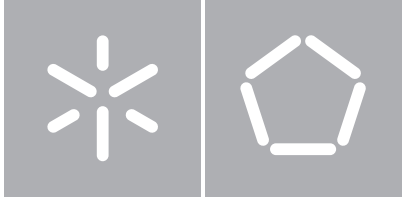
Universidade do Minho
Escola de Engenharia

Vasco Manuel de Frias Amaral

Framework e Cliente WebRTC

Outubro de 2013



Universidade do Minho
Escola de Engenharia
Departamento de Engenharia Informática

Vasco Manuel de Frias Amaral

Framework e Cliente WebRTC

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professora Drª Maria Solange Pires Rito Lima
Engª Telma Mota

Outubro de 2013

Framework e Cliente WebRTC

Vasco Manuel de Frias Amaral Amaral nº19821

Relatório de Dissertação submetido à Universidade do Minho, no âmbito do curso de Mestrado em Engenharia Informática, sob supervisão científica da Prof. Drª Maria Solange Pires Ferreira Rito Lima e da Engª. Telma Mota.

Universidade do Minho

Escola de Engenharia

Departamento de Informática

Outubro, 2013

Agradecimentos

Em primeiro lugar gostaria de agradecer à minha orientadora, Professora Doutora Solange Rito Lima, pelo apoio e orientação científica, assim como a disponibilidade que apresentou durante o decorrer deste trabalho. Como também à Eng^{a}. Telma Mota e ao Eng^{o}. Paulo Chainho pelo apoio, pela disponibilidade e também pela orientação que me deram na empresa PT Inovação, para que este projecto e dissertação fosse desenvolvido da melhor maneira possível.

Gostava de agradecer em especial à minha mãe, pai e irmãos por todo o apoio que me deram durante toda a minha vida académica e nesta fase final.

Agradeço ainda aos meus amigos que me acompanharam durante toda a infância e vida académica, como também aos meus colegas de curso e trabalho em especial ao Henrique Martins, Ricardo Macedo e Samuel Rodrigues pela ajuda e companheirismo que demonstraram durante este percurso académico.

Abstract

WebRTC is a standard technology which allows real-time communications between browsers, without installing additional plugins. In this way, for each device (computers, smartphones, etc.) with an installed browser, it is possible to perform peer-to-peer real-time communications natively, for instance, video and voice calls, chatting or instant messaging, file sharing and screen sharing.

This recent technology has grown exponentially both in implemented solutions and in browsers compatibility. WebRTC is therefore an evolutionary technology with a strong growth, where more solutions Over-The-Top (OTT) could appear and where the telecommunications operators could invest creating their own service solutions.

Facing the lack of standards regarding the communication between WebRTC endpoints, this project studies in depth the WebRTC technology in order to identify its potentiality and to assess in which way it could impact on the telecommunications world. This project also aims to create a framework that helps developing WebRTC applications and services at a higher level.

As proof-of-concept a WebRTC client is developed to allow testing the services implemented in the framework. The evaluation results address functionality tests, attesting that the implemented features of the framework work properly, and measure the CPU and memory consumption of WebRTC technology.

Resumo

WebRTC é uma tecnologia normalizada que permite a comunicação em tempo real entre browsers, sem a necessidade de instalar *plugins* adicionais. Desta forma, é possível a qualquer dispositivo (computadores, *smartphones*, etc.), que tenha instalado um browser, realizar comunicações em tempo real *peer-to-peer*, de uma forma nativa. Exemplo disso são as comunicações de voz, vídeo e também a possibilidade de falar por chat, partilhar ficheiros e partilhar ecrã.

Sendo uma tecnologia relativamente recente, o seu uso tem vindo a crescer exponencialmente, tanto a nível de soluções implementadas, como também a nível de compatibilidade de *web browsers*. Assim, a *WebRTC* torna-se uma tecnologia em forte crescimento e evolutiva, onde poderão surgir cada vez mais soluções de serviços *Over-The-Top* e os Operadores de Telecomunicações poderão investir, criando as suas próprias soluções e provocando um forte impacto ao nível de oferta de serviços.

Atendento a que ainda não está definida uma implementação normalizada para a comunicação entre *endpoints WebRTC*, nesta dissertação apresenta-se o resultado do estudo efetuado à tecnologia *WebRTC*, no sentido de identificar as suas potencialidades e o impacto que esta poderá ter no mundo das telecomunicações.Apresenta-se tambem a framework desenvolvida com o objetivo de tornar mais fácil a criação e implementação de serviços *WebRTC*, que servirá como uma solução de comunicação entre vários clientes.

Como prova de conceito, foi desenvolvida uma aplicação cliente, com a implementação de alguns serviços alvo. Para além dos testes de funcionamento dos serviços, foram realizadas análises de desempenho à utilização de CPU e de memória, no que diz respeito à tecnologia WebRTC.

Conteúdo

Agradecimentos **iii**

Abstract **v**

Resumo **vii**

Conteúdo **ix**

Lista de Figuras **xv**

Lista de Tabelas **xix**

Lista de Acrónimos **xxi**

1 Introdução **1**

1.1 Motivação e objetivos 2

1.2 Principais contribuições 4

1.3 Organização da dissertação 5

2 Conceitos Introdutórios **7**

2.1 *WebRTC* 7

2.1.1 WebRTC C++ API 8

2.1.2 *WebRTC API* 9

2.2 Compatibilidade 10
2.3 Protocolos *data channel* 11
2.3.1 *Stream Control Transmission Protocol (SCTP)* 12
2.3.2 *Datagram Transport Layer Security (DTLS)* 12
2.3.3 *User Datagram Protocol (UDP)* 12
2.3.4 *Secure Real-Time Protocol - Datagram Transport Layer Security* 13
2.3.5 *Session Description Protocol (SDP)* 13
2.4 *WebSockets* 14
2.5 Sinalização 15
2.5.1 *SIP over WebSockets* 15
2.5.2 *Jingle over WebSockets* 16
2.5.3 *JavaScript Session Establishment Protocol (JSEP)* 17
2.6 *Network Address Translation (NAT)* 18
2.6.1 *Interactive Connectivity Establishment (ICE)* 18
2.6.2 *Session Traversal Utilities for NAT (STUN)* 18
2.6.3 *Traversal Using Relays around NAT (TURN)* 19
2.7 *Codecs* 19
2.8 Sumário 20

3 Soluções *WebRTC* **21**
3.1 Soluções existentes 21
3.1.1 *PeerCDN* 21
3.1.2 *Conversat.io* 22
3.1.3 *BananaBread* 22
3.1.4 *Responsive Web Typography with WebRTC* 22
3.1.5 *SIPML5* 23
3.2 Bibliotecas *JavaScript* 23

3.2.1 *WebRTC Experiment* 24
3.2.2 *SimpleWebRTC* 26
3.2.3 *Adapter.js* 26
3.3 WebSockets 27
3.3.1 *Node.js* 27
3.3.2 *Vert.x* 28
3.3.3 *Google App Engine* 30
3.4 *Gateways* para *SIP* 30
3.4.1 *Asterisk* 31
3.4.2 *WebRTC2SIP (sipML5)* 31
3.5 Experimentação e seleção de soluções existentes 33
3.5.1 *Vert.x vs Node.js* 33
3.5.2 Bibliotecas *JavaScript WebRTC* 37
3.6 Sumário 39

4 *Framework Webrtc* 41
4.1 Requisitos funcionais 41
4.1.1 Chamada com controlo de estabelecimento de chamada 41
4.1.2 Chamada básica com controlo completo da chamada 42
4.1.3 Chamada com pessoas externas 42
4.1.4 Presença e Lista de contactos 42
4.1.5 Chat / Mensagens Instantâneas 43
4.1.6 Partilha de Ficheiros 43
4.1.7 Gravação de Voz e Vídeo 44
4.2 Arquitetura da *Framework* 44
4.2.1 Especificações Iniciais 44
4.2.2 Estrutura arquitetural 46

4.3 Descrição dos Módulos 47
4.3.1 Módulo servidor *web* 47
4.3.2 Módulo de base de dados 48
4.3.3 Módulo de gestão de autenticações e autorizações 49
4.3.4 Módulo de gestão de sessões 50
4.3.5 Módulo de registo utilizador 51
4.3.6 Módulo de pesquisa e intersecção de serviços 52
4.3.7 Módulo de gestão de utilizadores 52
4.3.8 Módulo de presença 53
4.3.9 Módulo de histórico de chamadas 53
4.3.10 Sinalização 54
4.4 Mensagens 54
4.4.1 Registo de cliente 54
4.4.2 Autenticação de Cliente 57
4.4.3 Módulo de presença e Interação no sistema 60
4.4.4 Módulo de gestão de utilizadores 63
4.5 Conferências áudio e vídeo 64
4.6 Sumário 68

5 Cenários de teste e resultados obtidos **71**
5.1 Testes de validação 71
5.2 Testes de desempenho computacional 72
5.2.1 Cenário 1 - Rede empresarial 74
5.2.2 Cenário 2 - Rede doméstica 78
5.3 Sumário 80

6 Conclusões e trabalho futuro **83**

6.1 Resumo do trabalho desenvolvido 83
6.2 Principais contribuições 85
6.3 Trabalho futuro 85

A Anexo A - Mensagem SDP **87**

B Anexo B **91**

C Anexo C **93**
C.1 Cenário 1 93
C.2 Cenário 2 94

Bibliografia **97**

Lista de Figuras

1.1 Estimativa da evolução da tecnologia *WebRTC* [3]. 2

1.2 Exemplo de motivação. 3

2.1 Arquitetura *WebRTC* [6]. 8

2.2 Pilha *Data Channel* [12]. 11

2.3 Comparação de latência entre *polling* e aplicações *Websockets* [21]. 15

2.4 Processo *Jingle* [23]. 17

2.5 Modelo *JSEP* [24]. 17

3.1 Funcionamento geral da aplicação Vert.x [46]. 29

3.2 Arquitetura *SipML5* [29]. 32

3.3 RTCWeb Breaker SipML5 [29]. 32

3.4 *Media Coder SipML5* [29]. 32

3.5 Esquema de interação de tecnologias. 33

3.6 Resultados do teste de respostas 200-OK [50]. 34

3.7 Resultados do teste de ficheiro 72bytes [50]. 35

3.8 Mensagens recebidas - *node.js* [51]. 36

3.9 Mensagens recebidas por segundo - *node.js* [51]. 36

3.10 Mensagens recebidas - *vert.x* [51]. 36

3.11 Mensagens recebidas por segundo - *vert.x* [51]. 36

4.1 Especificação inicial dos módulos. 44
4.2 Especificação da interação entre módulos. 45
4.3 Especificação da interação dos clientes com módulo base de dados. 46
4.4 Arquitetura do sistema. 47
4.5 Formulário de registo no sistema. 55
4.6 Mensagem de registo no sistema. 55
4.7 Mensagem para gravar grupo de utilizador. 56
4.8 Gravar serviços subscritos por um utilizador. 57
4.9 Autenticação de um utilizador no sistema. 57
4.10 Iniciar sessão para um utilizador. 58
4.11 Guardar informação do utilizador na sessão associada. 58
4.12 Mensagem de verificação de serviços de um utilizador. 59
4.13 *Array* de serviços em resposta do servidor. 59
4.14 Mensagem para obter os estados de presença do sistema. 61
4.15 Mensagem para notificar e obter lista de contactos. 62
4.16 Notificação de mudança de estado. 62
4.17 Mensagem para adicionar contacto. 63
4.18 Mensagem para pesquisa de utilizadores do sistema. 64
4.19 Convidar pessoas para uma sessão. 65
4.20 Mensagem de notificação para início de sessão. 65
4.21 Notificação para início de sessão. 66
4.22 Conferência entre três pessoas. 66
4.23 Registo de chamada no histórico de chamadas. 68

5.1 *Ping* ao servidor com VPN. 73
5.2 *Ping* ao servidor na rede doméstica. 74
5.3 Cenário 1 e Teste 1. 75

5.4 Gráfico de utilização de memória numa chamada. 75

5.5 Cenário 1 e Teste 2. 76

5.6 Utilização de *CPU* no inicio da conferência. 76

5.7 Utilização de *CPU* ao finalizar conferência. 76

5.8 Comportamento da utilização de *CPU*. 76

5.9 Utilização de memória ao iniciar a conferência. 76

5.10 Utilização de memória ao finalizar conferência. 77

5.11 Cenário 2 e teste 1. 79

5.12 Resultados de *CPU* no computador com menor poder computacional. 79

5.13 Resultados de *CPU* PC4. 80

5.14 Cenário 2 e teste 2. 80

B.1 Funcionamento de sinalização WebRTC. 92

Lista de Tabelas

2.1 Versões dos *web browsers* de suporte às *APIs WebRTC* [8]. 10
2.2 Percentagem dos *web browsers* usados [11]. 11
2.3 Tabela de suporte *codecs* nos diferentes *web browsers* [29]. 20

3.1 Perfixos das interfaces nos diferentes *browsers* [41]. 26

5.1 Percentagem dos *web browsers* usados [11]. 72
5.2 Utilização do CPU no servidor durante alguns eventos entre cliente e servidor. . . 77
5.3 Utilização de CPU no servidor durante uma conferência de 4 pessoas. 78

List of Acronyms

API	Application Programming Interface
CPU	Central Processing Unit
DMTF	Dual-Tone Multi-Frequency
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
HTML5	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICE	Interactive Connectivity Establishment
IE	Internet Explorer
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IP	Internet Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MTU	Maximum Transmission Unit
NAT	Network Address Translation
PBX	Private Branch Exchange
PHP	Hypertext Preprocessor

PSTN	Public Switched Telephone Network
P2P	Peer-to-Peer
QoS	Quality of Service
RTP	Real Time Protocol
RTT	Round Trip Time
SCTP	Stream Control Transmission Protoco
SIP	Session Initiation Protocol
SMS	Short Message Service
SQL	Structured Query Language
SSL	Secure Sockets Layer
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TLS	Transport Layer Protocol
TURN	Traversal Using Relays around NAT
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WebRTC	Web Real Time Communications
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Capítulo 1

Introdução

A *web* começou desde cedo, na data de 1945, a ser pensada apresentando um sistema chamado *Memex*, permitindo seguir *links* e documentos em microfilme. Em 1960 é apresentado um sistema semelhante onde as ligações são feitas em documentos de texto, contudo, no mesmo ano já se falava em *hypertext* [1].

