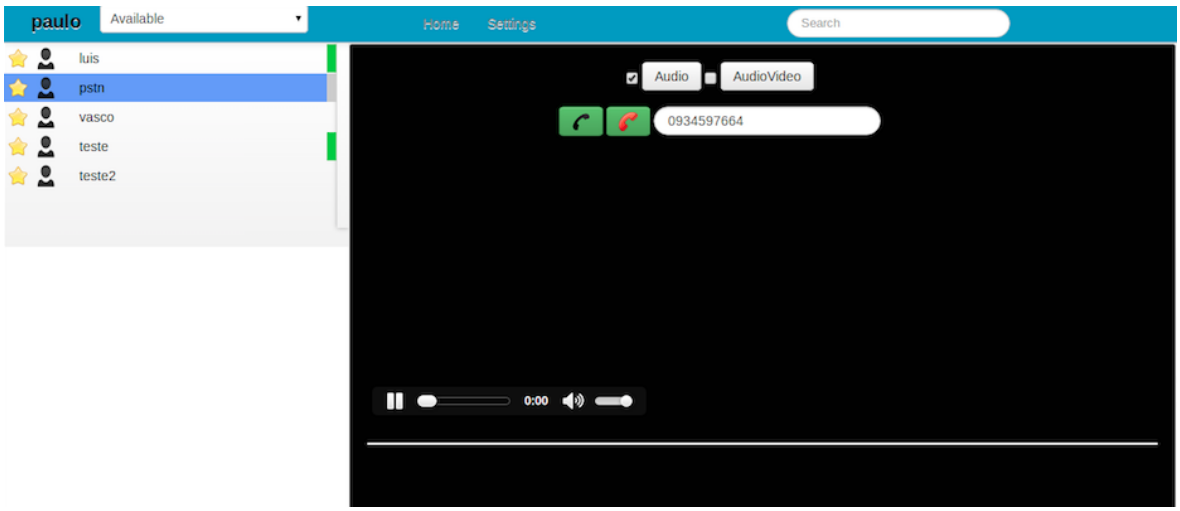


Figura 4.16: Aplicação reformulada para testes com a rede *PSTN*.

4.5 SUMÁRIO

Neste capítulo foi sumariado todo o projeto desenvolvido nesta dissertação, sendo especificado cada componente por que é constituído.

No início foi apresentado o projeto e conceito em que se insere, apresentando-se algumas das componentes essênciais ao mesmo, em seguida foi apresentado a componente mais importante que é o *MessagingStub*. No *MessagingStub* residirá a capacidade de trocar mensagens com as redes legadas e estabelecer sessões com as mesmas.

Foi ainda apresentado o principal conceito que é a sinalização *on-the-fly*, que irá permitir descarregar e instanciar um canal de comunicação para que se possa estabelecer um canal de comunicação com aplicações de domínios diferentes do próprio. É apresentado todo o processo de estabelecimento de uma chamada para um domínio diferente passo a passo.

Neste capítulo são também apresentados as aplicações desenvolvidas/alteradas para demonstrar a interoperabilidade entre os diferentes domínios.

No final são apresentados as classes por que é constituída toda a *API*, bem como as mensagens trocadas entre os participantes.

capítulo 5

Cenários de teste e resultados obtidos

Neste capítulo serão apresentados os testes realizados e os seus resultados. Para realização deste projeto foram disponibilizadas algumas máquinas com instalação do *Ubuntu 11.10/oneiric*, com as componentes necessárias para todo o processo de testes. As componentes instaladas nas diversas máquinas permitem criar a base para os diferentes domínios, sendo algumas delas de âmbito geral.

Devido a impossibilidade de testar a intercomunicação com a *PSTN*, foi necessário configurar outro cenário de testes de forma a testar o mesmo. Desta forma, foram usados dois cenários de testes um deles localizado na Universidade de Patras, que possui a rede *IMS*, e outro localizado na rede da empresa, que possui uma ligação à *PSTN*.

Todo o processo de validação da *API* foi contínuo, de forma a verificar todo o correto funcionamento da mesma.

Primeiramente serão apresentadas as componentes usadas para os testes na rede *PSTN*, e de seguida é apresentado o teste às componentes usadas na rede *IMS*. Por fim irá ser feita uma análise geral aos testes de interoperabilidade entre os diferentes domínios, bem como a qualidade de transmissão em ambos.

5.1 interoperabilidade entre pstn

Para realização dos testes com redes *PSTN* foi usado um cenário na rede interna da empresa, foram requisitadas algumas *VM's*, em que tiveram de ser instaladas e configuradas diversas componentes. Como é possível ver na figura 5.1, todas as componentes usadas. Foram instaladas de novo à excepção de um dos *Asterisk* já existente na rede empresarial. Neste caso, teve de ser feito a interligação entre ambos os *Asterisk's*, pelo que foi configurado um *SIP Trunking* entre eles. Em seguida são descritas as instalações que foram feitas.

Asterisk

Asterisk é um *framework* de código aberto fornecido pela *Digium* [65], que facilita o desenvolvimento de aplicações de comunicação multimídia. Ele fornece interoperabilidade entre muitos protocolos

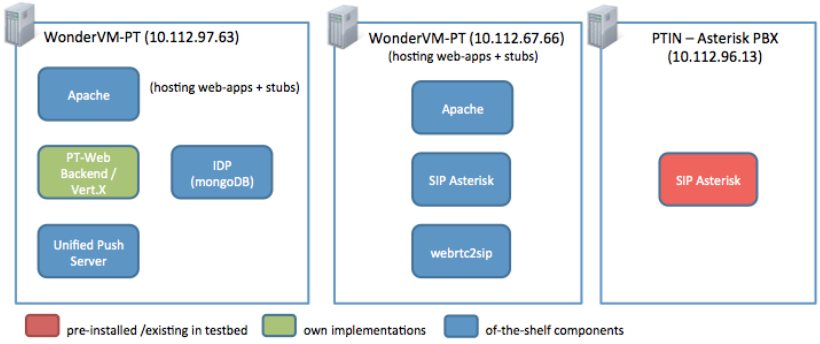


Figura 5.1: Componentes usadas na rede empresarial

de sinalização, bem como realiza a transcodificação de *media*, desde a versão 11 também suporta interoperabilidade com *endpoints WebRTC*. Isto significa *Asterisk* pode ser utilizado como uma porta de entrada para permitir que o *WebRTC* comunique com serviços *SIP/IMS* ou a *PSTN*.

A rede empresarial já continha um *Asterisk*, mas de forma a isolar toda a realização de testes do *Asterisk* central, foi feita outra instalação numa *VM* requerida. Foram feitas todas as configurações precisas para a realização dos testes. Os ficheiros utilizados para essas configurações foram o *sip.conf* e *extensions.conf*. No ficheiro *sip.conf*, são configuradas as contas dos utilizadores, bem como os *IP* e portas usadas para o serviço, é preciso ainda indicar os *codecs* que cada utilizador irá suportar. Neste ficheiro é ainda configurado o *SIP Trunking* como será apresentado de seguida. No ficheiro *extensions.conf* é configurado as extensões que serão usadas.

SIP Trunking

De forma a possibilitar uma comunicação com o *Asterisk* da PT Inovação e Sistemas, foi necessário configurar um *SIP Trunking* entre o *Asterisk* criado para realização de testes e o existente na empresa. Este mecanismo serve para encaminhar toda a comunicação existente entre ambos se assim for o caso. Para isto, é necessário fazer algumas configurações extras em ambos os *Asterisk's*. No ficheiro *sip.conf* foi necessário indicar qual o *IP* em que se encontra o *Asterisk*, bem como a chave de autenticação partilhada entre ambos. No código em baixo, é possível ver o código configurado no *Asterisk* instalado para testes. Outro ficheiro também que é necessário configurar é o *extensions.conf*. Neste ficheiro é configurado quais as comunicações que deverão ser encaminhadas para o *Asterisk* da empresa, isto será feito através de um prefixo. Neste caso queremos que sejam transferidas todas as chamadas cujo os números comecem por zero(0) que é para o caso do número ser referente a *PSTN*, e caso seja começado por 34 também por que é para as extensões internas da PT Inovação e Sistemas.

```
sip.conf
[freepbx]
type=peer
username=freepbx-user
fromuser=freepbx-user
secret=123123
host=10.23.96.13
allow=all
qualify=yes
context=webrtc
```

```
extensions.conf
exten => _0.,1,Dial(SIP/${EXTEN}@freepbx)
exten => _34.,1,Dial(SIP/${EXTEN}@freepbx)
```

Webrtc2sip

Webrtc2sip é uma *gateway* para *SIP* desenvolvida pela *Doubango* que permite a interoperabilidade entre a comunicação *web* e *SIP*. Para ser possível haver comunicação entre os diferentes domínios (*web* e SIP), foi preciso configurar uma *gateway*, a escolha foi a *webrtc2sip* [66]. Na instalação desta componente foi necessário alguma atenção pois esta tem diversos requisitos, um dos principais é de necessitar da versão do *openssl-1.0.c*. Foi ainda necessário criar um certificado e uma chave para associar a mesma *gateway*.

5.1.1 TESTES REALIZADOS

De forma a provar a interoperabilidade com redes legadas, foram projetados diversos testes, em que são apresentadas as componentes e a sua interação. Na rede empresarial existe um *PBX* que permite interligar as comunicações com a *PSTN*, esta comunicação é feita a partir de um servidor *Asterisk*.

- Teste 1 - Chamada entre um utilizador registado no *Asterisk* a partir da aplicação, para um número da *PSTN* (Figura 5.2);
- Teste 2 - Chamada entre um utilizador registado no domínio *web*, para um número da *PSTN* (Figura 5.6);
- Teste 3 - Chamada entre um utilizador registado no domínio *web*, para um utilizador registado num *softphone(Linphone)* (Figura 5.10);
- Teste 4 - Chamada entre um utilizador registado no domínio *SIP*, para um utilizador registado num *softphone(Linphone)* (Figura 5.14);

TESTE 1

Neste caso foi testada a interoperabilidade entre um utilizador registado no *Asterisk*, que comunica a partir de *WebSockets* com a *gateway*. Este utilizador inicialmente está ligado ao seu *MessagingServer*, com quem irá trocar todas as mensagens de registo. Após o utilizador estar registado, ele poderá iniciar uma conversação com um utilizador de qualquer domínio. Ao iniciar uma chamada a aplicação irá verificar se é um número da *PSTN* e caso seja ele irá encaminhar a chamada para a *PSTN*. Antes de a chamada estar estabelecida são trocadas as mensagens (*SIP*) para que sessão seja estabelecida (por exemplo *Invite,OK, Ack*), estas mensagens são enviadas para a *gateway*, que as irá traduzir em mensagens *SIP*. O *Asterisk (10.112.67.66)* ao receber as mensagens verifica que o número começa por zero (0) e encaminha todas as mensagens para o *Asterisk (10.112.96.13)*. O número zero(0) antes do número da *PSTN*, serve para o *Asterisk* saiba distinguir as chamadas que são feitas para as suas extensões das chamadas que são feitas para a *PSTN*. Este encaminhamento de mensagens entre os dois *Asterisk's* é feito devido a ter sido configurado um *SIP Trunking*, e desta forma é possível a

comunicação entre ambos. O *Asterisk (10.112.96.63)* ao receber as mensagens, encaminha-as para o *PBX*, após as mensagens estarem trocadas a sessão está estabelecida entre o utilizador *WebRTC* e o utilizador da *PSTN*.

De notar que toda a comunicação *RTP* é trocada a partir da *gateway*. Isto deve-se ao facto de a *gateway* actuar não só como *media coder*, mas também com *media relay* desta forma todo o tráfego *RTP* irá passar pela mesma. As chamadas no sentido inverso também foram testadas com sucesso, havendo para isso um requisito que é o utilizador tem de estar registado, para que possa receber todas as mensagens. Outro teste semelhante foi também efetuado para testar chamadas para as extensões da rede interna da empresa. O método de funcionamento é em tudo semelhante ao descrito até aqui ficando apenas a diferença no dispositivo final; as chamadas também foram testadas com sucesso em ambos os sentidos.

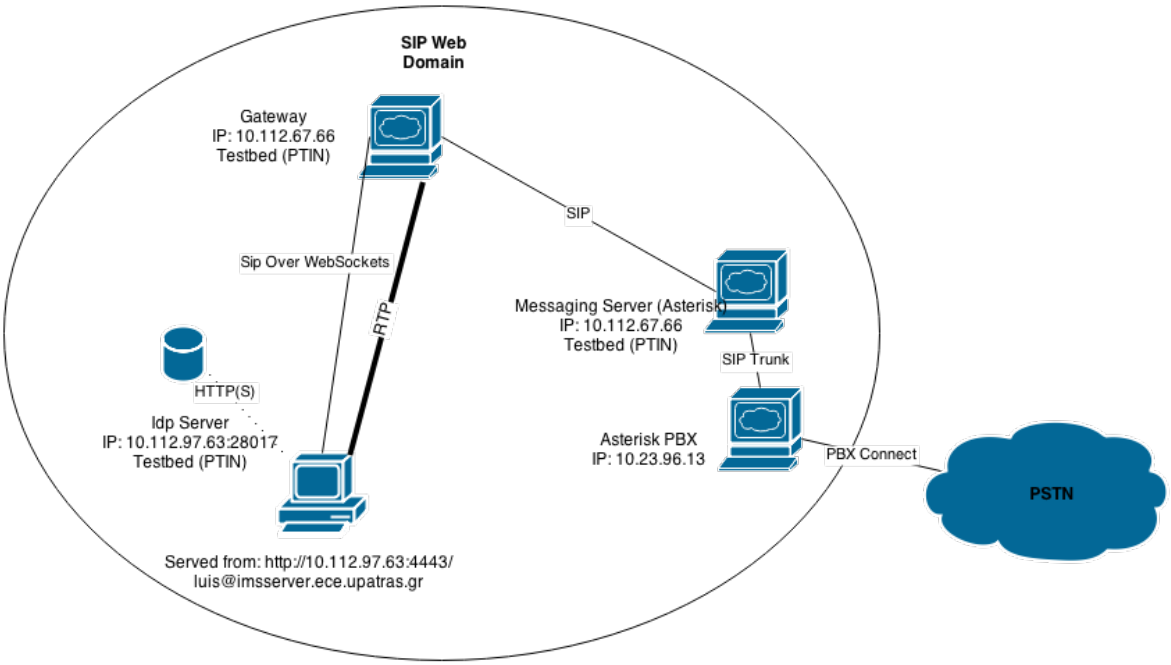


Figura 5.2: Chamada entre um utilizador *SIP* e um número na *PSTN*.

Teste de qualidade

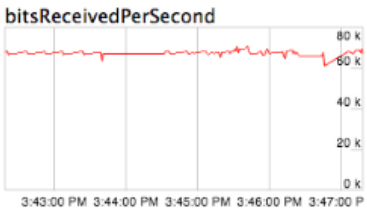


Figura 5.3: *Bits* recebidos

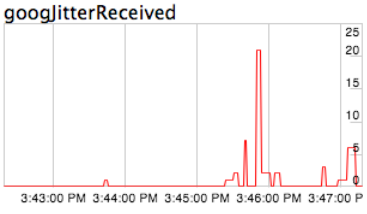


Figura 5.4: *Jitter*

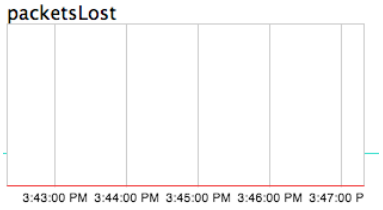


Figura 5.5: Pacotes perdidos

TESTE 2

De forma a testar também a interoperabilidade entre domínios, foi projetado também um teste de forma provar o correto funcionamento. Neste caso o utilizador esta registado no seu *MessagingServer* que é do domínio *web*. Este domínio esta ligado a um servidor *Vertx* comunicando por *WebSockets*. Ao querer iniciar uma conversação para um número da *PSTN*, a aplicação irá verificar que o utilizador está a querer falar com um utilizador de um domínio diferente do seu. Ao contactar o *Identity Provider(IDP)* ele irá obter informação sobre o domínio com que quere falar, desta forma irá fazer *download* do *MessagingStub* associado ao domínio *SIP*, e irá instancia-lo no seu *web browser*. Após estar feito este processo ele irá ligar-se ao *MensagingServer* do domínio *SIP*, e irá estar ligado ao *Asterisk*. A partir daí a sessão é estabelecida de igual forma ao exemplo anterior. As chamadas de um utilizador do domínio *PSTN* para um domínio *web* não são possíveis uma vez que o utilizador do domínio *PSTN* não tem forma de comunicar com esse domínio.

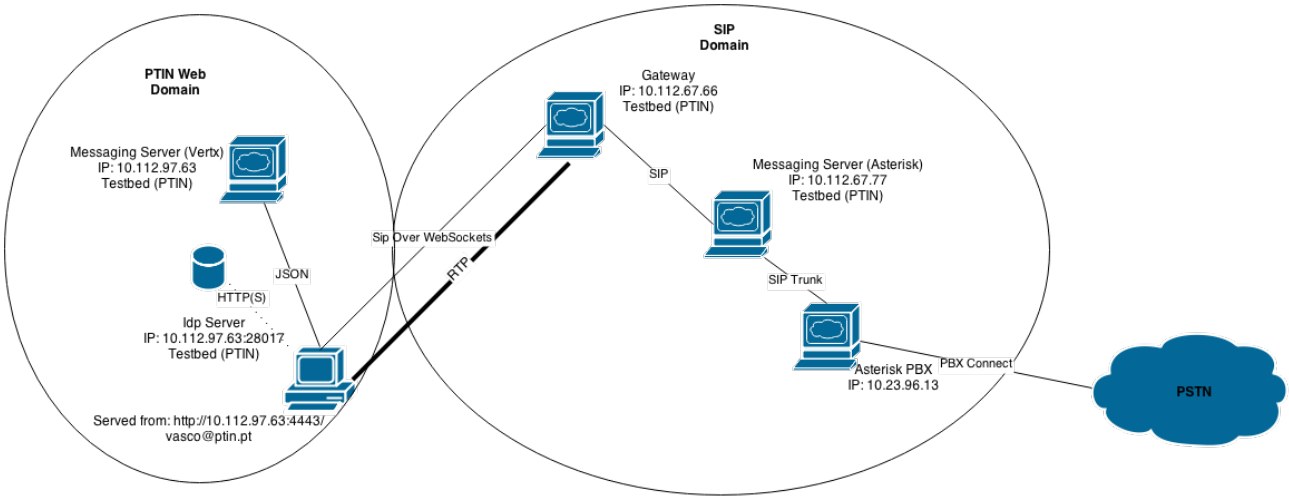


Figura 5.6: Chamada entre um utilizador do domínio *web* para um número na *PSTN*.

Teste de qualidade

Para verificar a qualidade de áudio/vídeo na comunicação foi usado o mecanismo anteriormente descrito. A partir das imagens 5.7, 5.8 e 5.9 as conclusões que se podem retirar da qualidade da comunicação são semelhantes as do teste anterior, embora neste caso exista uma maior variação do *jitter*.

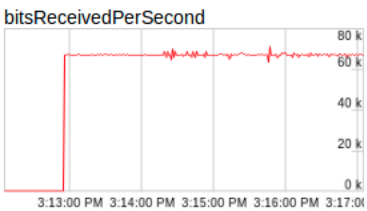


Figura 5.7: *Bits* recebidos

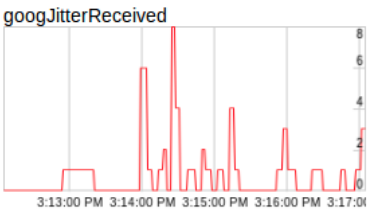


Figura 5.8: *Jitter*

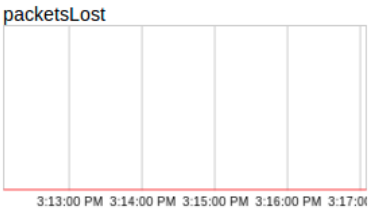


Figura 5.9: Pacotes perdidos

TESTE 3

Para demonstrar também a comunicação com *softphones*, foi desenhado um cenário de testes em que é usado o *Linphone* como *softphone* que irá comunicar com um utilizador registado no domínio *web*. O mecanismo de estabelecimento da sessão é em tudo similar ao mecanismo apresentado anteriormente, ficando apenas a diferença para a aplicação que o utilizador final usa. Neste caso é possível, testar a comunicação áudio/vídeo, entre ambos. A comunicação áudio/vídeo é possibilitada a partir da *gateway*, em que é necessário ter na configuração a *flag* de *mediaCoder* activo, isto porque os *softphones* podem não usar os mesmos *codecs* e com esta *flag* irá ser permitido que essa comunicação seja feita.