Depois disso surgiu a criação da *Advanced Research Projects Agency Network (ARPANET)* e começaram a surgir protocolos de comunicação, como também, um sistema de email. Em 1989 Tim Berners-Lee apresenta a proposta *Information Management: A Proposal*, que dizia respeito à gestão de informação sobre o acelerador de partículas e experimentações no CERN. Essa proposta aborda problemas de perda de informação em sistemas complexos e apresenta uma solução baseada no sistema de hypertext distribuído [2]. No ano seguinte Tim começa a trabalhar na implementação de um *browser* e num editor que permitisse o uso de *hyperlinks* [1].

Todo este processo foi evoluindo ao longo dos anos e, nos dias de hoje, a *Word Wide Web* (W3) é imprescindível no dia a dia das pessoas, seja num computador, num *tablet* ou até mesmo num *smartphone*. Face a esta grande evolução, várias discussões têm sido geradas acerca de aplicações nativas *"versus"* aplicações *web*. Por conseguinte, as grandes empresas estão cada vez mais a apostar em tecnologias *web*, exemplo disso é a *Google*, que tem uma série de serviços disponibilizados a partir de um *web browser* (*Gmail*, *Drive Google*, *PlayStore*, etc.). Para além desses serviços e com a ajuda da *World Wide Web Consortium (W3C)* e da *The Internet Engineering Task Force (IETF)*, foi possível criar uma tecnologia que permitisse a comunicação áudio e vídeo entre *web browsers*, que resultou na tecnologia *Web Real Time Communication (WebRTC)*. Com esta tecnologia é possível a criação de aplicações que permitam realizar comunicações em tempo real entre várias pessoas, como é o caso de algumas aplicações nativas (e.g., *Skype*). Esta

recente tecnologia foi apresentada em 2012 pela Google, atingindo uma rápida evolução, permitindo o surgimento de várias soluções utilizando as suas *Application Programming Interfaces (APIs)*. Num futuro próximo, prevê-se uma evolução exponencial do uso desta tecnologia não só ao nível de computadores, mas também em *tablets* e *smartphones*, como ilustrado na Figura 1.1.

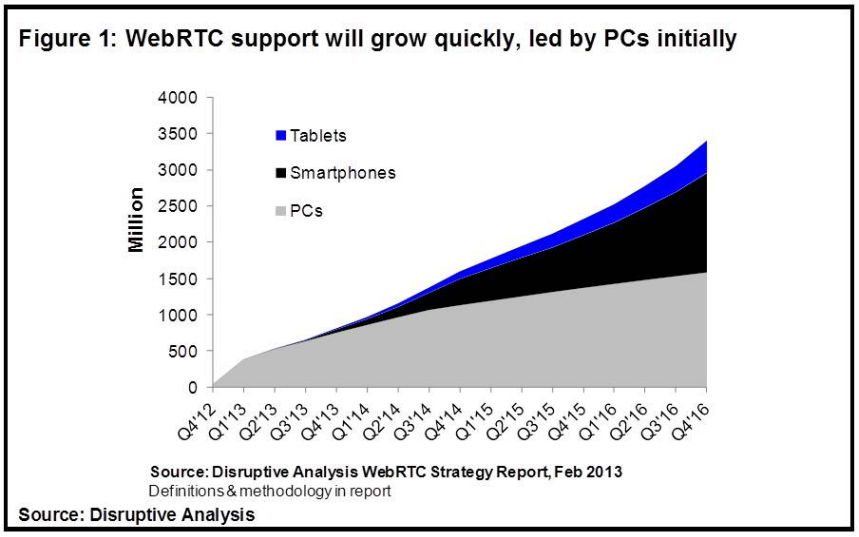


Figura 1.1: Estimativa da evolução da tecnologia *WebRTC* [3].

1.1 Motivação e objetivos

A tecnologia *WebRTC* tem vindo a evoluir desde que a *Google*, o *W3C* e o *IETF* começaram a trabalhar na sua normalização [4]. Desde então, têm vindo a ser desenvolvidas várias aplicações *WebRTC*, que serão apresentadas no decorrer desta dissertação. Em contraste das normas *Voice over Internet Protocol (VoIP)*, que utilizam vulgarmente protocolos de sinalização como *Session Initiation Protocol (SIP)*, *WebRTC* deixa em aberto de que forma a sinalização entre *peers* é efetuada, ficando esta decisão a cargo de quem desenvolve a aplicação.

A Figura 1.2, esquematiza a principal motivação deste projeto, que deriva do facto de na camada aplicacional não existir um mecanismo normalizado para a troca de informação de sinalização entre *peers*. Na tecnologia *WebRTC* é necessário haver uma negociação de sinalização entre os diferentes *peers*, de forma a criar ligações *peer* entre eles, para que a informação *media* e dados possa ser trocada.

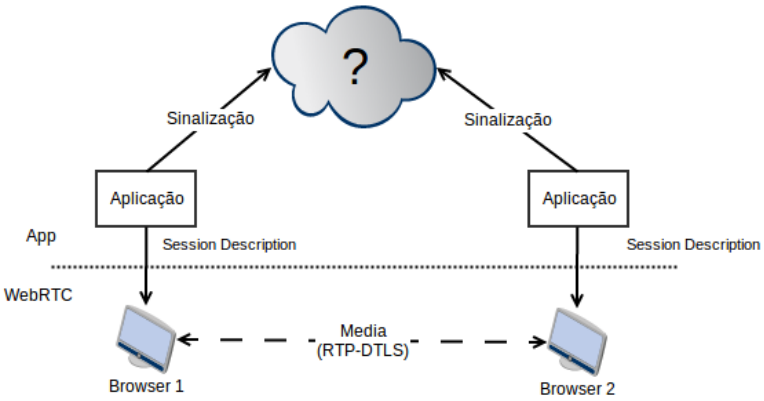


Figura 1.2: Exemplo de motivação.

Neste projeto efetuou-se uma análise aprofundada à tecnologia *WebRTC* e também a tecnologias que ajudem na troca de informação de sinalização entre os diferentes *peers*. Atendendo a que a solução a desenvolver usará tecnologias web, por estratégia da empresa, o destaque a este nível incidirá nas tecnologias WebSockets.

Neste contexto, este projeto tem como objectivo o estudo de várias tecnologias *WebSockets* e fazer uma análise comparativa entre essas diferentes tecnologias. Desta análise comparativa serão tomadas algumas decisões sobre as tecnologias que devem ser usadas no desenvolvimento de uma *Framework* que tem por objectivo principal facilitar o desenvolvimento de serviços *WebRTC* (tais como presença, histórico de chamadas, conferência, etc) a um alto nível.

Para isso foram definidas as seguintes fases no desenvolvimento deste projeto:

- estudo da tecnologia *WebRTC* e de soluções que permitam a comunicação entre *web browsers*;
- definição de casos de uso e definição de uma arquitectura *WebRTC* para *Framework*;
- desenvolvimento da *Framework* e definição da prova de conceito a realizar;
- testes e análise de resultados.

A prova de conceito servirá, essencialmente, para verificar se os serviços desenvolvidos na *Framework* se encontram operacionais. Para esse efeito, usando esta *Framework*, foi desenvolvido um cliente *WebRTC*, implementando os serviços de presença, histórico de chamadas, chamadas áudio e vídeo, partilha de ficheiros, partilha de ecrã e *chat*.

Para além dos testes de operacionalidade dos serviços, foram realizados testes ao nível de utilização de memória e *Central Processing Unit (CPU)*, tanto nos clientes como nos servidores. Desta forma poder-se-á verificar qual o peso computacional da tecnologia *WebRTC* nas diferentes máquinas. Tal permitirá ainda verificar se o servidor é usado durante uma chamada / conferência entre diferentes utilizadores, ou se é apenas necessário para a negociação da sinalização entre *peers*. Desta forma, é possível verificar se as ligações são realmente *peer-to-peer*, ou se o acesso ao servidor é recorrente para os *peers* realizarem essa comunicação.

1.2 Principais contribuições

Uma das discussões que se encontra mais em voga é relativa ao uso *web "versus"* aplicações nativas. Maior parte das análises realizadas fazem uma avaliação em termos de desempenho, verificando até que ponto seria viável a criação de aplicações *web* em vez de aplicações nativas. Neste sentido, existem cada vez mais aplicações *web* que fazem o mesmo que as várias aplicações nativas, cuja a única diferença existente é no facto das aplicações *web* correrem sobre um *web browser*.

A tecnologia *WebRTC* permite realizar comunicações entre diferentes clientes em tempo real. Como este projeto está a ser desenvolvido num âmbito empresarial, pretende-se tirar partido da *WebRTC* ser uma tecnologia recente, estudando-a em profundidade. Desta forma, pode-se verificar que vantagens poderá trazer no futuro das telecomunicações e que serviços poderão ser implementados, em comparação às aplicações nativas já existentes.

Assim, as contribuições deste trabalho surgem pelo estudo e classificação dos conceitos associados à *WebRTC* e, numa prespectiva mais prática, pelo desenvolvimento de uma *Framework* que permita a outras entidades desenvolver soluções *WebRTC*, a um mais alto nível e de uma forma mais simplificada. Para além disso, foi desenvolvido um cliente *WebRTC*, para verificar as funcionalidades da *Framework* desenvolvida. Assim sendo é possível explorar e estudar mais aprofundadamente a tecnologia *WebRTC*, analisando as vantagens e desvantagens desta tecnologia, como também estar ao corrente de todas as atualizações e modificações feitas na tecnologia.

Como fruto deste trabalho, foi elaborado e submetido um artigo científico na "13ª Conferência de Redes de Computadores"[5], já aceite para publicação. Foi também realizado um artigo, sendo este publicado na Revista Saber e Fazer Telecomunicações, da empresa em questão.

1.3 Organização da dissertação

No presente capítulo, Capítulo 1, é feita uma introdução aos conceitos das tecnologias que são abordadas nesta dissertação. É referida a principal motivação e os objectivos deste projeto, como também quais as principais contribuições.

No Capítulo 2 abordam-se os conceitos associados à tecnologia WebRTC, apresentando-se as *APIs* que a constituiem, que protocolos são usados, os *codecs* e a compatibilidade relativamente aos *web browsers* mais usados atualmente. É também abordada a componente de sinalização, onde é referido o protocolo *WebSockets*.

No Capítulo 3, relativo ao estado da arte, são apresentadas várias soluções que usam as diferentes *APIs* da tecnologia *WebRTC*. Para além de aplicações existentes, são apresentadas bibliotecas, repositórios e *frameworks*, que se encontram desenvolvidas e ainda são descritas soluções relativamente a *WebSockets*.

No Capítulo 4 são delineados os casos de uso e requisitos definidos para a aplicação cliente e *framework* a desenvolver neste projeto. Apresenta-se a arquitetura e estrutura de todo o sistema desenvolvido, bem como as mensagens trocadas entre os módulos do servidor e também com os clientes.

No Capítulo 5 são apresentados resultados dos testes realizados no decorrer do projeto. São apresentados dois cenários com características diferentes, para se poder analisar comportamentos relativamente aos sistemas e à rede.

No Capítulo 6 são apresentadas as principais conclusões resultantes do trabalho e pesquisa efetuada, quais as principais contribuições deste projeto e é também feita uma análise relativamente ao trabalho futuro.

Capítulo 2

Conceitos Introdutórios

O presente capítulo aborda os conceitos introdutórios deste projeto. São apresentados os conceitos de *WebRTC* e as suas *APIs*, a compatibilidade atual. De seguida são descritos os protocolos relacionados com a *API DataChannel WebRTC*, o protocolo WebSockets e ainda os métodos de sinalização que são normalmente usados neste tipo de soluções. Apresenta-se ainda o problema existente de *Network Address Translation (NAT)* relacionado com a tecnologia *WebRTC*. Por fim abordam-se os *codecs* utilizados por cada browser.

2.1 *WebRTC*

As *Real Time Communications (RTC)*, através da *web* têm vindo a evoluir devido a dois organismos de normalização – *Internet Engineering Task Force (IETF)* e *World Wide Web Consortium (W3C)*. *WebRTC* é um projeto *open source*, grátis e normalizado que permite aos browsers realizarem comunicações em tempo real. Um dos principais objetivos é desenvolver aplicações em *web browsers*, com a ajuda de *APIs* de *JavaScript* e *HyperText Markup Language 5* (*HTML5*) [6].