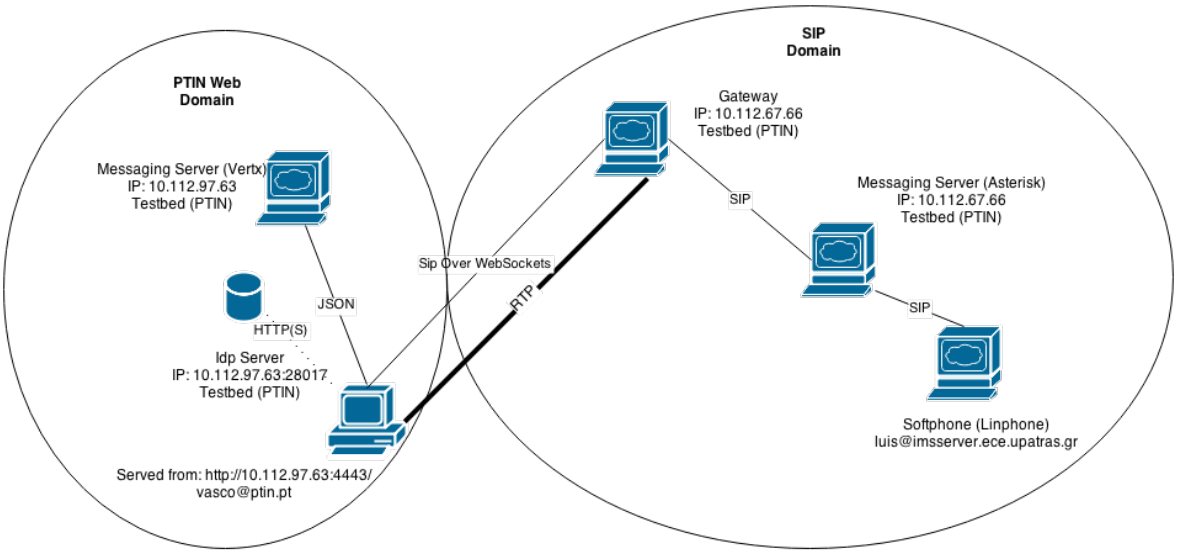


Figura 5.10: Chamada entre um utilizador do domínio *web* para um *Softphone(Linphone)*.

Teste de qualidade

Para verificar a qualidade de áudio/vídeo na comunicação foi usado o mecanismo anteriormente descrito. A partir das imagens 5.11, 5.12 e 5.13 as conclusões que se podem retirar da qualidade da comunicação neste caso é que a qualidade da comunicação tem várias interferências pelo que a quantidade de pacotes perdidos é grande.

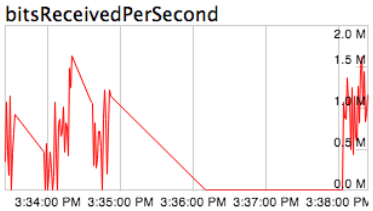


Figura 5.11: *Bits* recebidos

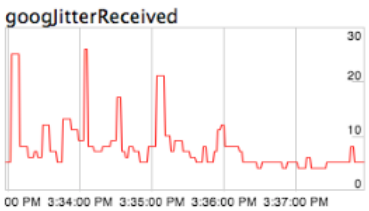


Figura 5.12: *Jitter*

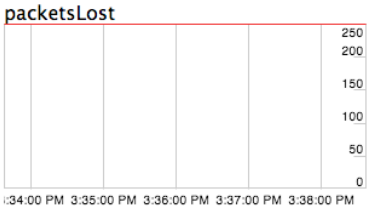


Figura 5.13: Pacotes perdidos

TESTE 4

Neste caso, foi realizado um teste de forma a demonstrar o funcionamento de intra-domínio, ou seja, entre utilizadores do mesmo domínio. O estabelecimento da sessão entre os dois utilizadores é

feito de uma forma simples uma vez que os utilizadores já estão registados ao *Asterisk*, desta forma qualquer utilizador está apto a trocar as mensagens de início de sessão através da *gateway*. Após estar estabelecida a sessão todo o tráfego *RTP* é trocado a partir da *gateway*.

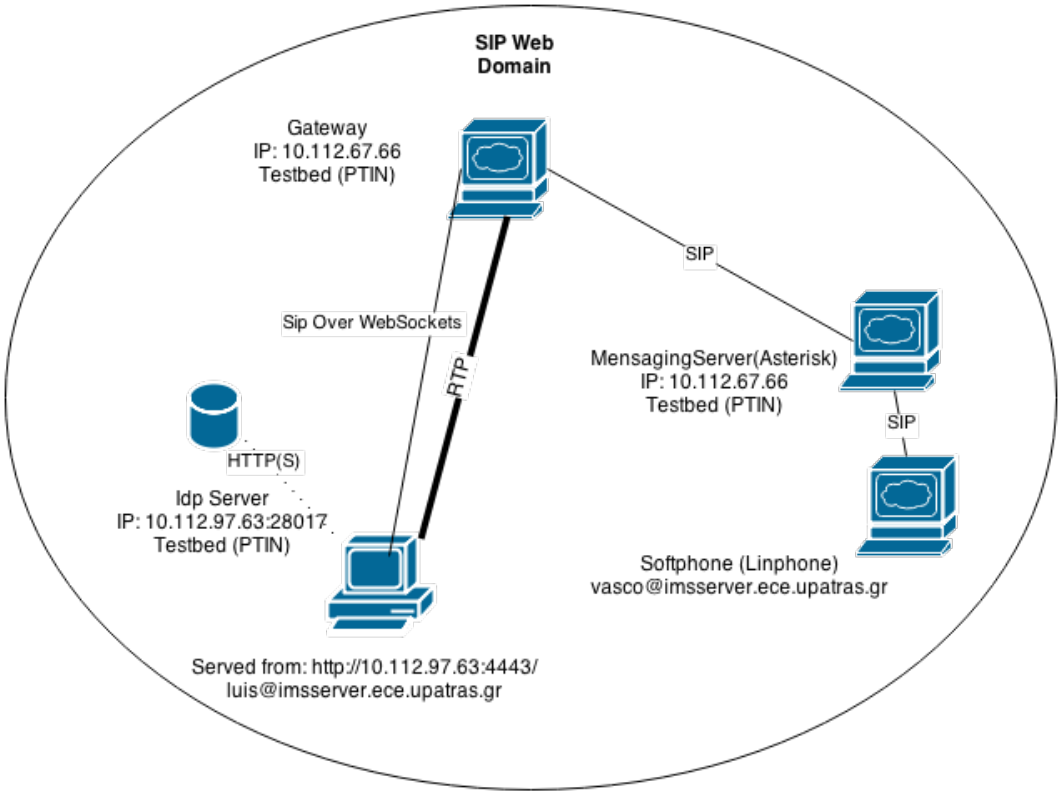


Figura 5.14: Chamada entre um utilizador do domínio *SIP* para um *Softphone(Linphone)*.

Teste de qualidade

Para verificar a qualidade de áudio/vídeo na comunicação foi usado o mecanismo anteriormente descrito. A partir das imagens 5.15, 5.16 e 5.17 as conclusões que se podem retirar da qualidade da comunicação se vai degradando, pois o número de pacotes perdidos vai aumentando. Quanto ao *jitter* este também tem grandes variações. A qualidade da comunicação neste caso é má devido ao facto de todo o tráfego de *media* ter de passar pela *gateway*.

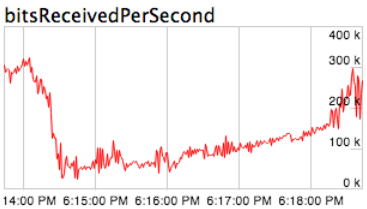


Figura 5.15: *Bits* recebidos

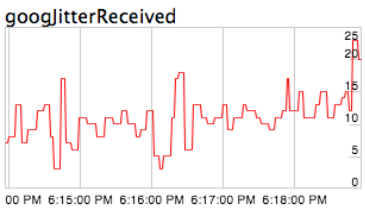


Figura 5.16: *Jitter*

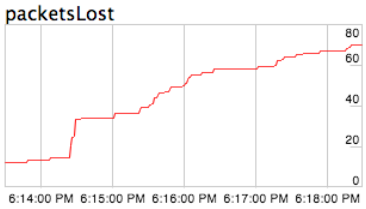


Figura 5.17: Pacotes perdidos

RESULTADOS

A tabela 5.1, pretende demonstrar os diversos casos de interação, usando os diferentes domínios. Na tabela é possível ver que foram efectuadas chamadas com sucesso em ambos os sentidos, com qualidades de áudio/vídeo. Só não se conseguiram estabelecer chamadas entre *PSTN* e o domínio *web*, uma vez que não foi encontrada forma de notificar o utilizador *web* que estaria a receber uma chamada.

Tabela 5.1: Resultados dos testes efetuados na rede empresarial

Domínio/Aplicação	**PT SIP (*web browser*)**	**PT SIP (*Linphone*)**	**PT Web (*web browser*)**	**PSTN**	**PSTN (extensão PTIN)**
PT SIP (*web browser*)	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo	Áudio	Áudio
PT SIP (*Linphone*)	Áudio/Vídeo	Áudio/Vídeo	-	Áudio	Áudio
PT Web (*web browser*)	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo	Áudio	Áudio
PSTN (extensão PTIN)	Áudio	Áudio	-	Áudio	Áudio
PSTN	Áudio	Áudio	-	Áudio	Áudio

5.2 INTEROPERABILIDADE COM REDE IMS

De forma a realizar testes na rede *IMS* foi necessário instalar algumas componentes nas diversas máquinas. O cenário principal foi fornecido pelo consórcio *OpenLab* [77] em que tinha como principais características:

- Prestação de um serviço *IMS*;
- Servidores/*VM'S* para extensões próprias;
- Acesso a rede para ambos e entre si;
- Servidores de *STUN/TURN*;
- *Gateway webrtc2sip*;
- Servidor de presença;
- Servidor de *XDMS*.

Algumas das componentes foram previamente fornecidas, outras tiveram de ser instaladas e configuradas como é possível verificar na figura 5.18. De seguida, irão ser explicadas as que foram usadas de forma directa nos testes realizados.

OpenIMS core

OpenIMS é uma implementação *open source*. Devido à sua natureza, como *software* de código aberto, *OpenIMS Core* é um *playground* bastante comum para os desenvolvimentos e testes *IMS*. Neste contexto esta plataforma é usada para suportar as aplicações do domínio *IMS*, que são usados para testes de interoperabilidade. Como é possível verificar na figura 5.19 o *OpenIMS* é consistuído entre outras componentes por funções de *Call Session Control Functions (CSCFs)*, os elementos de roteamento central para qualquer sinalização *IMS*, e um *Home Subscriber Server (HSS)* para gerir perfis de utilizador e regras de roteamento associados.

Servidores de STUN/TURN

O servidor de *TURN* foi também instalado nos servidores de forma a lidar com os problemas de *NAT* existentes na tecnologia *WebRTC* [30]. Este servidor fornece um mecanismo para retransmitir o tráfego entre dois *peers*, em que existe *NAT* entre ambos [79]. Foi também instalado um servidor de

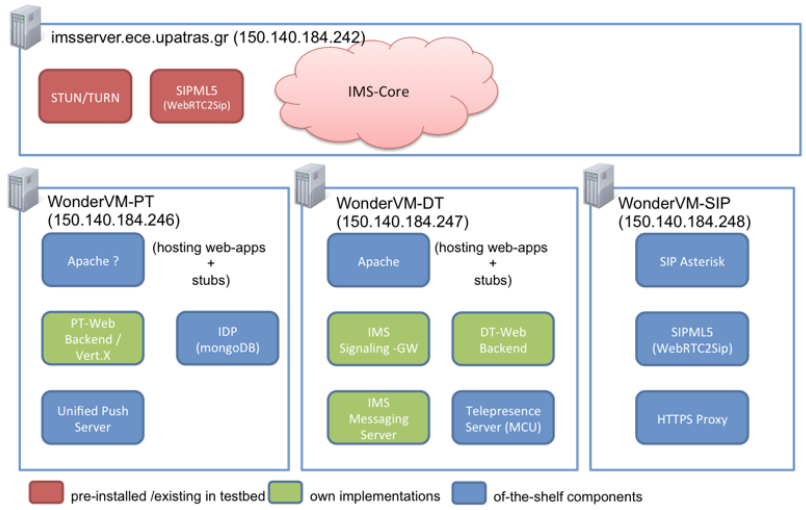


Figura 5.18: Todas as componentes relevantes instaladas na *testbed*.

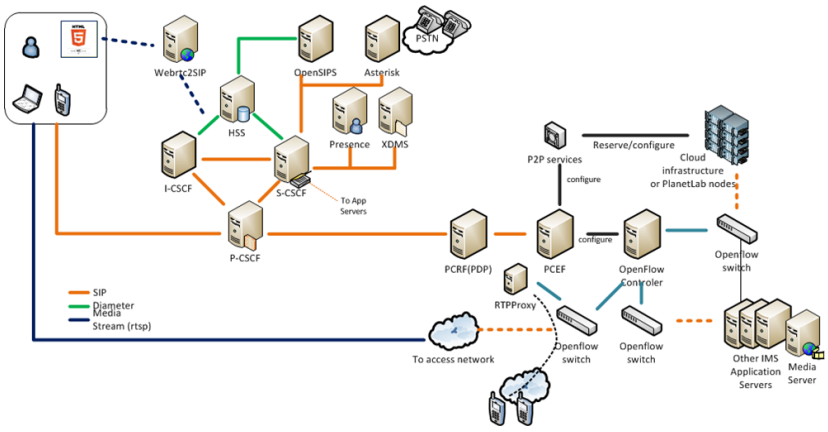


Figura 5.19: Estrutura da rede OpenIMS [78]

STUN, que permite que um *host* descubra os seus próprios endereços externos se ele estiver por de trás de *NAT* [29]. Ambos os servidores funcionam como base para o estabelecimento a intercomunicação do protocolo *ICE*, sendo estes fundamentais na tecnologia *WebRTC* [80].

5.2.1 TESTES REALIZADOS

- Teste 1 - Chamada entre um utilizador registado no domínio *SIP*, para um utilizador registado num *softphone(Linphone) usando uma rede IMS* (Figura 5.20);
- Teste 2 - Chamada entre um utilizador registado no domínio *SIP*, para um utilizador também do domínio *SIP* usando uma rede *IMS* (Figura 5.24);
- Teste 3 - Chamada entre um utilizador registado no domínio *SIP*, para um utilizador do domínio *DT IMS* (Figura 5.28);
- Teste 4 - Chamada entre um utilizador registado no domínio *SIP*, para um utilizador do domínio *DT Web* (Figura 5.32);

TESTE 1

De forma a demonstrar o correto funcionamento numa rede *IMS*, foram realizados alguns testes de forma a provar o mesmo. Neste caso demonstra uma chamada entre um utilizador registado na aplicação (*browser*), a comunicar com um utilizado que usa um *softphone(Linphone)* como aplicação. O utilizador que está a usar a aplicação *web*, ao iniciar a sessão irá invocar uma série de passos até que esteja registado no *core* da rede *IMS*. Começando por invocar o *Proxy Call Session Control Function(P-CSCF)*, este é o primeiro contacto com a rede *IMS*. De seguida irá ser invocado o *Interrogating Call Session Control Function (I-CSCF)*, que será o responsável por perguntar ao *HSS* se o utilizador que se está a fazer registo se encontra na sua base de dados local. Caso isso aconteça a sessão é registada no *Serving Call Session Control Function (S-CSCF)* passando toda a sinalização e controlo feita através desta componente. Após a sessão estar estabelecida e o utilizador registado devidamente na rede *IMS* a comunicação com um utilizador que esteja registado na mesma rede efectua-se trocando as mensagens de início de sessão (*Invite, Ok,Ack*). Esta troca de mensagens é permitida através do uso de uma *gateway*, uma vez que a aplicação *web* comunica através de *WebSockets* e o *Linphone* comunica por *SIP*. Depois de todas as mensagens estarem trocadas a sessão *RTP* está estabelecida e todas as mensagens *RTP* são enviadas com recurso a *gateway*. As chamadas neste caso foram estabelecidas com sucesso nos dois sentidos, mas verificou se no entanto alguns problemas de visualização do vídeo quando a velocidade de internet não era a melhor. Este facto deve-se a localização da *gateway* não ser mais próxima, uma vez que todo o tráfego *RTP* terá de passar pela *gateway* localizada na Universidade de Patras.

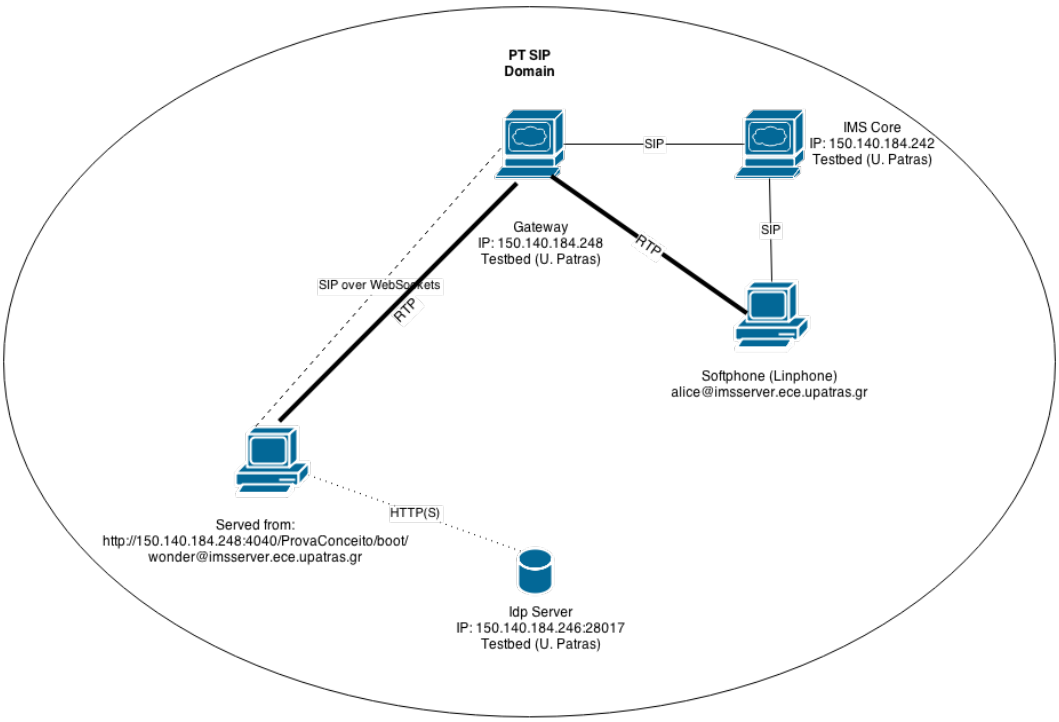


Figura 5.20: Chamada entre um utilizador do domínio *SIP* para um *Softphone(Linphone)* usando a rede *IMS*.

Teste de qualidade

Para verificar a qualidade de áudio/vídeo na comunicação foi usado o mecanismo anteriormente descrito. A partir das imagens 5.21, 5.22 e 5.23 as conclusões que se podem retirar da qualidade

da comunicação é que esta tem graves problemas, devido ao facto da *gateway* estar localizada na Universidade de Patras e todo o tráfego de *media* ter de passar pela mesma.

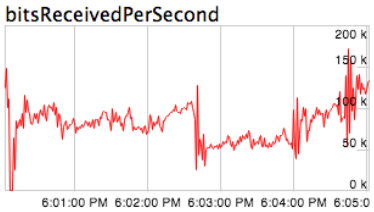


Figura 5.21: *Bits* recebidos

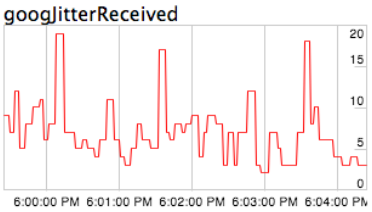


Figura 5.22: *Jitter*

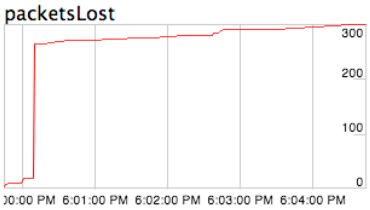


Figura 5.23: Pacotes perdidos

TESTE 2

Neste caso de teste, o principal objectivo é provar a interoperabilidade entre utilizadores do mesmo domínio usando uma rede *IMS* a partir da aplicação *web*. Desta forma todo o processo de registo é igual ao processo acima descrito. Depois de todo o processo de registo estar feito a comunicação entre ambos é feito sem recurso a *gateway*, uma vez que ambos os ambientes comunicam a partir de *WebSockets*, ou seja, não é necessário tradução.