Esta tecnologia permite ainda o acesso seguro a componentes internos (como a *webcam* e/ou microfone) de qualquer sistema, desta forma é possível receber e enviar informação (*media*) *peer-to-peer* (*P2P*), em tempo real entre dois *browsers*, sem qualquer tipo de *plugin*.

A Figura 2.1, ilustra a arquitetura da tecnologia WebRTC, onde se verifica que é constituída por duas *APIs* diferentes: *WebRTC C++ API* e *Web API*. A *WebRTC C++ API* é utilizada por programadores, que queiram desenvolver *web browsers*, enquanto a *Web API* é usada por pessoas

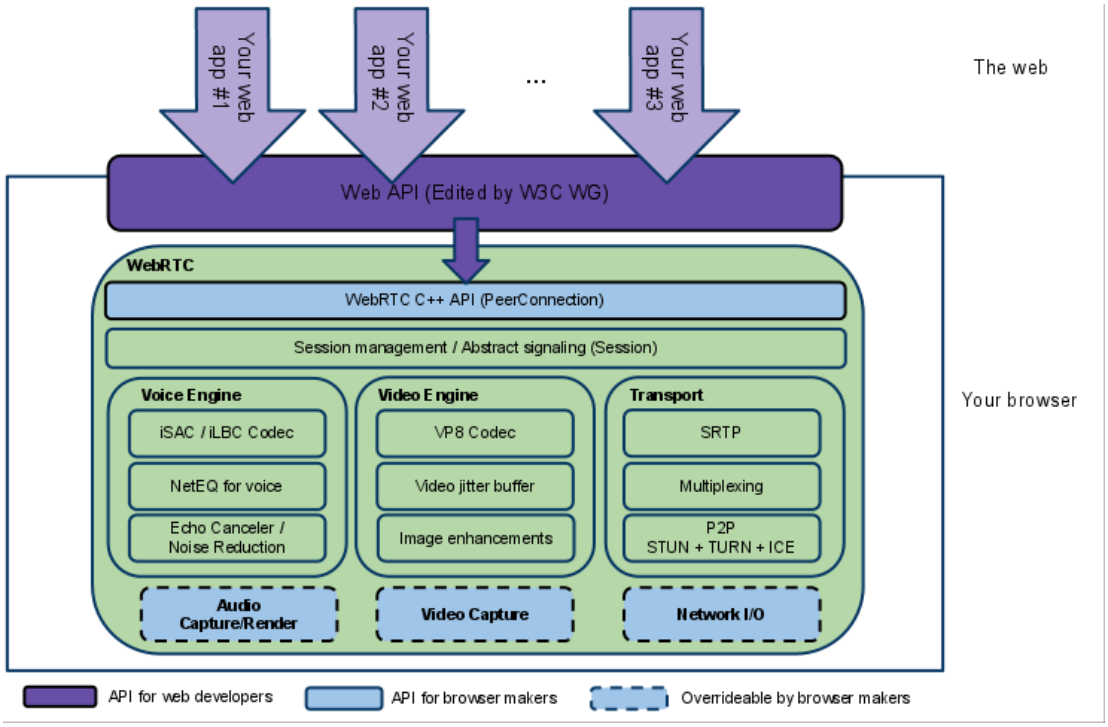


Figura 2.1: Arquitetura *WebRTC* [6].

que queiram desenvolver aplicações *web*, com o objetivo de tirar proveito das funcionalidades desta tecnologia.

Como este projeto consiste na criação de uma aplicação *WebRTC*, na secção seguinte é apresentada a *Web API WebRTC* e os conceitos associados, a compatibilidade atual desta tecnologia nos *web browsers* e quais os protocolos usados para comunicação e para sinalização.

2.1.1 WebRTC C++ API

A WebRTC C++ é a API que é utilizada para o desenvolvimento de *web browsers*, no que diz respeito à implementação *WebRTC*. Sendo esta tecnologia *open source*, o código fonte encontra-se disponível para qualquer pessoa [7]. Para além do desenvolvimento de *web browsers*, é possível estar atento a novas atualizações e até mesmo ajudar a encontrar erros, visto que todo o código se encontra num repositório *Subversion (SVN)*.

2.1.2 *WebRTC API*

Esta *API* permite desenvolver aplicações em *web browsers*, o que fez com que a linguagem de programação adotada fosse *JavaScript*, assim as aplicações são desenvolvidas do lado do cliente, o que permite explorar as capacidades em tempo real dos browsers. A API WebRTC é definida sobre três principais conceitos *PeerConnection*, *MediaStreams* e *DataChannel*, apresentados a seguir [8, 9].

PeerConnection

PeerConnection permite que dois utilizadores comuniquem diretamente, *browser-to-browser*. Para estabelecer esta ligação e haver uma negociação de sinalização, é necessário que haja um canal de sinalização. Este é implementado num servidor *Web*, utilizando *WebSockets* ou *XML-HttpRequest*. Este mecanismo usa o protocolo *Interactive Connectivity Establishment (ICE)* juntamente com *Session Traversal Utilities for NAT (STUN)* e *Traversal Using Relays arround NAT (TURN)* para permitir ajudar as *streams* de *media* a passarem por *Network Address Translation (NAT)* e *firewalls* [8, 9, 10].

Media Streams

MediaStream é uma forma abstrata de representar uma *stream* de dados áudio e/ou vídeo. Este tipo de aplicações pode ser usada para mostrar, gravar ou enviar o seu conteúdo para um *peer* remoto. Existem dois tipos de *stream*: *Local MediaStream* ou *Remote MediaStream*. *Local MediaStream* é a *stream* capturada no próprio sistema (*webcam* e microfone) enquanto que *Remote MediaStream* é a *stream* recebida de outro *peer*. Para ter acesso à *media* dos componentes do terminal é necessário executar a função *getUserMedia()*, onde podem ser definidos alguns parâmetros, dependendo do que o utilizador do terminal queira reproduzir. Para a transmissão das *streams* são usados protocolos como *Secure Real-time Transport Protocol (SRTP)*, *Real-time Transport Protocol (RTP)* e *Real-time Transport Control Protocol (RTCP)* para monitorização de transmissão de *media*. O Protocolo *Datagram Transport Layer Security (DTLS)* é usado como uma chave de *SRTP* e para gestão de associações [8, 9, 10].

RTC Data Channels

RTCDataChannel é um canal de dados bidirecional em ligações *peer-to-peer*. É possível serem transferidos dados que não sejam *media* e é necessário usar outro protocolo, como *SCTP* encapsulado em *DTLS*. Desta forma existe uma solução para *NAT* com confidencialidade, autenticação da fonte e integridade dos dados que sejam necessários transferir. O *SCTP* permite a entrega, fiável e não fiável, de dados e permite que as aplicações abram várias *streams* independentes. Para a criação de um *DataChannel* é necessário executar a função *CreateDataChannel()* numa instância da *PeerConnection* [8, 9, 10].

2.2 Compatibilidade

Por *WebRTC* ser uma tecnologia *web*, esta não depende apenas de si própria, assim, é essencial garantir que haja compatibilidade nos diferentes *web browsers*.

É então necessário garantir que os *web browsers* suportem *HTML5*, que é um novo *standard* para a *syntax HTML*. Suporta novas funcionalidades, como a redução da utilização de *plugins* adicionais como *flash*, novas *tags* que permitem substituir *scripting* e melhoramentos no tratamento de erros.

Juntando *HTML5* e *WebRTC*, é possível ter uma solução *web* para realização de chamadas áudio e vídeo, pois *WebRTC* tira proveito das *tags* áudio e vídeo para reprodução do conteúdo de *streams*. Isto é feito sem qualquer tipo de *plugin* adicional, desde que o *web browser* suporte essas funcionalidades.

As três *APIs* que o *WebRTC* implementa, já são suportadas em alguns *browsers*, mas com algum trabalho por realizar. A tabela 2.1 apresenta o suporte dos *web browsers* em relação a essas *APIs*.

Tabela 2.1: Versões dos *web browsers* de suporte às *APIs WebRTC* [8].

API	Internet Explorer	Firefox	Chrome	Opera
MediaStream	Não Suporta	Suporta >v17	Suporta >v18	Suporta >v12
Peer Connection	Não Suporta	Suporta >v22	Suporta >v20	Sem Informação
RTC Data Channels	Não Suporta	Suporta >v22	Suporta >v26	Sem Informação

A tabela 2.2 permite perceber quais os *web browsers* mais utilizados entre março e julho de

2013.

Tabela 2.2: Percentagem dos *web browsers* usados [11].

2013	Internet Explorer	Firefox	Chrome	Opera	Safari
Julho	11,8%	28,9%	52,8%	1,6%	3,6%
Junho	12,0%	28,9%	52,1%	1,7%	3,9%
Maio	12,6%	27,7%	52,9%	1,6%	4,0%
Abril	12,7%	27,9%	52,7%	1,7%	4,0%
Março	13,0%	28,5%	51,7%	1,8%	4,1%

Este tipo de informação é importante pois é necessário saber quais os *web browsers* mais usados recentemente, de forma a que possa haver uma interoperabilidade entre eles no que diz respeito a estas novas tecnologias.

2.3 Protocolos *data channel*

Para além de *media* na tecnologia *WebRTC*, que é trocada sobre *peer connections* em canais de *media* (RTP), existe também a possibilidade de trocar dados entre os utilizadores de uma sessão. Foi assim que surgiram os canais de dados (*Data Channels*) nesta tecnologia. Os *Data Channels* usam quatro tipo de protocolos, *Stream Control Transmission Protocol (SCTP)* para controlo de *streams*, *Datagram Transport Layer Security (DTLS)* para segurança dos dados e *UDP/ICE* que são definidos como protocolos da camada de transporte.

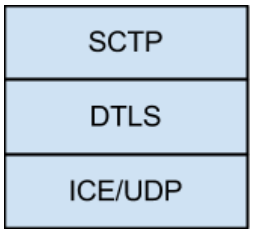


Figura 2.2: Pilha *Data Channel* [12].

A figura 2.2 representa a *stack* base para os *Data Channels*, em que *SCTP* se encontra encapsulado em *DTLS*, que por sua vez se encontram encapsulados em *ICE/UDP* [12].

2.3.1 *Stream Control Transmission Protocol (SCTP)*

O *SCTP* é um protocolo fiável que trabalha na camada de transporte, de redes que não garantem entrega de pacotes no destino, exemplo disso é a rede *Internet Protocol (IP)*. É um protocolo que oferece serviços como, entrega livre de erros, garantia que os dados não são entregues duplicados, fragmentação de dados de acordo com o *Maximum Transmission Unit (MTU)*, entrega sequencial de mensagens de utilizador em múltiplas *streams* com opção de entrega ordenada, tolerância a falhas ao nível da rede, devido ao suporte de *multi-homing* nos terminais de associação [13]. Para além destes serviços este protocolo implementa métodos para evitar congestão de dados e evita ataques de *flooding* e *masquerade* [13].

2.3.2 *Datagram Transport Layer Security (DTLS)*

O *DTLS* é um protocolo usado para permitir a segurança de tráfego na rede, assim, este, é como *"TLS over Datagram"* onde a sua utilização tem como objetivo proteger dados em aplicações de comunicação. *DTLS* é uma solução criada devido ao aparecimento de protocolos ao nível aplicacional, que usam como protocolo de transporte *User Datagram Protocol (UDP)*.

DTLS é uma variante do *TLS*, para protocolos que usam *UDP*. A sua definição permitiu usar a maior parte da infraestrutura e código que já tinha sido definido no *TLS* [14]. Foi então desenhado de forma a proteger dados em aplicações de comunicação, que não oferecem fiabilidade ou qualquer ordem de entrega de dados. Quando usado, não traz diferenças no que diz respeito a atrasos nas aplicações e para além disso não trata de perdas nem de ordenar os dados [14].

2.3.3 *User Datagram Protocol (UDP)*

User Datagram Protocol (UDP) permite transmitir mensagens com o mínimo *overhead* protocolar e sendo um protocolo base da família TCP/IP, sem prévio estabelecimento de conexões. Este protocolo é normalmente usado para transportar pequenos pacotes de dados e para trocar media em sessões *Real Time Protocol (RTP)*. Uma vez que as aplicações de comunicação em tempo real não são tolerantes a falhas ou a atrasos, logo é necessário garantir uma entrega rápida de pacotes de dados. O protocolo *UDP* responde a essas exigências, porém não garante entrega ordenada dos dados, não retransmite pacotes falhados e não tem controlo de congestão. Existem métodos ao nível aplicacional que podem ajudar na resolução de alguns destes problemas, como *codecs*, sistema de redução da largura de banda enquanto existe congestão [15].

Este protocolo é fornecido pelo sistema operativo sob o *web browser*.

2.3.4 *Secure Real-Time Protocol - Datagram Transport Layer Security*

O *RTP* é um protocolo de entrega de serviços fim-a-fim em tempo real, usado em redes *IP*. Inicialmente foi desenvolvido para aplicações onde existem vários participantes, ou seja, conferências multimédia (áudio e vídeo). Atualmente é usado por diferentes aplicações *peer-to-peer*, tais como, *Simple Multicast Audio*, conferências áudio e vídeo, *Mixers* and *Translators* e *Layered Encodings* [16]. No entanto, o *RTP* não garante *qualidade de serviço (QoS)* ou entrega de pacotes de dados, contudo consegue uma reconstrução temporal, deteção de perda, segurança e identificação de conteúdo [16, 17].