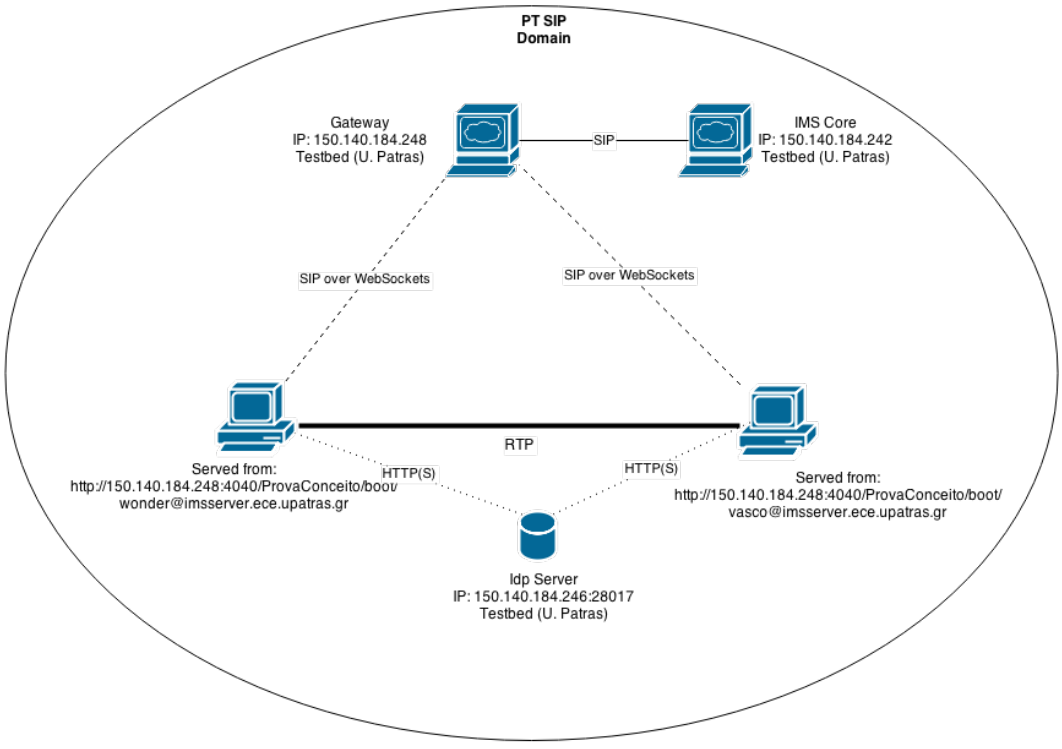


Figura 5.24: Chamada entre um utilizador do domínio *SIP* para o domínio *SIP* usando a rede *IMS*.

Teste de qualidade

Para verificar a qualidade de áudio/vídeo na comunicação foi usado o mecanismo anteriormente descrito. A partir das imagens 5.25, 5.26 e 5.27 as conclusões que se podem retirar da qualidade da

comunicação neste caso vai perdendo a qualidade, uma vez que os pacotes perdidos vão aumentando com a duração da chamada. No caso do *jitter* verifica-se alguma variação do mesmo.

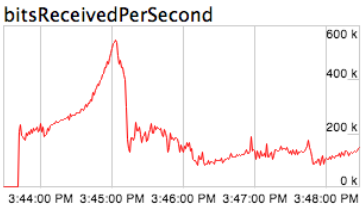


Figura 5.25: *Bits* recebidos

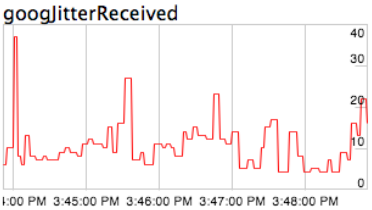


Figura 5.26: *Jitter*

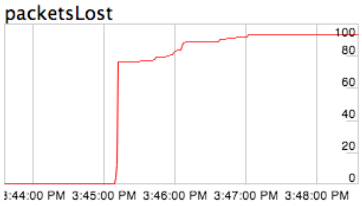


Figura 5.27: Pacotes perdidos

TESTE 3

O projeto *Wonder* tem como principal objetivo permitir a interoperabilidade entre os diferentes domínios. Todo o processo de registo é semelhante ao anteriormente descrito. Após estar o registo feito, o utilizador irá comunicar com o *Idp* e descarregar o *MessagingStub* associado ao domínio *DT IMS*, e irá instanciá-lo no seu *browser*, neste momento os dois utilizadores comunicam a partir do mesmo serviço. De seguida todo o tráfego *RTP* será trocado numa ligação *peer-to-peer* uma vez que os dois utilizadores comunicam por *websockets* não será necessário a *gateway* para encaminhar o tráfego.

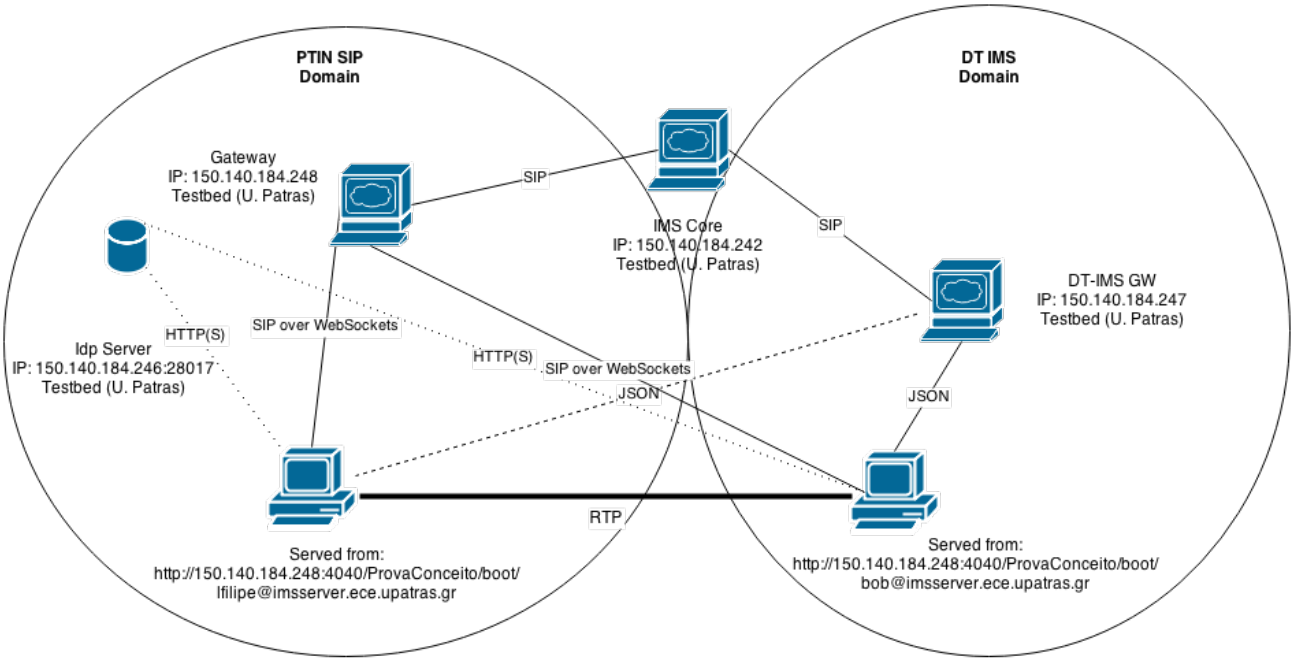


Figura 5.28: Chamada entre um utilizador do domínio *SIP* para o domínio *DT IMS* usando a rede *IMS*.

Teste de qualidade

Para verificar a qualidade de áudio/vídeo na comunicação foi usado o mecanismo anteriormente descrito. A partir das imagens 5.29, 5.30 e 5.31 as conclusões que se podem retirar da qualidade da comunicação são semelhantes as do teste anterior.

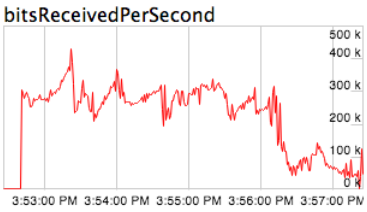


Figura 5.29: *Bits* recebidos

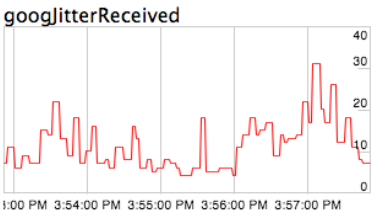


Figura 5.30: *Jitter*

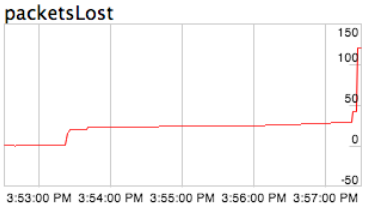


Figura 5.31: Pacotes perdidos

TESTE 4

Neste último teste realizado, o objetivo principal foi mostrar a interoperabilidade com o domínio *DT Web*. Este encontra-se hospedado num servidor *apache* e tem uma ligação *websockets* ao *node.js*. Todo o processo de registo dos utilizadores e estabelecimento da chamada, é semelhante ao descrito anteriormente. Após a sessão estar estabelecida o tráfego *RTP* também será encaminhado numa ligação *peer-to-peer*.

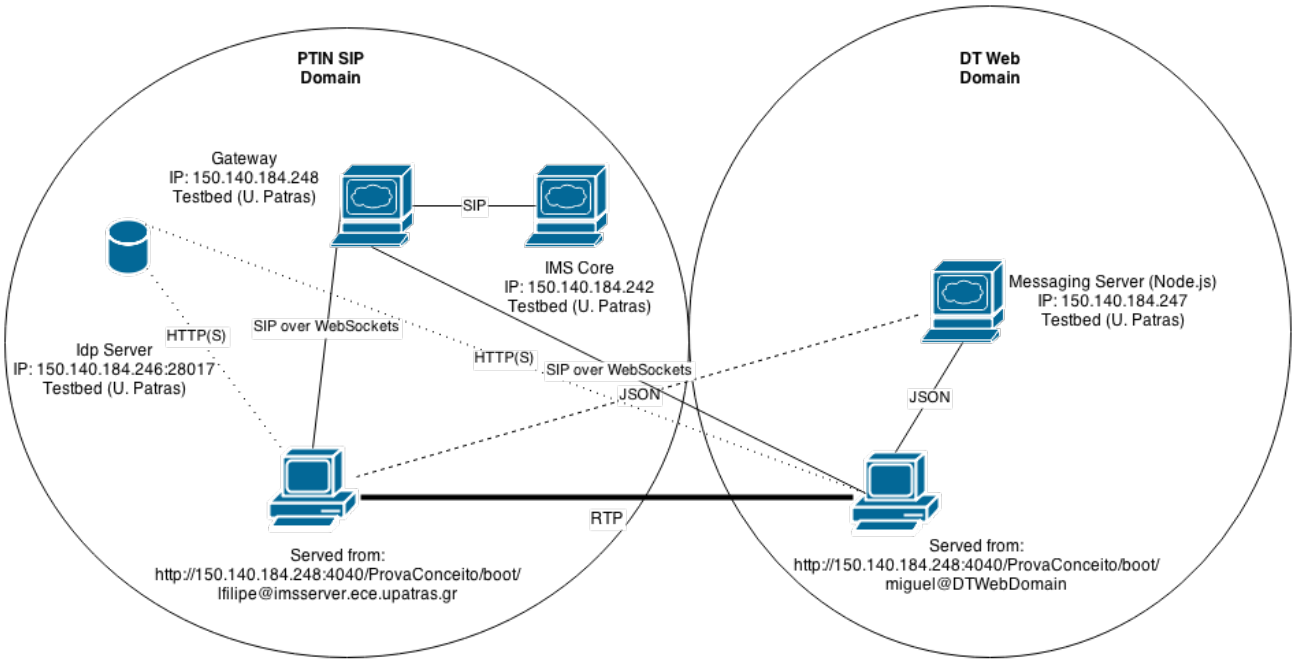


Figura 5.32: Chamada entre um utilizador do domínio *SIP* para o domínio *DT Web* usando a rede *IMS*.

Teste de qualidade

Para verificar a qualidade de áudio/vídeo na comunicação foi usado o mecanismo anteriormente descrito. A partir das imagens 5.33, 5.34 e 5.35 as conclusões que se podem retirar da qualidade da comunicação são iguais as dos testes anteriores.

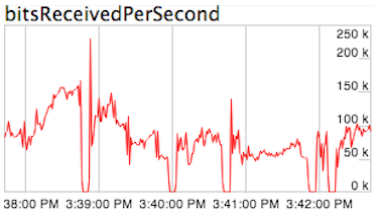


Figura 5.33: *Bits* recebidos

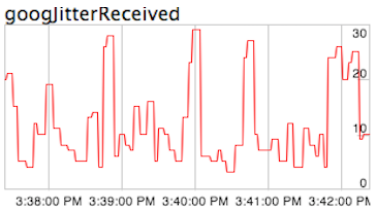


Figura 5.34: *Jitter*

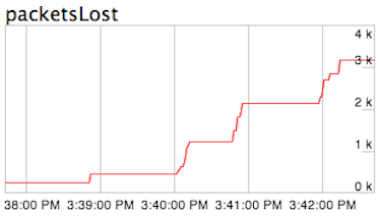


Figura 5.35: Pacotes perdidos

RESULTADOS

A tabela 5.2, pretende demonstrar os diversos casos de interação, usando os diferentes domínios. Na tabela é possível ver que foram efectuadas chamadas com sucesso em ambos os sentidos, com qualidades de áudio/vídeo. Foi possível ainda verificar que a qualidade da comunicação na rede *IMS* foi pior que a comunicação na rede empresarial.

Tabela 5.2: Resultados dos testes efetuados na rede *IMS*.

Domínio	**PT SIP**	**PT Web**	**DT IMS**	**DT Web**
PT SIP	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo
PT Web	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo
DT IMS	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo
DT Web	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo	Áudio/Vídeo

5.3 SUMÁRIO

Neste capítulo foram apresentadas os testes realizados, e as componentes usadas nos mesmos testes. Foi descrito as diferentes redes usadas para realização dos testes, sendo usado dois redes distintas, uma na rede empresarial e outra numa rede criada para o projeto (*testbed*). Ao longo deste capítulo foram descritos os diversos testes realizados para permitir mostrar as funcionalidades implementadas por está *API*.

Começando por se descrever os testes realizados na rede empresarial em que o objetivo era descrever todo o mecanismo envolvido para que sejam estabelecidas chamadas com redes legadas, para isso usando o *Asterisk* instalado na rede interna da empresa.

Em seguida foram apresentados os testes realizados na *testbed* fornecida para o projeto, em que foi necessário algumas reconfigurações, devido ao facto de a *gateway* inicial não permitir a função de *media coder*. Nestes testes foi possível verificar o correto funcionamento de toda a *API* numa rede *IMS*, para estes testes foi usado um *sofphone (Linphone)*.

Por fim foram apresentados os resultados dos testes realizados nas diversas redes, e foram tiradas as conclusões referentes aos mesmos.

capítulo 6

Conclusões e trabalho futuro

Neste capítulo será efetuada uma análise crítica e um resumo de todo o trabalho desenvolvido bem como conclusões sobre os resultados obtidos. Serão apresentadas algumas ideias para trabalho futuro, usando esta *API* e algumas funcionalidades que poderão ser implementadas.

6.1 trabalho desenvolvido

A tecnologia *WebRTC* veio revolucionar o meio das comunicações, tornando-se numa tecnologia com um crescimento exponencial. Desta forma as empresas de telecomunicações tendem a acompanhar esta evolução. A interoperabilidade entre *WebRTC* e as redes legadas é um grande desafio para as operadoras, e com este projeto esse desafio é superado.

Wonder é o projeto em que enquadrou inserido esta dissertação, cujo principal objetivo era criar uma *API* que fosse capaz de comunicar com diversos domínios usando a tecnologia *WebRTC*. Foram análisados domínios como: *PT Web* que era um domínio que usava como servidor o *Vertx* e estabelecia um canal de comunicação *WebSockets* com o mesmo; *DT Web* domínio que usava como servidor o *Apache* e utilizava um canal de comunicação *WebSockets* com o *Node.js*; *PT SIP* domínio que usava como servidor *Apache* e que permitiria estabelecer a comunicação entre utilizadores *SIP/IMS*; *DT IMS* este domínio usava como servidor o *Apache* e deveria estabelecer uma comunicação entre clientes *IMS*.

Esta dissertação focou-se mais no domínio *PT SIP*, que tinha como objetivo à interoperabilidade entre as redes legadas *IMS* e *PSTN*, mas surgiram alguns problemas de ordem técnica devido a rede usada de testes, em Patras, não permitir testar a interoperabilidade entre a *PSTN*, desta forma, foram usadas duas redes distintas para realização dos testes de forma a garantir a total interoperabilidade entre a rede *IMS* e *PSTN*. Nos testes, com a rede da *PSTN*, foi usada a rede interna da empresa, de forma que só é possível testar o cenário estando ligado por *VPN* à rede da empresa.

No início desta dissertação foi apresentado o estado de arte referente as temáticas e tecnologias envolvidas neste projeto. Sendo necessário um estudo de toda a parte de comunicação bem como os conceitos do *WebRTC*, este estudo teve de ser contínuo pois esta tecnologia ainda se encontra a

ser desenvolvimento e por vezes com actualizações do *web browsers*, algumas das funcionalidades são implementadas de forma diferente. Uma das dificuldades encontradas relativamente ao *WebRTC*, foi ainda a pouca documentação fornecida embora cada vez surja mais documentação associada. Existe ainda, poucos projetos que tenham estudado a interoperabilidade de *WebRTC* com as redes legadas.

Foi efetuada uma análise às *gateways* existentes, tendo em conta os *codecs* suportados, de forma a que não houvesse limitações ao nível das comunicações áudio/vídeo. Surgiram algumas dificuldades, na instalação e configuração da *gateway*, devido a ser um sistema complexo.

No desenvolvimento deste projeto tiveram de ser considerados diversos aspectos, devido a ser um sistema de alguma complexidade, que exigia o uso de diversas componentes, desde uma *gateway* que permitisse a comunicação entre um cliente *WebRTC* e um cliente *SIP*, a um servidor *SIP* (*Asterisk*) que permite a integração com a *PSTN*. Foram efetuados estudos e testes em torno destes aspectos de forma a avaliar qual a melhor solução para garantir total interoperabilidade entre os diferentes domínios.

Inicialmente, foram definidos como objetivos principais o estabelecimento de chamadas nos dois sentidos entre os diferentes domínios, ou seja, as aplicações *web* ligadas aos diferentes domínios comunicarem entre si. Seguindo-se o objetivo de integrar a solução construída com as redes *IMS* e *SIP*, de forma a verificar a sua comunicação com as redes legadas.

Todo o processo de validação da *API* foi um processo contínuo, sendo elaborados diversos testes para garantir a total interoperabilidade. No âmbito desta dissertação foram elaborados testes em duas redes diferentes pelo motivo anteriormente referido.

Numa visão global todos os testes foram muito bem sucedidos utilizando o *Chrome*. Com o *Firefox*, foram encontrados alguns problemas de compatibilidade com a *API* desenvolvida. Tal deveu-se devido ao facto da *API* usar algumas funções que ainda não estão implementadas no *Firefox*.

Quanto a qualidade de áudio/vídeo na comunicação verificou-se, que a distância entre a aplicação e a *gateway* tem interferência na comunicação. Quanto maior for a distância para a *gateway* maior vai ser a taxa de perda de pacotes. Por outro lado a qualidade da ligação a *Internet* também tem interferência na qualidade da comunicação.

Com isto pode se concluir, que todo o trabalho realizado neste projeto poderá ser utilizado por diversas pessoas, de forma a tornar o desenvolvimento de aplicações *WebRTC* mais simples, tendo uma maior abstração na concretização das mesmas. Quanto à interoperabilidade entre redes legadas, a tecnologia revelou já conseguir estabelecer chamadas entre os dois *domínios*, tendo para isso a necessidade de usar uma *gateway*. Foi observado ainda que a ligação a *Internet* tem uma grande influência na qualidade de áudio/vídeo recebido.

6.2 TRABALHO FUTURO

O trabalho futuro deste projeto terá de ser um processo contínuo visto que o *WebRTC* está em constante evolução, acompanhando, assim, todo o processo evolutivo desta tecnologia.

Teria de ser também estudado a forma de garantir interoperabilidade entre os diferentes *web browsers* de forma a não haver restrições no uso da aplicação.

Como este projeto foi possível criar uma API que certamente irá ser utilizada por diversos programadores de aplicações *web*. No meio empresarial (PT Inovação e Sistemas) já existem algumas aplicações que usam esta tecnologia, da forma, que se pretende num futuro próximo integrar a *API*

desenvolvida nas aplicações existentes e usá-la nas que surgirão eventualmente, criando assim uma maior abstração dos conceitos, melhorando assim a facilidade com que é construída a solução.

Referências

[1] About.com, *Timeline of electronics.* [Online; acedido Janeiro 2014], 2014. endereço: http://inventors.about.com/od/timelines/a/electricity_timeline.htm.

[2] ——, *A brief history of telecommunications.* [Online; acedido Janeiro 2014], 2014. endereço: http://www.cellphones.ca/features/brief-history-telecommunications/.

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

[4] D. Analysis, *Webrtc market status & forecasts:* [Online; acedido Janeiro 2014], 2014. endereço: http://www.disruptive-analysis.com/webrtc.htm.

[5] L. Oliveira, M. Seijo, V. Amaral, P. Chainho, S. Druesedow e K. Hansge., *Public webrtc framework developed in the wonder project featuring seamless inter-domain interoperability*, Marth 2014. endereço: https://github.com/hypercomm/wonder/.