O *SRTP* tal como o protocolo *RTP* fornece confidencialidade e autenticação de mensagens. A solução *DTLS-SRTP* está definida para sessões *media point-to-point*, onde se encontram apenas dois participantes.

DTLS é usado como uma extensão do *SRTP*, uma vez que este não permite uma gestão de chaves nativa, assim, é necessário recorrer a mecanismos externos. Integra essa gestão de chaves, negociação de parâmetros e transferência segura de dados, desta forma é possível ter uma solução que combina desempenho e encriptação do SRTP, com a gestão de chaves e associações do DTLS [17]. Esta é a solução que WebRTC usa para a encriptação de dados RTP e gestão de chaves entre dois clientes.

2.3.5 *Session Description Protocol (SDP)*

SDP é um protocolo que permite a descrição de uma sessão entre dois clientes. Quando se inicia uma conferência multimédia, chamadas *Voice over IP (VoIP)*, *streaming* de vídeo, ou outro tipo de sessão, é necessário ter em conta algumas características, como os detalhes de *media* (e.g. *codecs*), os endereços de transporte, e outros dados da sessão que precisam de ser trocados entre os participantes [18]. É um protocolo completamente de texto com uma grande quantidade de informação, mas que é essencial para estabelecer uma sessão de *media* entre os clientes. Este protocolo é usado numa grande quantidade de aplicações e diferentes tipos de redes, no entanto, não suporta nenhum tipo de negociação de sessões. A descrição de uma sessão SDP inclui o propósito e o nome da sessão, o tempo que a sessão está ativa e informação para compreender os meios de comunicação da sessão e a informação necessária para receber esses meios (endereços,

portas, formatos, etc). Caso os recursos sejam limitados, convém incluir informação sobre a largura de banda usada na sessão e sobre o contacto da pessoa que está responsável pela sessão [18].

No caso da tecnologia *WebRTC*, a informação é codificada num objeto designado por *RTCSessionDescription* [8, 10]. Este objeto serve para descrever a sessão entre dois clientes e as características *media* de uma ligação *peer* (*Peer Connection*). Este processo na tecnologia *WebRTC* é conhecida pelos métodos *offer* e *answer*. A *offer* leva uma descrição sobre os protocolos, os *media* e os *codecs* que o cliente emissor suporta, já no caso de *answer*, é uma resposta originada caso seja recebida a *offer* e leva a mesma informação, mas neste caso do cliente recetor [8, 10].

2.4 *WebSockets*

A criação de aplicações *web* tem sido efetuada com base no mecanismo pedido/resposta do protocolo *HTTP*, entre cliente e servidor. Depois da informação de uma página *HTTP* ser carregada é necessário que exista uma interação do utilizador, como por exemplo, abrir outra página, para que haja uma atualização da informação que é mostrada. Alguns destes problemas vieram ser resolvidos com o aparecimento do AJAX tornando os sítios mais dinâmicos, porém teria de haver sempre uma interação do utilizador ou um temporizador que atualizasse a página, para ver os dados atualizados. As tecnologias que permitem que o servidor envie dados atualizados mal os recebam, foram usadas desde algum tempo. Uma das soluções era a sondagem longa, que permite a abertura de uma ligação ao servidor, até que este receba um pedido. Um dos principais problemas deste tipo de soluções era o overhead de HTTP, que poderia prejudicar soluções que não são tolerantes a atrasos. Para além disto, o servidor era obrigado a criar várias ligações TCP para cada cliente, uma para enviar informação e várias para cada mensagem recebida [19, 20]. WebSocket Protocol foi desenhado para resolver este tipo de problemas. É um protocolo que fornece uma comunicação bidirecional, é baseado em *sockets* e usa HTTP como camada de transporte. Desta forma é possível usar *WebSockets* com soluções já existentes, visto que pode trabalhar sob portas HTTP como 80 e 443, o que facilita a integração deste protocolo com as soluções já existentes. Para além disso, este protocolo não está apenas limitado ao HTTP, o que permite fazer uma *handshake* simples sob uma porta sem reinventar o protocolo [19, 20].

Na Figura 2.3 estão representados dois tipos de ligações: *polling (HTTP)* e *WebSockets*. Numa ligação *polling* o *browser* necessita de estar sempre a fazer pedidos, pois o servidor só

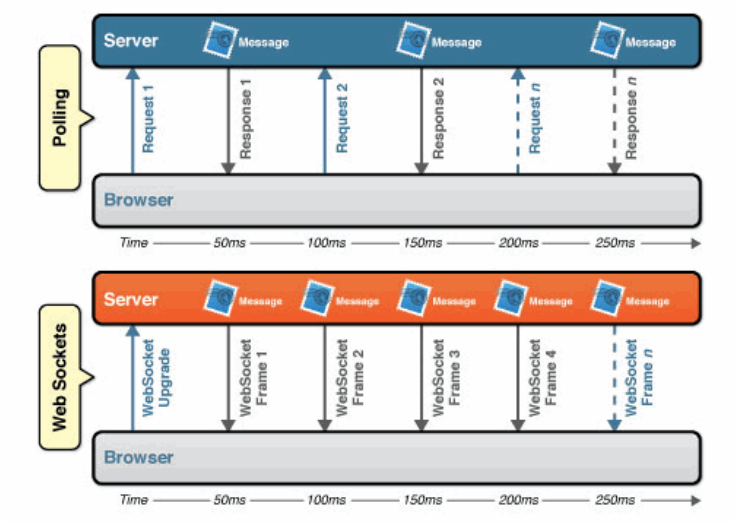


Figura 2.3: Comparação de latência entre *polling* e aplicações *Websockets* [21].

responde no caso de receber esses pedidos. No caso do *WebSocket* o *browser* faz apenas um pedido e o servidor envia toda a informação em vários pacotes de dados, sem ter que receber qualquer pedido adicional por parte do *browser*. Esta ligação entre o servidor e o *web browser* é designada como uma ligação persistente, pois sempre que um deles precisar de enviar qualquer informação podem fazê-lo a qualquer momento.

2.5 Sinalização

Ao contrário do que acontece com uma infraestrutura completamente pensada para estabelecer uma ligação ou até mesmo controlar todo o *media*, o mesmo não acontece quando se fala da sinalização. Para estabelecer uma chamada entre os utilizadores, é necessária uma negociação de parâmetros da sessão, descritos via SDP. A definição do método a utilizar fica a cargo da camada aplicacional, ou seja, a pessoa que está a desenvolver uma aplicação pode definir o método que quer para a sinalização. Apesar disso existem alguns protocolos que definem como deve ser feita a sinalização.

2.5.1 *SIP over WebSockets*

O protocolo *WebSockets*, descrito na secção 2.4, permite a troca de mensagens entre clientes *web* e o servidor, em tempo real e de uma forma persistente. *SIP over Websockets* é uma especificação

que define um subprotocolo de *Websocket*, permitindo a troca de mensagens *SIP* entre um cliente e um servidor *web*. Está definido na camada da aplicação e é um protocolo fiável.

Existem duas entidades principais nesta especificação [22]:

- *SIP WebSocket Client*: entidade capaz de abrir ligações de *websockets* saída para o servidor e que comunica por *WebSocket SIP subprotocol*.

- *SIP WebSocket Server*: entidade que escuta por ligações de entrada dos clientes e comunica por *WebSocket SIP subprotocol*.

Em suma, este é um protocolo idêntico ao *WebSocket*, mas que transporta mensagens *SIP* entre as entidades já apresentadas.

2.5.2 *Jingle over WebSockets*

Extensible Messaging and Presence Protocol (XMPP) Jingle define um protocolo *XMPP* para a gestão de sessões *media peer-to-peer*, que permite fazer a gestão de múltiplos conteúdos em simultâneo [23]. Este protocolo foi definido principalmente para comunicações em tempo real, como presença e chat, visto que, as mensagens trocadas por este protocolo não são enviadas em *frames*, mas em *streams XMPP*. A especificação *Jingle* permite fazer a negociação de parâmetros de uma sessão entre dois clientes. Depois dessa negociação é criado um caminho de media onde são trocadas *Jingle streams*, diretamente entre dois utilizadores.

Em comparação ao que acontece com a sinalização com outros protocolos (e.g. *SIP*), a sinalização é feita a partir de um caminho *XMPP* e de *Jingle media*. Como este tipo de técnicas são baseadas em *Hypertext Transfer Protocol (HTTP) long polling*, que envolve demasiado *overhead*. Uma solução para este problema consiste em evitar *XMPP* sobre *HTTP*, o que indica que seria necessário usar *XMPP* nativo. Porém os *web browsers* ainda não suportam essas funcionalidades nativamente, para resolver esse problema existe o protocolo *WebSocket*. À semelhança de *"SIP sobre WebSockets"*, a solução *Jingle* sobre *WebSockets* permite enviar informação *XMPP* entre os vários *web browsers* e os servidores. Desta forma a comunicação e a troca de mensagens entre as várias identidades é feita de uma forma mais robusta e eficiente [24].

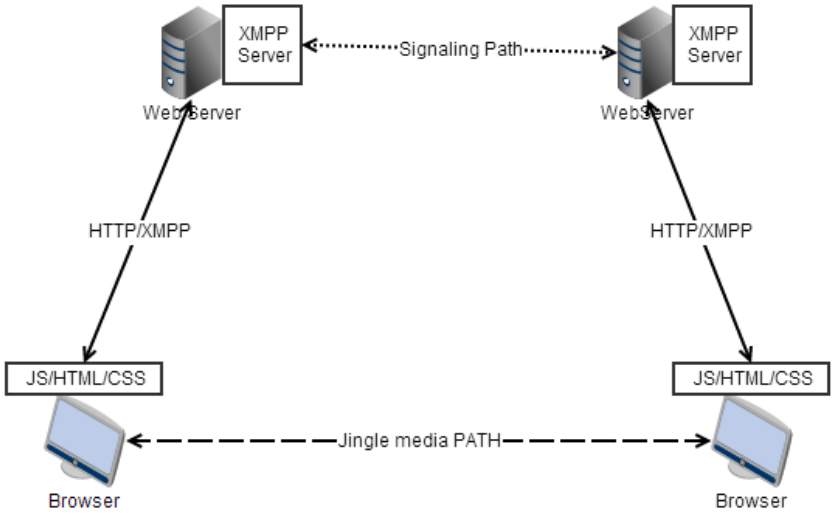


Figura 2.4: Processo *Jingle* [23].

2.5.3 *JavaScript Session Establishment Protocol (JSEP)*

O JSEP define de que forma as aplicações *JavaScript* interagem com as funções *Real Time Communication (RTC)*. Este protocolo é responsável pela forma como os clientes podem recolher informação sobre o tipo de *media* que suportam e que *codecs* usam, o que fica definido num objeto *SDP RTCSessionDescription*. O *JSEP* fornece mecanismos de criação de *offers* e *answers*, como também de que forma podem ser aplicadas numa sessão [24]. Este protocolo não define de que forma essa informação é trocada, ou seja, não é definido como essa informação é enviada ou recolhida do servidor *web*, ficando isso a cargo do programador.

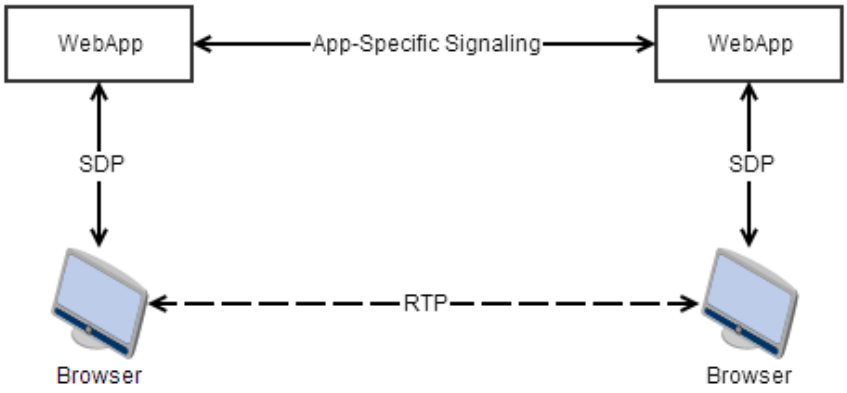


Figura 2.5: Modelo *JSEP* [24].

A figura 2.5 mostra o modelo utilizado pelo *JSEP*, onde as mensagens entre o *browser* e a

aplicação são objetos *SDP*, e o protocolo que é usado para transmitir essa informação é especificado na camada aplicacional.

2.6 *Network Address Translation (NAT)*

Comparativamente com o protocolo *Session Initiation Protocol (SIP)*, *WebRTC* tem problemas *NAT* e *Firewall*, para isso são usados os mesmos protocolos (*Interactivity Connectivity Establishment, Session Traversal Utilities for NAT e Traversal Using Relays around NAT*), para resolver alguns desses problemas.

2.6.1 *Interactive Connectivity Establishment (ICE)*

ICE é uma solução para *streams* de *media* baseadas em *User Datagram Protocol (UDP)*, que sejam estabelecidas num modelo pedido/resposta. Este protocolo permite a vários clientes trocarem informação *media* através de *NAT*, sendo assim, estes conteúdos têm incluídos endereços *IP* e portos, que por norma são problemáticos no que diz respeito sistemas com *NAT*.

O *ICE* é um protocolo que usa uma técnica conhecida como *"hole punching"*, que foi definida para que os utilizadores possam trocar informação *media* entre eles na presença de *NAT*. Esta técnica nem sempre é fiável e poderá falhar e para este tipo de situações o protocolo *ICE* tira proveito das funcionalidades do protocolo *TURN*.

No que diz respeito à tecnologia *WebRTC*, o protocolo permite a troca de *media* entre dois clientes que estejam por trás de *NAT*. Adicionalmente contém um método de verificação de comunicação, que permite prevenir ataques *Denial of Service (DoS)*, ou seja, isto acontece porque não é enviada media enquanto a informação *ICE* não seja trocada [25].

2.6.2 *Session Traversal Utilities for NAT (STUN)*

Session Traversal Utilities é um protocolo normalizado que permite lidar com *NAT*. Este protocolo fornece um mecanismo que permite a um cliente descobrir o endereço *IP* e o porto alocado por *NAT*. Este porto e o endereço *IP* privado permitem que a ligação *NAT* permaneça ativa. As funcionalidades principais deste protocolo verificam conectividade entre dois agentes ou permitem a retransmissão de pacotes de dados entre dois agentes. Este protocolo é usado na tec-

nologia *WebRTC* para estabelecer uma sessão entre dois clientes, onde o *web browser* funciona como cliente STUN e o servidor web como servidor STUN, por isso, são enviados à priori, pacotes de teste para estabelecer uma sessão, de forma a descobrir o caminho que se encontra por trás de *NAT*, de endereços *IP* e portos mapeados [26].

2.6.3 *Traversal Using Relays around NAT (TURN)*

É possível a partir de técnicas *"hole punching"* descobrir um caminho direto de um cliente para o outro, atravessando *NAT* como faz o protocolo *ICE*. Este tipo de técnicas podem falhar caso os clientes que estejam por trás de *NAT* não tenham um comportamento bem definido. Quando isto acontece, não é possível encontrar um caminho direto entre dois clientes, é necessário recorrer a serviços de um host intermediário que vai agir como um *relay*. Normalmente este *relay* situa-se numa rede pública (*Internet*) e retransmite os pacotes entre os dois clientes, que se encontram por trás de *NAT*.

Este tipo de especificação define o protocolo *TURN*, em que o seu funcionamento passa por, um cliente *TURN* fazer um pedido a outro cliente para que este aja como um servidor *TURN*, de forma a que os pacotes que estão a ser transmitidos, possam ser reencaminhados para o seu destino. No caso do *WebRTC* o *web browser user agent* inclui um cliente *TURN* e um servidor *TURN*. É feito um pedido ao servidor *TURN*, de um endereço *IP* público e um porto que irá funcionar como endereço de um *relay* de transporte [27].

2.7 *Codecs*

Ao nível aplicacional existem mecanismos adaptativos para lidar com a qualidade de serviço e a degradação da mesma, exemplo disso são os *codecs* de vídeo e áudio.

No desenvolvimento destas aplicações, a escolha dos protocolos de transporte e dos mecanismos de compressão de voz e vídeo tem de ser rigorosa, por forma a obter o melhor desempenho fim-a-fim. Em termos de *hardware* são usados codificadores e descodificadores, que por sua vez poderão executar *software* baseado em algoritmos de compressão e descompressão.

Para garantir a interoperabilidade entre os vários clientes *WebRTC*, foi necessário definir requisitos, no que diz respeito a *codecs* áudio e vídeo. Os clientes *WebRTC* que são desenvolvidos devem garantir como requisitos de áudio *Pulse Code Modulation (PCMA/PCMU)*, *telefone event*

e *Opus* e *G.711* como *codecs* de áudio. No que toca aos *codecs* de vídeo devem ser garantidos 10 *frames* por segundo e uma resolução mínima de 320x240, suportando também resoluções 1280x720, 720x480, 1024x768, 800x600,640x480 e 640x360 [28]. *RTCWeb* definiu dois *codecs* áudio obrigatórios: *Opus* e *G.711*. No entanto, ainda não foram definidos quais os *codecs* de vídeo obrigatórios, dependendo assim, da implementação do *web browser*. A tabela 2.3 mostra quais os *codecs* que cada um dos *web browsers* usa, tanto áudio como vídeo.

Tabela 2.3: Tabela de suporte *codecs* nos diferentes *web browsers* [29].

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

2.8 Sumário

Neste capítulo descreveram-se, de um modo geral, a arquitetura *WebRTC*, as *APIs* que constituem essa tecnologia (*PeerConnection*, *Media Streams*, *RTC Data Channels*) e de que forma podem ser usadas. Abordou-se a compatibilidade desta tecnologia e também da tecnologia *HTML5* nos *web browsers*, visto que por norma são usadas, atualmente, em conjunto para desenvolver aplicações *web*.

Foram discutidos alguns métodos que podem ser usados para a sinalização e de que forma podem ser transportados de uma aplicação para outra, para haver negociação de parâmetros. Por fim descreve-se o protocolo *WebSockets* e de que forma se diferencia dos tradicionais protocolos que os servidores *web* usam.

Discutiu-se ainda o problema de *NAT* em WebRTC e quais os protocolos que ajudam a resolver esse problema, como *ICE*, *TURN* e *STUN*. A questão relacionada com os *codecs* de áudio e vídeo também foi abordada, visto que os diferentes *browsers* podem usar diferentes *codecs*, e deve haver uma interação e negociação para que possa haver interoperabilidade. Discutiram-se ainda alguns protocolos usados para a troca de media e questões de segurança, protocolos como *SDP*, *SRTP* e *DTLS*.

Capítulo 3

Soluções *WebRTC*

Neste capítulo serão apresentadas algumas soluções WebRTC, com funcionalidades diferentes, mostrando de uma forma mais abrangente, como é que as diferentes *APIs* de *WebRTC* podem ser usadas e em que soluções devem ser implementadas.

De seguida serão analisadas algumas bibliotecas *JavaScript* existentes que permitam a implementação de serviços de uma forma mais simples, e que podem ser integradas para o desenvolvimento da *Framework* e da prova de conceito. Serão introduzidas/descritas plataformas de *WebSockets*, que terão potencial e poderão ser usadas para transportar informação de sinalização entre os diferentes clientes. Por fim são analisados *gateways SIP*, que permitem a interoperabilidade entre este tipo de soluções e clientes *SIP* que não são soluções *web*.

Na última secção, são apresentados alguns estudos realizados com as tecnologias apresentadas durante este capítulo. São apresentadas também as decisões tomadas relativamente a esses estudos e a análise realizada no primeira fase deste projeto.

3.1 Soluções existentes

3.1.1 *PeerCDN*

Esta é uma solução de entrega de conteúdos *peer-to-peer* e tem por objetivo reduzir o uso de largura de banda e do servidor. Permite a partilha de ficheiros não só *server-to-peer*, mas também *peer-to-peer*, isto é, pode-se imaginar uma rede criada entre os peers que estão ligados ao servidor e entre si, partilhando conteúdo entre si. Assim é reduzida em grande parte a largura

de banda utilizada na parte do servidor, reduzindo assim os custos [30]. Nesta solução é usada a *API WebRTC DataChannel*, para estabelecer a troca de dados entre os vários *peers* e é totalmente desenvolvida em *JavaScript* [30].

3.1.2 *Conversat.io*

Conversat.io é uma aplicação *web* que permite realizar chamadas entre duas a seis pessoas. O funcionamento é muito simples, a aplicação cliente liga-se a uma porta específica do servidor (*socket.io*) que permite executar a parte de sinalização e negociação entre os *peers*. Nesta aplicação é pedida a permissão para aceder aos dispositivos internos do computador (câmara e vídeo), de forma a poder alocar as streams associadas a cada um deles, num objeto *HTML5* vídeo ou áudio. É também criado um *"room"*, caso não exista, e a pessoa junta-se a este, onde estarão outras pessoas para entrar em contacto [31]. É uma solução que usa uma biblioteca *JavaScript* (*SimpleWebRTC.js*) e que permite a criação de aplicações para conferência multiutilizador de vídeo, de uma forma simples e rápida [31].

3.1.3 *BananaBread*

É um jogo em *3D* que corre totalmente em *web browser*, o seu desenvolvimento é realizado num motor multiplataforma *Cube 2*: *Sauerbraten*, que está escrita em *C++* e *OpenGL*, onde é usado *Emscripten* para compilar para *JavaScript* e *WebGL*. Desta forma é possível correr a aplicação nos *web browsers* modernos usando *web APIs*, sem qualquer tipo de *plugin* adicional. Para além destas características usa *WebRTC Data Channels*, para permitir a funcionalidade de *Multiplayer*, com suporte a dados binários [32].

3.1.4 *Responsive Web Typography with WebRTC*

Esta solução tira proveito da tecnologia *WebRTC* para criar aplicações *web* que sejam responsivas. Assim, é possível aumentar e diminuir o tamanho de letra de uma aplicação *web*, analisando apenas a distância da cara das pessoas da camara de vídeo do computador.

Efetivamente, quem desenvolve este tipo de soluções, acaba por dizer aos utilizadores de que forma devem ser usadas nos diferentes dispositivos [33], contudo é necessário ter em atenção outros fatores como, por exemplo, meio ambiente onde o utilizador se encontra, que tipo de

necessidades o utilizador tem, etc. O *Responsive Web Typography with WebRTC* em específico concentra-se principalmente na distância a que os utilizadores se encontram do dispositivo, isto é, ser possível criar uma solução em que o tamanho da letra e dos objetos se adaptem à distância entre o utilizador e o ecrã do computador. Para isso é usada a *API getUserMedia*, para recolher esse tipo de informação e criar um sitio *web* completamente responsivo [33].

3.1.5 *SIPML5*

sipML5 é o primeiro cliente *HTML5 SIP open source* desenvolvido em *JavaScript*, com integração em redes sociais como *Facebook*, *Google* +, *Twitter*. Este cliente pode ser usado em qualquer *web browser*, para uma ligação a uma rede *SIP* ou *IP Multimedia Subsystem (IMS)*, permitindo realizar e receber chamadas vídeo ou voz [34].

Este cliente suporta as seguintes funcionalidades:

- chamadas vídeo e áudio;
- mensagens instantâneas;
- chamada em espera;
- transferência de chamadas;
- *multi-line* e *multi-account*;
- modos de tons em *Dual-Tone Multi-Frequency (DMTF)* usando *SIP INFO*.

Encontra-se disponível para *Chrome*, *Firefox Nightly*, *Firefox stable*, *Internet Explorer (IE)*, *Opera* e *Safari*, apesar de que os mais aconselháveis são *Google Chrome* e *Firefox stable*. Nos restantes é necessário instalar a extensão *webrtc4all*. Juntando *webrtc2sip* com o *sipML5* [34, 29] consegue-se ter uma ligação a uma rede *SIP* ou a uma rede *Public Switched Telephone Network (PSTN)* para comunicar com outros dispositivos.

3.2 Bibliotecas *JavaScript*

Apesar de *WebRTC* ser uma tecnologia recente, há já vários esforços na implementação de soluções para vários contextos. Algumas soluções têm por base bibliotecas *Javascript* que foram

programadas de forma a facilitar o desenvolvimento de aplicações. Nesta secção serão apresentadas três bibliotecas estudadas e que poderão ser úteis para o desenvolvimento deste trabalho.

3.2.1 *WebRTC Experiment*

É um repositório de bibliotecas *JavaScript*, que implementa demos WebRTC e que tem como objetivo apoiar o desenvolvimento de aplicações deste tipo. Existem quatro bibliotecas diferentes neste repositório *RTCMultiConnection.js*, *DataChannels.js*, *RecordRTC.js* e *RTCall.js* [35].

RCMultiConnection.js

Com esta biblioteca é possível desenvolver alguns serviços avançados de uma forma mais simples, como [36]:

- conferências áudio/vídeo;
- partilha de dados ou de ficheiros;
- partilha de ecrã;
- renegociação de *streams* múltiplas e remoção de *streams* individuais;
- fazer *mute* ou *unmute* às várias *streams* (áudio e vídeo);
- banir utilizadores;
- deteção de presença (orientado a eventos onde é verificado quando um utilizador entra ou sai, sem vários estados de presença).

Esta biblioteca usa *firebase* por defeito, que é uma tecnologia com uma base de dados na *cloud* e que permite desenvolver aplicações em tempo real. Tem uma estrutura de dados partilhada e sincronizada, desta forma sempre que os dados são mudados ou atualizados, é possível atualizar essa informação em todos os clientes que estão ligados [36].

Para além disso, é possível implementar uma solução que use *socket.io* ou *WebSockets*.

DataChannel.js

É uma biblioteca *JavaScript* que foi desenhada para ajudar a desenvolver aplicações para partilha de dados. Estes dados podem ser ficheiros ou simples mensagens de texto. Tal como a *RTCMultiConnection.js* simplifica o desenvolvimento de aplicações. Esta biblioteca implementa as seguintes funcionalidades [37]:

- mensagens de texto diretas entre utilizadores;
- banir ou rejeitar qualquer utilizador;
- sair de uma sessão ou encerrar uma sessão completa;
- tamanho de ficheiros ilimitado;
- quantidade de dados ilimitada;
- deteção de presença (orientado a eventos onde é verificado quando um utilizador entra ou sai, sem vários estados de presença).