[6] L. Oliveira e V. Amaral, *Web real time communication: (the 8th) wonder*, [Online; acedido Abril 2014], 2014. endereço: https://codebits.eu/intra/s/session/381.

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

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

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

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

[11] 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.

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

[13] *Webrtc codec and media processing requirements*, 12 March 2012. endereço: http://tools.ietf.org/html/draft-cbran-rtcweb-codec-02.

[14] A. Grange, H. Alvestrand e Google, *A vp9 bitstream overview*, [Online; acedido Março 2014], February 18, 2013. endereço: http://tools.ietf.org/html/draft-grange-vp9-bitstream-00.

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

[16] X.-0. Jingle, *Xmpp.org*, [Online; acedido Janeiro 2014]. endereço: http://xmpp.org/extensions/xep-0166.html.

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

[18] R. Jesup, Mozilla, S. Loreto, Ericsson, M. Tuexen e M. U. of Appl. Sciences, *Rtcweb data channels*, [Online; acedido Janeiro-Abril 2014], February 25, 2013. endereço: http://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-04.

[19] A. S. Ahson e M. Ilyas, *High Performance Browser Networking What every web developer should know about networking and web performance*, 1ª ed. O'Reilly Media, September 2013, Capítulo 18.

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

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

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

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

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

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

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

[27] J. Uberti, Google, C. Jennings e Cisco, *Javascript session establishment protocol*, February 13, 2014. endereço: http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-06.

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

[29] J. Rosenberg, Cisco, R. Mahy, P. Matthews, Unaffiliated e D. Wing, *Session traversal utilities for nat (stun)*, [Online; acedido Janeiro-Março 2014], October 2008. endereço: http://tools.ietf.org/html/rfc5389.

[30] *Free open source implementation of turn and stun server*, 2014. endereço: https://code.google.com/p/rfc5766-turn-server/.

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

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

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

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

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

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

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

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

[39] *Imsdroid - high quality video sip/ims client for google android*, January 2014. endereço: https://code.google.com/p/imsdroid.

[40] *Imsdroid - high quality video sip/ims client for google android*, January 2014. endereço: https://code.google.com/p/imsdroid/wiki/.

[41] *Idoubs - sip/ims videophone for ios (iphone, ipad and ipod touch) and mac os x*, 2014. endereço: https://code.google.com/p/idoubs/.

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

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

[44] *Sofia-sip library.* endereço: http://sofia-sip.sourceforge.net/.

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

[46] Dougango, *Class sipml.stack.* [Online; acedido Janeiro-Abril 2014], 2014. endereço: http://sipml5.org/docgen/symbols/SIPml.Stack.html.

[47] *Android-rcs-ims-stack - rcs-e stack for android platform*, 2014. endereço: http://www.findbestopensource.com/product/android-rcs-ims-stack.

[48] *Qoffeesip homepage home*, 2014. endereço: https://quobis.atlassian.net/wiki/display/QoffeeSIP/QoffeeSIP+Homepage+Home.

[49] *License*, 2014. endereço: http://sipjs.com/license/.

[50] *Development guides*, 2014. endereço: http://sipjs.com/guides/.

[51] *Pjsip version 2.2.1 is released!*, 2014. endereço: http://www.pjsip.org/.

[52] Tropo, *Phono.* [Online; acedido Janeiro 2014], 2014. endereço: http://phono.com/.

[53] PeerJSteam, *Peerjs simplifies webrtc peer-to-peer data, video, and audio calls.* [Online; acedido Fevereiro-Março 2014], Marth 2014. endereço: http://peerjs.com/.

[54] andYet, *Simplewebrtc - you can build cool stuff with webrtc in five minutes.* [Online; acedido Fevereiro-Março 2014], Marth 2014. endereço: http://simplewebrtc.com/.

[55] M. Khan, *Webrtc experiments & demos*, [Online; acedido Janeiro-Março 2014], Marth 2014. endereço: https://www.webrtc-experiment.com/.

[56] ——, *Webrtc experiments & demos*, [Online; acedido Janeiro-Março 2014], Marth 2014. endereço: https://github.com/muaz-khan/WebRTC-Experiment/tree/master/RecordRTC.

[57] ——, *Webrtc experiments & demos*, [Online; acedido Janeiro-Março 2014], Marth 2014. endereço: http://www.rtcmulticonnection.org/docs/.

[58] ——, *Webrtc experiments & demos*, [Online; acedido Janeiro-Março 2014], Marth 2014. endereço: https://github.com/muaz-khan/WebRTC-Experiment/tree/master/Translator.js.

[59] ——, *Webrtc experiments & demos*, [Online; acedido Janeiro-Março 2014], Marth 2014. endereço: https://github.com/muaz-khan/WebRTC-Experiment/tree/master/DataChannel.

[60] ——, *Webrtc experiments & demos*, [Online; acedido Janeiro-Março 2014], Marth 2014. endereço: https://github.com/muaz-khan/WebRTC-Experiment/tree/master/DetectRTC.

[61] ——, *Webrtc experiments & demos*, [Online; acedido Janeiro-Março 2014], Marth 2014. endereço: https://github.com/muaz-khan/WebRTC-Experiment/tree/master/getMediaElement.

[62] ——, *Webrtc experiments & demos*, [Online; acedido Janeiro-Março 2014], Marth 2014. endereço: https://github.com/muaz-khan/WebRTC-Experiment/tree/master/ffmpeg.

[63] *Adapter.js*, [Online; acedido Janeiro 2014], Marth 2014. endereço: https://github.com/GoogleChrome/webrtc/blob/master/samples/web/js/adapter.js.

[64] Google, *Interop notes.* [Online; acedido Janeiro 2014], Marth 2014. endereço: http://www.webrtc.org/interop.

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

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

[67] Meetecho, *Janus: the general purpose webrtc gateway*, [Online; acedido Janeiro 2014], Marth 2014. endereço: http://janus.conf.meetecho.com/index.html.

[68] Joyent, *Node.js*, [Online; acedido Janeiro 2014], Marth 2014. endereço: http://nodejs.org/.

[69] Wonder, *Welcome to the wonder wiki!*, [Online; acedido Junho-Julho 2014], 2014. endereço: https://github.com/hypercomm/wonder/wiki/.

[70] L. Oliveira, M. Seijo, V. Amaral, P. Chainho, S. Druesedow e K. Hansge., *Wonder*, [Online; acedido Maio 2014], Marth 2014. endereço: https://github.com/hypercomm/wonder/tree/master/docs/images.

[71] Vertx, *Vertx*, [Online; acedido Janeiro 2014], Marth 2014. endereço: http://vertx.io/.

[72] *The mongodb 2.6 manual*, 2014. endereço: http://docs.mongodb.org/manual/.

[73] *Introducing json*, 2014. endereço: http://www.json.org/.

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

[75] Doubango, *Namespace sipml*, [Online; acedido Janeiro-Abril 2014], Marth 2014. endereço: http://sipml5.org/docgen/symbols/SIPml.html.

[76] wonder, *Pt sip wonder aplication*, [Online; acedido Janeiro-Julho 2014], Marth 2014. endereço: http://150.140.184.248:4040/ProvaConceito/boot/.

[77] OpenLab, *Node.js*, [Online; acedido Janeiro 2014], Marth 2014. endereço: https://www.onelab.eu/index.php/projects/openlab.html.

[78] *Osims - an open source ims experimentation platform*, 2014. endereço: http://nam.ece.upatras.gr/ppe/?q=node/2.

[79] *Traversal using relay nat (turn protocol)*, 2014. endereço: http://www.voip-info.org/wiki/view/TURN.

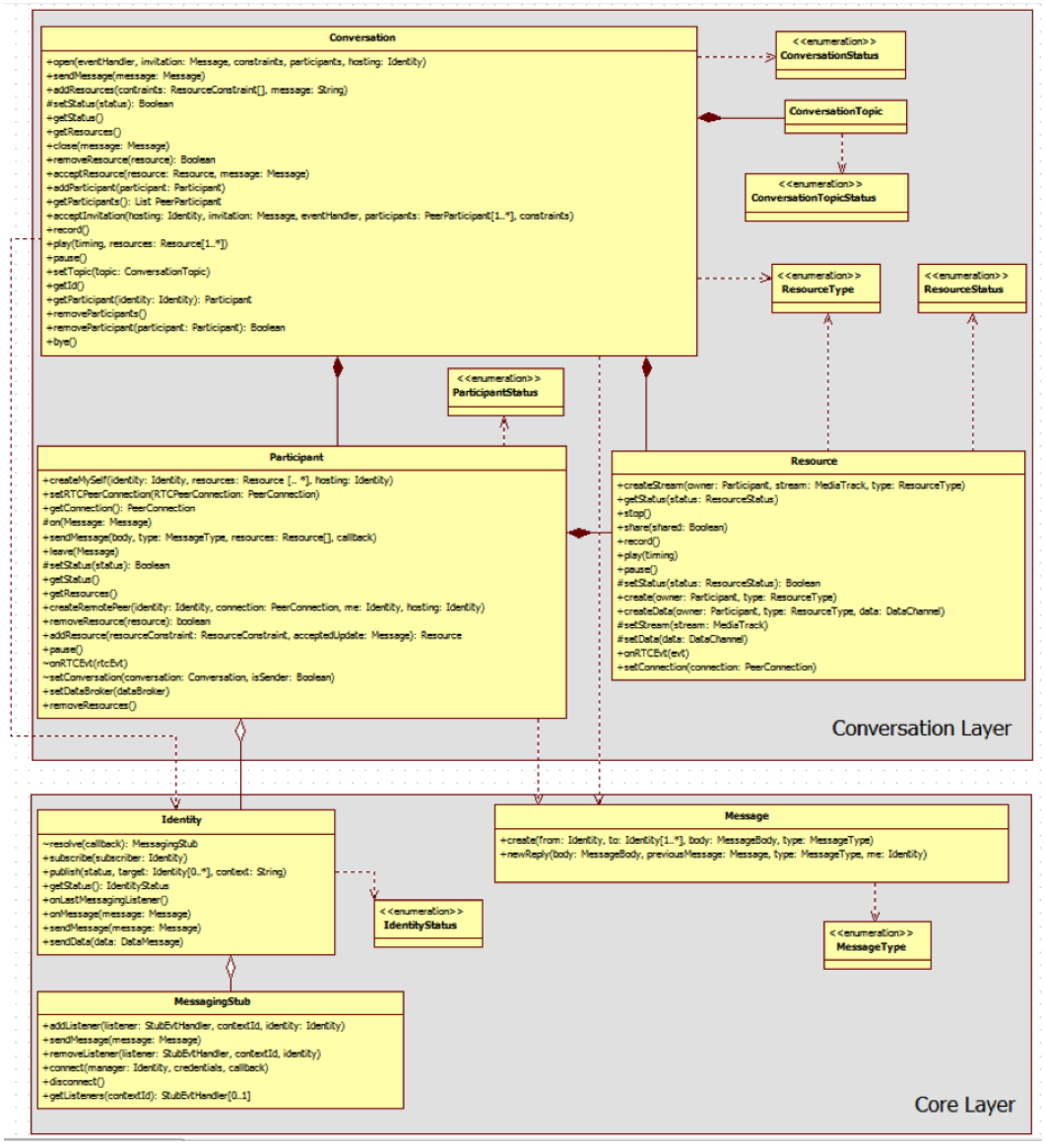
[80] *Ice.* endereço: http://www.voip-info.org/wiki/view/ICE.