Como indicado na biblioteca *RTCMultiConnection.js*, usa por defeito e como *backend* o *firebase*, mas também pode usar *socket.io* ou *Websockets* para sinalização.

RecordRTC.js

Esta biblioteca foi desenvolvida para permitir aos utilizadores guardarem áudio, vídeo ou até mesmo imagens animadas. A *stream* de áudio é gravada no formato *.wav*, a *stream* de vídeo é guardada em formato *WebM* e as imagens animadas em *.GIF*. Estes formatos podem ser gravados no disco de forma a ser possível retornar o *URL* desse ficheiro, para aceder posteriormente a essa informação. Fica a cargo de quem está a desenvolver a aplicação dizer se quer como retorno o *URL*, *DataURL* ou o objeto *Blob* [38]. Esta biblioteca pode ser útil para gravar chamadas entre clientes, ou até mesmo gravar uma mensagem de *videomail* ou *voicemail*.

RTCCall.js

Esta biblioteca foi desenvolvida para ter apenas uma funcionalidade. Esta funcionalidade passa por um administrador de um sistema poder falar com os seus clientes/visitantes. Como exemplo,

um administrador de um sitio *online*, onde são oferecidos serviços, poderá ter um botão em que os seus clientes clicam e telefonam para si. Desta forma é possível tirar dúvidas sobre produtos ou até mesmo reportar problemas encontrados. Assim há uma maior interatividade entre administradores e clientes [39].

3.2.2 *SimpleWebRTC*

É uma biblioteca *Javascript* usada para o desenvolvimento de aplicações *WebRTC*. É possível configurar variáveis, para simplificar a implementação das soluções que se pretendem desenvolver, como saber quais os objetos *HTML5* aos quais se devem alocar as *streams*, se o pedido para permissão de uso de câmara e microfone deve ser feito automaticamente ou não. Para além disso, é uma biblioteca orientada a eventos, que permite saber se o utilizador está pronto e se está à espera que alguém se junte à sessão [40]. É necessário usar um servidor *socket.io*, que servirá como servidor de sinalização, que está implementado num servidor *Web* fornecido pelos criadores deste repositório [40].

3.2.3 *Adapter.js*

As chamadas aos sistema de cada *web browser* são feitas de forma diferente, ou seja, são necessárias executar funções diferentes na mesma aplicação caso se queira que a aplicação funcione em dois ou mais *browsers* diferentes.

A tabela 3.1 representa o tipo de funções usadas para desenvolver uma solução *WebRTC* em diferentes *web browsers*. Esta biblioteca vem unificar estas funções, isto é, ao carregar esta biblioteca para uma aplicação é possível usar funções *W3C Standard* que possam ser usadas tanto no *Chrome* como no *Firefox*.

Tabela 3.1: Perfixos das interfaces nos diferentes *browsers* [41].

W3C Standard	Google Chrome	Mozilla Firefox
getUserMedia	webkitGetUserMedia	mozGetUserMedia
RTCPeerConnection	webkitRTCPeerConnection	mozRTCPeerConnection
RTCSessionDescription	RTCSessionDescription	mozRTCSessionDescription
RTCIceCandidate	RTCIceCandidate	mozRTCIceCandidate

3.3 WebSockets

Como já foi referido anteriormente, não existe nenhuma especificação normalizada de como é feita a sinalização entre clientes. A definição de como este processo é feito fica ao nível da camada aplicacional, isto é, quem desenvolve a aplicação define um protocolo de sinalização, para essa aplicação. Visto que são aplicações que normalmente precisam de manter a informação sempre atualizada, deve ser utilizada uma solução que faça com que isso seja possível. Como já foi referido na Secção 2.4, *WebSockets* é a solução mais aceitável e que a maior parte das aplicações deste tipo usa, devido às suas características. Existem vários tipos de implementação de um servidor *WebSockets*. Nesta secção serão apresentados três tipos e serão descritas as características de cada um deles.

3.3.1 *Node.js*

É uma plataforma orientada a eventos, do lado do servidor, que permite a criação de aplicações web escaláveis. Os programas que são desenvolvidos com esta plataforma são escritos em *JavaScript* e são assíncronos, o que permite diminuir o *overhead* e aumentar a escalabilidade [42].

Contém uma biblioteca de servidor HTTP, que permite correr aplicações web sem usar um servidor típico HTTP (ex. Apache). Esta plataforma permite a criação de aplicações que tenham por objetivo funcionar completamente em tempo real, como por exemplo, *streaming* de voz ou vídeo, ou então serviços de chat, partilha de ficheiros [42].

Implementa um repositório de módulos, onde cada um tem uma funcionalidade diferente. Cada um destes módulos pode ser instalado a partir de *Node Package Manager (npm)* [43]. O *npm*, para além de servir para instalar os módulos, permite também tratar da gestão de versões e gestão de dependências. Um dos módulos que permite desenvolver aplicações que atuem em tempo real e que utilizem uma *API* tipo *WebSockets*, é o *socket.io*. Este permite que *WebSockets* e tempo real estejam presentes em qualquer *browser*, para além disso ainda fornece *multiplexing*, escalabilidade horizontal e codificação e descodificação em *JavaScript Object Notation (JSON)* [43].

3.3.2 *Vert.x*

É uma *framework* da próxima geração, que corre em *Java Virtual Machine (JVM)*, é orientada a eventos e assíncrona. Implementa um modelo de concorrência assíncrono que permite a criação de aplicações facilmente escaláveis, de uma forma simples. No que diz respeito às linguagens de programação que podem ser usadas, é uma *framework* poliglota, suportando desenvolvimento em: *Java*, *JavaScript*, *Groovy*, *Ruby*, *Python* e *Scala*.

Para além das características apresentadas anteriormente, esta *framework* utiliza alguns conceitos que são importantes para o desenvolvimento de aplicações [44, 45], nomeadamente:

- *Verticle*: Um *verticle* é definido por ter uma função principal (*main*), que corre um *script* específico. Um *verticle* pode conter mais *verticles*, desde que sejam referenciados na função *main*.

- *EventLoops*: São *threads* que estão sempre a correr e estão sempre à escuta, de forma a verificar se existem operações a executar ou dados a tratar. A gestão destas *threads* é feita por uma instância do *Vert.x*, que aloca um número específico de *threads* a cada núcleo do servidor.

- *Work Verticle*: Este tipo de *verticle* não é atribuído a uma *thread event loop* do *Vert.x*, em vez disso, é executado a partir de uma *thread* que se encontra numa *pool* de *threads* internas, à qual se chama *background pool*.

- *EventBus*: Esta é uma característica do *Vert.x* bastante importante que permite a comunicação entre vários *verticles* ou módulos do *Vert.x*. Para além disso, é possível haver uma comunicação não só entre *verticles* mas entre entidades que registem *handlers* num servidor (ex. clientes). Desta forma é possível a partilha de informação entre as várias entidades que constituam um sistema, chamada de *Message Passing*.

- *Modules*: É possível a criação de módulos individuais, que executem diferentes funções. Isto permite uma melhor organização de código e uma maior escalabilidade de aplicações. A comunicação entre os módulos é feita a partir de um *eventbus*. Existe ainda um repositório onde existem módulos previamente criados, que podem ser usados para facilitar a construção de aplicações.

Pode-se verificar na Figura 3.1 uma arquitetura específica do *Vert.x*.

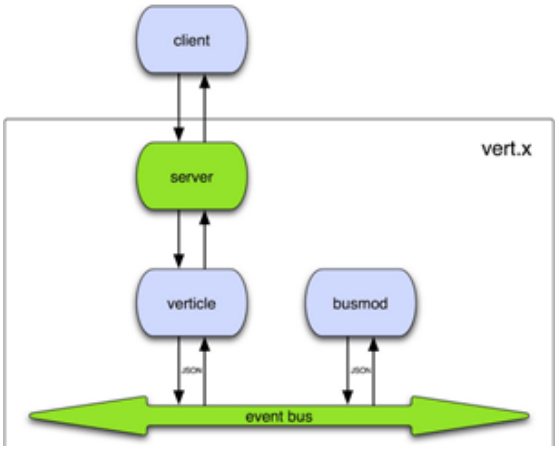


Figura 3.1: Funcionamento geral da aplicação Vert.x [46].

Em termos de *core*, são fornecidos vários serviços e métodos que são chamados diretamente em *verticles* e que podem ser implementados em *modules* [44].

- *HTTP/HTTPS servers and clients*
- *WebSockets servers and clients*
- *Accessing the distributed event bus*
- *Periodic and one-off timers*
- *Buffers*
- *Flow control*
- *Accessing files on the file system*
- *Shared map and sets*
- *Logging*
- *Accessing configuration*
- *Writing SockJS servers*
- *Deploying and undeploying verticles*
- *TCP/SSL servers and clients*

3.3.3 *Google App Engine*

Google App Engine é uma ferramenta que permite correr aplicações web na infraestrutura da *Google*. Normalmente quando se desenvolve uma aplicação é necessário a existência um administrador que trate de manter os serviços sempre ativos, que garanta que o servidor não vai abaixo, etc. *App Engine* é requisitado como sendo um serviço pago à empresa Google que garante redundância, gestão de serviços, bastando fazer *upload* para esta ficar a correr no servidor. Este serviço suporta várias linguagens de programação tais como: *Java*; *Python*; *PHP*; *Go*.

Para sistema de gestão de base de dados e de armazenamento são usados *Google Cloud SQL* e *Google Cloud Storage*. Estas características permitem criar aplicações fiáveis, mesmo que haja uma carga pesada e muitos dados a executar, além disso são oferecidas algumas características, como [47]:

- serviço de web dinâmico com suporte a tecnologias web comuns;
- sistema de base de dados com consultas, transações, ordenação, balanceamento de carga e escalabilidade;
- *APIs* com suporte a autenticação e possibilidade de enviar e-mails com conta google;
- possibilidade de simular *Google App Engine* numa máquina local;
- lista de tarefas para realizar trabalhos fora do âmbito web;
- tarefas agendadas para disparar eventos a uma hora especifica e em intervalos regulares.

Para além disso permite um nível elevado de segurança, onde é fornecido um acesso limitado ao sistema operativo. Assim é possível distribuir os pedidos por vários servidores, fazendo com que eles possam ou não conhecer as exigências de tráfego. Consegue ainda isolar uma aplicação no seu ambiente de segurança, que é independente do *hardware* e do *software*, e também da localização física do servidor *web* [47].

3.4 *Gateways* para *SIP*

Existem *Gateways* que permitem a comunicação de clientes totalmente *WebRTC* com clientes *SIP*, isto é, tornam praticável passar de um ambiente totalmente *web* para um ambiente total-

mente *SIP* e vice-versa. Este tipo de soluções pode ser uma mais valia no mundo das telecomunicações, permitindo uma interoperabilidade entre os vários sistemas existentes. Nesta secção serão apresentados e analisados alguns *Gateways SIP*.

3.4.1 *Asterisk*

Asterisk é uma *framework opensource* para o desenvolvimento de aplicações para comunicações. É capaz de tornar um computador num servidor de comunicações, implementando sistemas de *IP Private Branch Exchange (PBX)*, *gateways VoIP*, servidores de conferência e outro tipo de soluções.

Esta *framework* é usada por pequenas empresas, grandes empresas, e até mesmo agências governamentais, em mais de 170 países, permitindo o desenvolvimento de multiprotocolos, aplicações de comunicações em tempo real com voz e vídeo. Fornece uma abstração da complexidade dos protocolos e tecnologias de comunicação permitindo criar produtos e soluções inovadoras. Como é um produto *opensource*, pode ser editado e modelado da forma que a pessoa ou empresa, que esteja a desenvolver, pretenda [48].

Na versão 11 do *Asterisk* foi adicionado o suporte à tecnologia *WebRTC* e criado o módulo *res_http_websocket*, que permite a programadores *JavaScript* desenvolverem soluções em que haja interação e comunicação entre essas soluções *WebRTC* e o servidor *Asterisk*, a partir de *WebSockets*. Dentro do módulo *chan_sip* foram adicionados *WebSockets* para permitir o uso de *SIP* como protocolo de sinalização. Para além disso, foi adicionado o suporte dos protocolos *ICE*, *STUN* e *TURN* ao módulo *res_rtp_asterisk* para que os clientes que estejam por trás de *NAT* tenham uma melhor comunicação com *Asterisk*.

Para segurança de comunicação RTP já existe a biblioteca *libsrtp* que foi implementada na versão 10 [49]. *SRTP* é um requisito de *WebRTC*, ou seja, sempre que se queira usar *WebRTC* com *Asterisk* é necessário que se instale o pacote relacionado com esta biblioteca, a entrega de *media* irá falhar [49]. Apesar destes módulos estarem implementados no *Asterisk* é necessário que sejam ativados, configurados devidamente e de uma forma individual.

3.4.2 *WebRTC2SIP (sipML5)*

O *Webrtc2sip* serve como um *gateway*, permitindo a existência de um *web browser* que funcione como um telefone/telemóvel, que possibilite a realização de chamadas, tanto áudio como vídeo

e, ainda, enviar *Short Message Service (SMS)*, sobre qualquer tipo de rede *PSTN* e *SIP*, como se pode verificar na Figura 3.2.

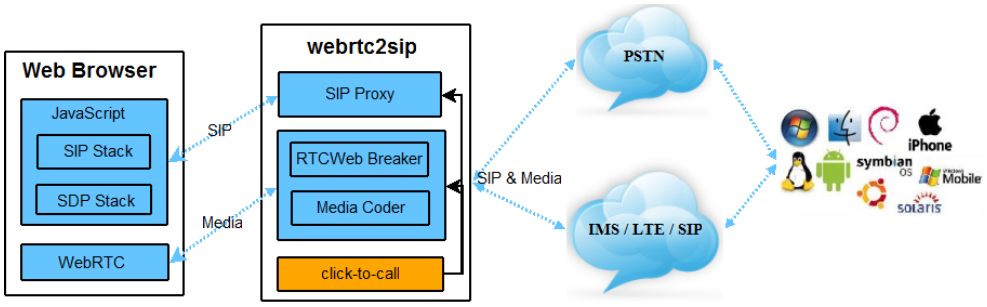


Figura 3.2: Arquitetura *SipML5* [29].

O *web browser* usa dois tipos de *stacks Javascript* que são *SIP Stack* e *SDP Stack*, que comunicam com *SIP Proxy*, sobre o protocolo SIP. No que diz respeito ao *WebRTC*, que trata da parte *media* das aplicações, este comunica com dois componentes do *webrtc2sip*, o *RTCWeb Breaker* e *Media Coder*. Portanto, o *RTCWeb Breaker* trata de fazer a conversão de *streams media*, que proporcione a comunicação com dispositivos finais que não suportem caraterísticas e protocolos como *ICE* e *DTLS/SRTP*. A figura 3.3 mostra como é feita essa conversão.

Figura 3.3: RTCWeb Breaker SipML5 [29].

A normalização de *RTCWeb* definiu dois *Mandatory To Implement (MTI) codecs* áudio, *opus* e *g.711*. Apesar de ainda existirem bastantes discussões sobre quais os *codecs* de vídeo a usar, ainda existe uma escolha entre *VP8* e *H.264*. *Chrome*, *Mozilla*, *Opera*, usam *VP8*, enquanto a *Microsoft* irá usar *H.264*. *Media Coder* vai permitir que se façam chamadas de vídeo entre os vários *web browsers*. A figura 3.4 mostra como é feita a conversão dos vários *codecs*.

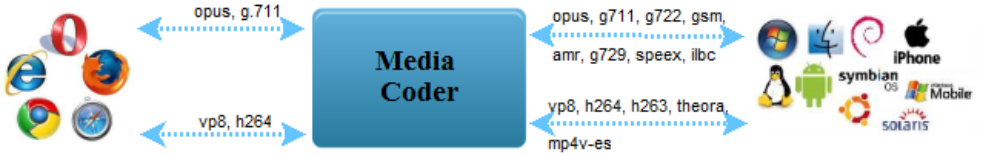


Figura 3.4: *Media Coder SipML5* [29].

3.5 Experimentação e seleção de soluções existentes

Como em todos os projetos, deve haver uma fase de estudo, investigação e definição de requisitos, para o desenvolvimento de uma solução capaz de forma organizada. Esta secção vem dar enfâse ao estudo das várias tecnologias analisadas, efetuando uma análise comparativa relativamente a estudos realizados anteriormente, de forma a ajudar a escolher as soluções mais eficazes e com mais potencial para o desenvolvimento deste projeto do mestrado. Será dada atenção tanto à parte de servidor como cliente, analisando diferentes soluções o que, por fim, levará à escolha da melhor.

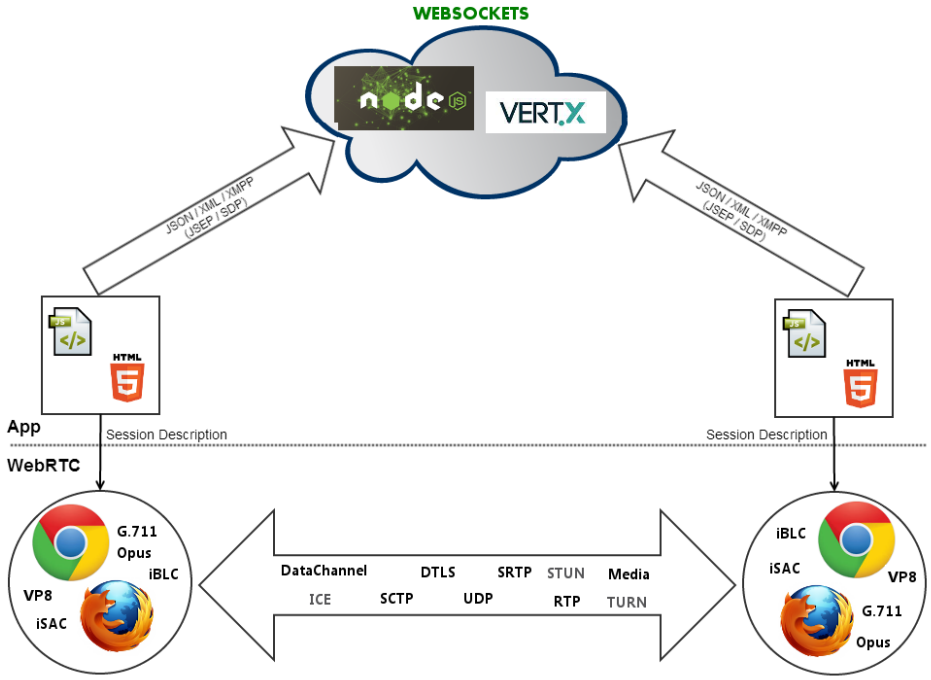


Figura 3.5: Esquema de interação de tecnologias.

Com vista à clarificação do papel de cada uma das soluções/tecnologias anteriormente estudadas no desenvolvimento da *framework WebRTC*, apresenta-se na Figura 3.5 um esquema ilustrando essas tecnologias e a sua interação.

3.5.1 *Vert.x vs Node.js*

Nesta secção serão analisadas soluções *WebSockets* que funcionarão, tanto na parte do servidor, como no cliente. Será feita uma análise comparativa, principalmente entre duas principais

plataformas: *Vert.x e Node.js*.

Nesta análise serão apresentados alguns resultados de estudos que já foram realizados anteriormente, comparando vários fatores desde a facilidade de desenvolvimento, o tipo de linguagem, a escalabilidade, o desempenho de servidores, etc.

Vert.x vs node.js simple HTTP benchmarks

O estudo realizado em [50] engloba dois tipos de testes diferentes, comparando o desempenho *HTTP* do *vert.x* e do *node.js*. Estes testes foram realizados apenas numa máquina com as seguintes especificações: AMD Phenom II X6; 8GB RAM; Ubuntu 11.04.

As versões do *vert.x* e do *node.js* são 1.0 e 0.6.6, respetivamente. Sendo *vert.x* uma plataforma poliglota, no que diz respeito às linguagens de programação, foram testadas *JavaScript*, *Ruby*, *Groovy* e *Java*. Assim também é possível fazer uma análise comparativa entre o desempenho desta plataforma nas diferentes linguagens.

O primeiro teste passa por medir o desempenho de um servidor que retorna uma mensagem 200-Ok a cada pedido do cliente.

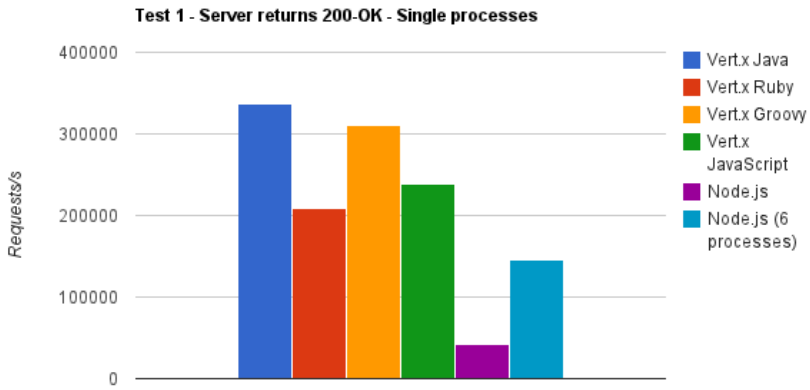


Figura 3.6: Resultados do teste de respostas 200-OK [50].

A Figura 3.7 representa os resultados depois dos testes. É possível verificar que o *vert.x* tem um desempenho mais elevado do que o *node.js*. Além disso apresentou um maior desempenho quando usada a linguagem *Java*, conseguindo responder a mais de 200000 pedidos que o *node.js* no mesmo espaço de tempo.

Um aspeto muito importante é que o *vert.x* é capaz de correr mais do que um *eventloop* num simples servidor, enquanto o *node.js* apenas consegue correr um, tendo a possibilidade de correr mais do que um processo num modo *cluster*. Nesse caso o *node.js* a correr com 6 processos consegue ter um melhor desempenho do que se corresse apenas num processo, no entanto comparando com o *Vert.x* continua atrás.

O segundo teste, passa avaliar a capacidade do servidor na transferência de um ficheiro de 72bytes. As especificações para correr este tipo de teste foram as mesmas que o anterior.

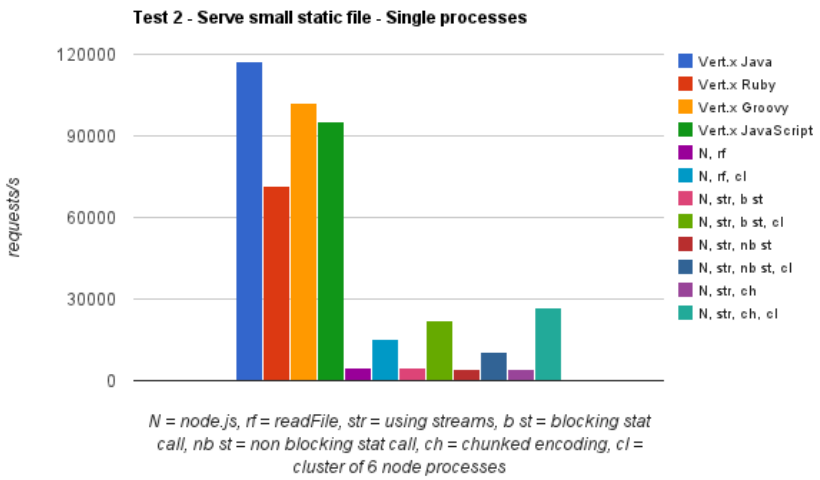


Figura 3.7: Resultados do teste de ficheiro 72bytes [50].

A Figura 3.7 representa os resultados obtidos na transferência de um ficheiro estático na resposta a pedidos. Pode-se verificar que mais uma vez o *vert.x* se destacou bastante pela positiva, tendo melhores resultados mais uma vez na linguagem *Java*, obtendo quase mais 100000 respostas que o melhor desempenho do *node.js*.

Comparação de *server side websockets* utilizando *atmosphere*, *netty*, *node.js* e *vert.x*

Outro teste realizado foi a comparação de *Websockets* usados no lado do servidor, não só para o *node.js* e *vert.x*, mas também com outro tipo de soluções: *netty* e *atmosphere*. No entanto neste documento, irão ser apresentados apenas os resultados do *node.js* e do *vert.x*, pois são as plataformas escolhidas como candidatas a serem usadas neste projeto. Este teste utilizou como caso de uso, um servidor *publish/subscribe* onde os clientes subscrevem para um determinado canal e recebem updates desses canais. Este caso de uso foi executado da mesma forma para

todas as plataformas.

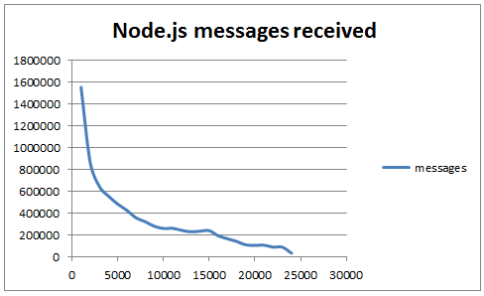


Figura 3.8: Mensagens recebidas - *node.js* [51].

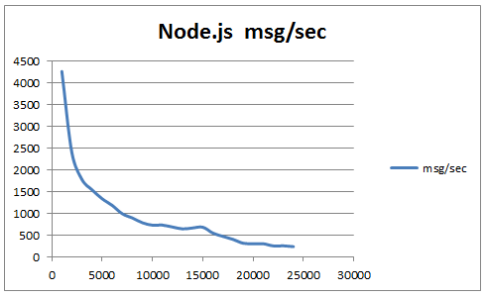


Figura 3.9: Mensagens recebidas por segundo - *node.js* [51].

A Figura 3.8 representa o número de mensagens recebidas pelo número de ligações, enquanto a Figura 3.9, representa as mensagens recebidas por segundo dependendo do número de ligações ativas. Conclui-se *node.js* conseguiu chegar às 24000 ligações, mas acabou por ser tornar lento e o cliente não conseguiu alocar mais memória para essas ligações.

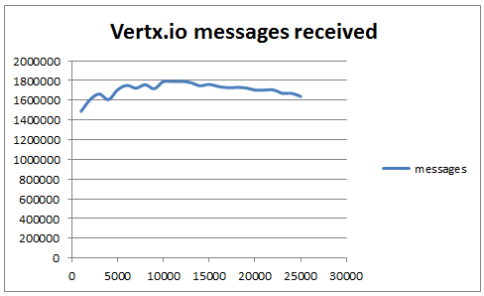


Figura 3.10: Mensagens recebidas - *vert.x* [51].