Anexo A - Mensagem SDP de uma chamada entre WebRTC e a PSTN (Answer)

---

v=0
o=doubango 1983 678901 IN IP4 10.112.67.66
s=-
c=IN IP4 10.112.67.66
t=0 0 a=fingerprint:sha-256
D2:A2:1A:46:32:A0:0B:29:CF:8C:FC:F9:1C:D3:C6:95:B7:BF:9B:FD:3A:53:54:D7:2B:66:85:21:5D:BF:AF:70
a=setup:active
a=connection:new
m=audio 58904 RTP/SAVPF 0 8 126
c=IN IP4 10.112.67.66
a=ptime:20
a=minptime:1
a=maxptime:255
a=silenceSupp:off - - - -
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:126 telephone-event/8000/1
a=fmtp:126 0-16
a=sendrecv
a=rtcp-mux
a=ssrc:3496589836 cname:71b65daf0d662e69ec2113f5f98da55e
a=ssrc:3496589836 mslabel:6994f7d1-6ce9-4fbd-acfd-84e5131ca2e2
a=ssrc:3496589836 label:doubango@audio
a=ice-ufrag:lz1K2C7zt3nAjYo
a=ice-pwd:cUdKfZbJRe5pLzNKIjuFdT
a=candidate:DPkhbr281seomDN 1 udp 2130706431 10.112.67.66 58904 typ host

Anexo B - Descrição e dependências da API Wonder


Conversation
+open(eventHandler, invitation: Message, constraints, participants, hosting: Identity)
+sendMessage(message: Message)
+addResources(constraints: ResourceConstraint[], message: String)
#setStatus(status): Boolean
+getStatus()
+getResources()
+close(message: Message)
+removeResource(resource): Boolean
+acceptResource(resource: Resource, message: Message)
+addParticipant(participant: Participant)
+getParticipants(): List PeerParticipant
+acceptInvitation(hosting: Identity, invitation: Message, eventHandler, participants: PeerParticipant[1..*], constraints)
+record()
+play(timing, resources: Resource[1..*])
+pause()
+setTopic(topic: ConversationTopic)
+getId()
+getParticipant(identity: Identity): Participant
+removeParticipants()
+removeParticipant(participant: Participant): Boolean
+bye()
<<enumeration>>
ConversationStatus
ConversationTopic
<<enumeration>>
ConversationTopicStatus
<<enumeration>>
ResourceType
<<enumeration>>
ResourceStatus
<<enumeration>>
ParticipantStatus
Participant
+createMySelf(identity: Identity, resources: Resource [.. *], hosting: Identity)
+setRTCPeerConnection(RTCPeerConnection: PeerConnection)
+getConnection(): PeerConnection
#on(Message: Message)
+sendMessage(body, type: MessageType, resources: Resource[], callback)
+leave(Message)
#setStatus(status): Boolean
+getStatus()
+getResources()
+createRemotePeer(identity: Identity, connection: PeerConnection, me: Identity, hosting: Identity)
+removeResource(resource): boolean
+addResource(resourceConstraint: ResourceConstraint, acceptedUpdate: Message): Resource
+pause()
~onRTCEvt(rtcEvt)
~setConversation(conversation: Conversation, isSender: Boolean)
+setDataBroker(dataBroker)
+removeResources()
Resource
+createStream(owner: Participant, stream: MediaTrack, type: ResourceType)
+getStatus(status: ResourceStatus)
+stop()
+share(shared: Boolean)
+record()
+play(timing)
+pause()
#setStatus(status: ResourceStatus): Boolean
+create(owner: Participant, type: ResourceType)
+createData(owner: Participant, type: ResourceType, data: DataChannel)
#setStream(stream: MediaTrack)
#setData(data: DataChannel)
+onRTCEvt(evt)
+setConnection(connection: PeerConnection)
Conversation Layer
Identity
~resolve(callback): MessagingStub
+subscribe(subscriber: Identity)
+publish(status, target: Identity[0..*], context: String)
+getStatus(): IdentityStatus
+onLastMessagingListener()
+onMessage(message: Message)
+sendMessage(message: Message)
+sendData(data: DataMessage)
<<enumeration>>
IdentityStatus
Message
+create(from: Identity, to: Identity[1..*], body: MessageBody, type: MessageType)
+newReply(body: MessageBody, previousMessage: Message, type: MessageType, me: Identity)
<<enumeration>>
MessageType
MessagingStub
+addListener(listener: StubEvtHandler, contextId, identity: Identity)
+sendMessage(message: Message)
+removeListener(listener: StubEvtHandler, contextId, identity)
+connect(manager: Identity, credentials, callback)
+disconnect()
+getListeners(contextId): StubEvtHandler[0..1]
Core Layer

Anexo C - Mensagens recebidas numa chamada entre WebRTC e PSTN

Mensagem de Ok - registo com sucesso.

```
Via: SIP/2.0/UDP
    10.112.67.66:10060;rport=10060;received=10.112.67.66;
branch=z9hG4bKpVGpkciG9fxdCf3SOWFi75N8wvZ3hGmi
From: "paulo@ptin.pt"<sip:paulo_friend@imsserver.ece.upatras.gr>;
tag=MsrpAkCHfnn7iUf6WLzf
To:
    "paulo@ptin.pt"<sip:paulo_friend@imsserver.ece.upatras.gr>;tag=as790e1358
Contact: <sip:paulo_friend@10.112.67.66:10060;rtcweb-breaker=yes;
transport=udp;ws-src-ip=10.112.210.113;ws-src-port=47220;
ws-src-proto=ws>;expires=500
Call-ID: 28967875-99bf-46f4-71bb-8cffc4a6b2ea
CSeq: 26769 REGISTER
Expires: 500
Content-Length: 0
Via: SIP/2.0/TCP 10.112.210.113:47220;rport;
branch=z9hG4bKpVGpkciG9fxdCf3SOWFi75N8wvZ3hGmi;ws-hacked=WS
Server: Asterisk PBX 11.10.0
Allow:
    INVITE,ACK,CANCEL,OPTIONS,BYE,REFER,SUBSCRIBE,NOTIFY,INFO,PUBLISH,MESSAGE
Supported: replaces,timer
Date: 2 Jul 2014 14:12:9 GMT;2
```

Mensagem de Trying

```
Via: SIP/2.0/WS df7jal23ls0d.invalid;rport;
```

```
branch=z9hG4bKtcWc18p9TPaFifQnNo02wqzgqmgEXXcl
From: "paulo@ptin.pt"<sip:paulo_friend@imsserver.ece.upatras.gr>;
tag=EbMT9aAI1yrGNzak1Xlw
To: <sip:0934597664@imsserver.ece.upatras.gr>
Call-ID: 4d3d1f3f-0412-29bb-4aed-5368d6f1065f
CSeq: 2043 INVITE
Content-Length: 0
```

Mensagem de Ringing

```
Via: SIP/2.0/WS df7jal23ls0d.invalid;rport;
branch=z9hG4bKtcWc18p9TPaFifQnNo02wqzgqmgEXXcl
From: "paulo@ptin.pt"<sip:paulo_friend@imsserver.ece.upatras.gr>;
tag=EbMT9aAI1yrGNzak1Xlw
To: <sip:0934597664@imsserver.ece.upatras.gr>;
tag=1515205504
Contact: <sip:0934597664@10.112.67.66:10060;
transport=ws;ws-src-ip=10.112.210.113;ws-src-port=47220;ws-src-proto=ws>
Call-ID: 4d3d1f3f-0412-29bb-4aed-5368d6f1065f
CSeq: 2043 INVITE
Content-Length: 0
Allow:
    ACK,BYE,CANCEL,INVITE,MESSAGE,NOTIFY,OPTIONS,PRACK,REFER,UPDATE
```

Mensagem de OK - com o SDP (Answer)

```
Via: SIP/2.0/WS df7jal23ls0d.invalid;rport;
branch=z9hG4bKtcWc18p9TPaFifQnNo02wqzgqmgEXXcl
From: "paulo@ptin.pt"<sip:paulo_friend@imsserver.ece.upatras.gr>;
tag=EbMT9aAI1yrGNzak1Xlw
To: <sip:0934597664@imsserver.ece.upatras.gr>;tag=1515205504
Contact: <sip:0934597664@10.112.67.66:10060;
transport=ws;ws-src-ip=10.112.210.113;ws-src-port=47220;ws-src-proto=ws>
Call-ID: 4d3d1f3f-0412-29bb-4aed-5368d6f1065f
CSeq: 2043 INVITE
Content-Type: application/sdp
Content-Length: 795
Allow:
    ACK,BYE,CANCEL,INVITE,MESSAGE,NOTIFY,OPTIONS,PRACK,REFER,UPDATE

v=0
o=doubango 1983 678901 IN IP4 10.112.67.66
s=-
```

```
c=IN IP4 10.112.67.66
t=0 0
a=fingerprint:sha-256
   D2:A2:1A:46:32:A0:0B:29:CF:8C:FC:F9:1C:D3:C6:95:B7:BF:9B:FD
:3A:53:54:D7:2B:66:85:21:5D:BF:AF:70
a=setup:active
a=connection:new
m=audio 58904 UDP/TLS/RTP/SAVPF 0 8 126
c=IN IP4 10.112.67.66
a=ptime:20
a=minptime:1
a=maxptime:255
a=silenceSupp:off - - - -
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:126 telephone-event/8000/1
a=fmtp:126 0-16
a=sendrecv
a=rtcp-mux
a=ssrc:3496589836 cname:71b65daf0d662e69ec2113f5f98da55e
a=ssrc:3496589836 mslabel:6994f7d1-6ce9-4fbd-acfd-84e5131ca2e2
a=ssrc:3496589836 label:doubango@audio
a=ice-ufrag:lz1K2C7zt3nAjYo
a=ice-pwd:cUdKfZbJRe5pLzNKIjuFdT
a=candidate:DPkhbr281seomDN 1 udp 2130706431 10.112.67.66 58904 typ
   host
```

Anexo D - Alice invites Bob to a conversation hosted by her