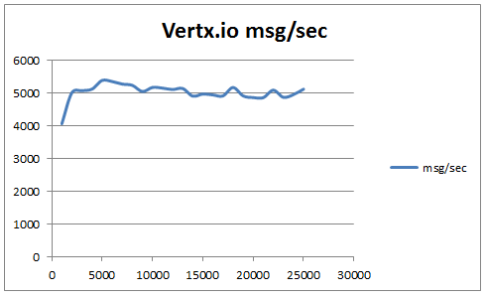


Figura 3.11: Mensagens recebidas por segundo - *vert.x* [51].

No caso do *vert.x* foi possível chegar às 25000 ligações, sem grande diferença, no que diz respeito a ficar lento, consegue, ainda assim, ter um tempo de resposta bastante linear sem grandes diferenças, mesmo que o número de ligações aumentem [51], como é demonstrado nas Figuras 3.10 e 3.11. No caso da plataforma *Atmosphere* não conseguiu lidar com mais de 3500 ligações, enquanto no *netty* a aplicação bloqueou nas 6500 ligações [51].

Para a escolha da tecnologia a usar para o desenvolvimento deste projeto, deu-se mais importância às seguintes características:

- escalabilidade;

- simplicidade;
- controlo de concorrência;
- possibilidade de escolha de linguagem de programação;
- orientação a eventos.

Para a *framework* e para a prova de conceito a desenvolver, pretende-se ter um desenvolvimento simples e eficaz. *Vert.x* é realmente uma plataforma simples, permitindo uma organização por módulos. Para cada um destes módulos pode ser designada uma função diferente, podendo haver módulos para controlo de ligações a um *WebServer*, para gestão de mensagens de base de dados ou gestão de autenticação. Tal perante uma melhor organização de código e da estrutura da arquitetura em si. Existe ainda a possibilidade de cada um destes módulos comunicar sobre um *eventbus* e, desta forma é possível uma interação entre os diferentes módulos. Para além disto mostrou, nos testes anteriores, um melhor desempenho em termos de escalabilidade e controlo de concorrência, em relação ao *node.js*. O *vert.x* permite ainda escolher entre várias linguagens de programação, suportando, como mencionando, *Python*, *JavaScript*, *Groovy*, *Ruby*, *Java* e *Scala*.

Conclui-se então que a melhor escolha seria a utilização do *Vert.x* para o desenvolvimento de produtos *WebRTC*.

3.5.2 Bibliotecas *JavaScript WebRTC*

Foi efetuada uma análise comparativa entre duas bibliotecas, apresentadas na Secção 3.2, relativamente aos serviços que é possível ter e qual a escalabilidade de cada uma delas.

WebRTC Experiment vs SimpleWebRTC

Como já referido anteriormente, *WebRTC Experiment* é um repositório que contém quatro principais bibliotecas: *RTCMultiConnection.js*, *DataChannels.js*, *RecordRTC.js* e *RTCall.js*. Cada uma delas tem um objetivo diferente e com estas bibliotecas é possível ter serviços como [38, 36, 37, 39]: conferências áudio/vídeo; partilha de dados ou de ficheiros; partilha de ecrã; renegociação de *streams* múltiplas e remoção de *streams* individuais; *mute* ou *unmute* às várias *streams* (áudio e vídeo); banir utilizadores; detetar de presença (orientado a eventos onde é verificado quando um utilizador entra ou utilizador sai, sem vários estados de presença); gravar mensagens

de vídeo e áudio; integrar um sistema de chamadas para ligar para um administrador (ex. administrador de um sitio web).

WebRTC é um projeto que continua em constante desenvolvimento e, por isso, nem todos estes serviços são suportáveis em todos os *web browsers*. Este repositório permite não só uma comunicação entre *web browsers* do mesmo tipo, como também de diferentes tipos, como *Chrome* e *Firefox*. Permite uma abstração sobre toda a *API* do *WebRTC*, fazendo com que a programação em *JavaScript*, do lado do cliente, seja mais simples e mais organizada.

A biblioteca *SimpleWebRTC* permite apenas fazer conferências áudio e vídeo com dois ou mais utilizadores e usa como base o módulo *socket.io*, *node.js* [40].

Em suma, *WebRTC Experiment* é o repositório mais completo, e mais flexível em termos de serviços, pois inclui uma maior diversidade de serviços que podem ser fornecidos. Fornece uma *API* de desenvolvimento simples, criando objetos *JavaScript* que concedem funções, para tratar de negociações de *media*, e permite uma *API* orientada a eventos. Usa como *backend firebase* para manter a informação sempre atualizada, apesar disso é possível a integração de *WebSockets over socket.io* e de *node.js over socket.io*.

Base de dados

Visto ser necessário guardar a informação da aplicação a desenvolver num sistema de base de dados, optou-se por usar algo que já estivesse implementado. Face à escolha do *vert.x* para implementar uma solução *WebSockets*, selecionou-se um sistema de base de dados não relacional, *MongoDB*. Esta escolha ocorreu devido à existência de um módulo *vert.x* já implementado, que contém todas as operações que se podem realizar nesta base de dados.

Linguagens de Programação

Em termos de linguagem de programação no lado do cliente não é possível muita escolha, pois a *WebRTC* permite o desenvolvimento de aplicações *web* a partir da linguagem *JavaScript*. Em termos do servidor existe uma oferta mais ampla, devido à escolha da plataforma *vert.x* para o desenvolvimento do servidor. Como foi referido anteriormente, a linguagem que mais se destacou nos testes realizados, foi o *Java*. Assim optou-se pelo uso de *Java* para o desenvolvimento da componente servidor, visto que irá permitir um melhor desempenho e escalabilidade.

3.6 Sumário

Sendo este capítulo referente ao estado da arte, foram descritas diferentes soluções já implementadas com a tecnologia *WebRTC*. Cada uma destas soluções tira proveito das funcionalidades das *APIs* referentes a esta tecnologia, demonstrando como funcionam e de que forma podem ser utilizadas.

Foram também abordadas algumas bibliotecas que facilitam a implementação de uma solução *WebRTC*, permitindo desenvolver serviços a um alto nível de programação. Neste contexto, foram descritas algumas plataformas *WebSockets* que podem ser usadas tanto no servidor como no cliente, como também *gateways SIP* que podem ajudar na implementação de soluções com interoperabilidade entre clientes *WebRTC* e clientes *SIP* externos.

Para finalizar, foi realizado um estudo comparativo de soluções existentes, para sustentar as decisões tomadas no desenvolvimento da *framework WebRTC* que a seguir se apresenta.

Capítulo 4

Framework Webrtc

Depois do estudo e seleção das tecnologias envolvidas em *WebRTC*, iniciou-se a fase de desenvolvimento da *framework WebRTC*. Definiram-se os requisitos funcionais para que a mesma seja implementada de acordo com o pretendido.

Neste capítulo serão apresentadas as especificações iniciais da *framework*, como também a sua estrutura arquitetural. De seguida, será feita uma descrição dos módulos desenvolvidos para a execução de funções, de forma a descrever a finalidade de cada um, bem como a interação entre todo o sistema. Será feita uma abordagem ao cliente desenvolvido e simultaneamente apresentar-se-ão algumas funcionalidades implementadas, tirando partido do que foi desenvolvido na *framework* e da biblioteca *RTCMultiConnection.js*, apresentada na secção 3.2.1.

4.1 Requisitos funcionais

O objetivo deste projeto centra-se na tecnologia *WebRTC* e no desenvolvimento de uma *framework*, que fosse testada numa aplicação. Esta secção apresenta os requisitos funcionais que foram definidos inicialmente para esta plataforma.

4.1.1 Chamada com controlo de estabelecimento de chamada

Este é considerado um requisito mínimo obrigatório, visto que este projeto é referente à tecnologia *WebRTC*, que tem por objetivo facilitar as comunicações em tempo real via *web*. Consiste na possibilidade de um utilizador conseguir realizar uma chamada áudio e vídeo simples, para

outro utilizador. Para tal é necessário haver uma ligação inicial com o servidor para que estes possam fazer a negociação da sessão, para estabelecer a chamada. Depois da chamada estar estabelecida entre os clientes, ambos podem desligar-se do servidor, pois a comunicação será feita *peer-to-peer*, com media *RTP*.

4.1.2 Chamada básica com controlo completo da chamada

Este requisito é uma extensão do anterior, ou seja, para além de ser preciso o servidor para estabelecer uma sessão entre dois clientes, é necessário que a sua ligação permaneça ativa durante essa chamada. Desta forma é possível saber quando a sessão é encerrada, por exemplo, para poder saber qual foi a duração da chamada, ou então para mudança de parâmetros dessa sessão que seja necessário enviar ao servidor.

4.1.3 Chamada com pessoas externas

Possibilidade de um cliente realizar uma chamada com uma pessoa que não esteja registada na base de dados do sistema, ou seja, quando é feito *INVITE* para iniciar a chamada, deve ser criado simultaneamente um *Uniform Resource Locator (URL)* específico. Este *URL* deve ser legível de forma a conseguir retirar informação suficiente sobre a sessão que foi iniciada, para que a pessoa externa consiga estabelecer uma chamada na mesma sessão.

Este requisito é uma extensão dos anteriores, dando a possibilidade de realizar uma conferência entre várias pessoas. Quando todas as pessoas estiverem dentro dessa sessão, podem-se desligar do servidor e continuar a falar. Também deve ser possível ter um controlo total desta sessão, ou seja, conseguir verificar a duração da chamada, e quem se encontra nessa sessão. Para adicionar pessoas externas, deve funcionar da mesma forma que a chamada com pessoas externas, isto é, quem se quiser juntar à conferência deve receber um *URL* específico, ao qual irá aceder para se conseguir juntar.

4.1.4 Presença e Lista de contactos

Estes serviços encontram-se na maior parte das aplicações de mensagens instantâneas e chamadas de voz e vídeo. É necessário garantir que um utilizador tem uma lista de contactos, para isso é essencial existir uma base de dados elaborada, onde fiquem guardadas as informações das pes-

soas que se encontram na lista de contactos. Associado ao contacto deve ser garantido um serviço de presença, que para cada pessoa na lista, permita saber os estados de presença em que uma pessoa se encontra (e.g. Ocupado, Disponível, Ausente, Offline). Sempre que o estado de uma pessoa muda, deve garantir que todas as pessoas da sua lista de contacto, recebam uma notificação de que esse estado foi alterado. Esta informação deve ficar guardada na base de dados / servidor de forma a garantir que a informação se encontra sempre atualizada.

4.1.5 Chat / Mensagens Instantâneas

O serviço de *chat* ou mensagens instantâneas, é encontrado maioritariamente em aplicações multimédia, *peer-to-peer* e até em redes sociais, sendo este definido como um requisito para este projeto. Desta forma, sempre que é realizada uma chamada / conferência, este serviço deve estar presente, assim, os utilizadores têm a possibilidade de enviar mensagens de texto para outros utilizadores que se encontrem na mesma sessão.

Existem duas abordagens diferentes em relação a este serviço:

- todas as mensagens trocadas entre os clientes passam pelo servidor, desta forma é possível, ter um controlo mais específico de todas as mensagens enviadas.

- é negociada uma sessão onde são criados *RTCDataChannels*, neste caso as mensagens não passam pelo servidor. É feita uma negociação inicial para ter uma ligação *Data Channel* entre clientes e desta forma é possível enviar mensagens *peer-to-peer*, mesmo que posteriormente os clientes se desliguem do servidor. Esta solução tira todo o proveito da tecnologia *WebRTC*.

4.1.6 Partilha de Ficheiros

Visto que a tecnologia *WebRTC* continua a crescer, começa a ser possível criar vários serviços, devido ao desenvolvimento dos *web browsers*, que começam a ter suporte à *API Data Channels*. Isto permite a criação de canais de dados entre clientes, tornando possível a transferência de dados e partilha de informação em tempo real. Por estas razões decidiu-se fazer deste serviço também um requisito para este projeto.

4.1.7 Gravação de Voz e Vídeo

Esta funcionalidade foi definida como um requisito e pode ser abordada de duas maneiras:

- ser possível a gravação total de uma chamada. Isto deve ser feito tanto para chamadas de áudio como para chamadas de áudio e vídeo. É uma funcionalidade que normalmente se encontra implementada por maior parte dos serviços que são fornecidos pelos operadores. Posteriormente é possível fazer uma revisão de todas chamadas que foram gravadas;
- esta funcionalidade pode servir para gravar mensagens de *voicemail* ou *videomail*. Também são serviços que se encontram disponíveis pelas operadoras, que permitem a gravação de mensagens, para que mais tarde possam ser ouvidas pelas pessoas que foram contactadas num momento e não se encontravam disponíveis.

4.2 Arquitetura da *Framework*

A preceder a fase de implementação foi necessário desenvolver uma arquitetura que permita obter uma *framework* organizada e funcional. Esta secção descreve a arquitetura que foi definida e as decisões que foram tomadas na especificação da mesma.

4.2.1 Especificações Iniciais

Como a tecnologia escolhida para os clientes e o servidor comunicarem foi o *vert.x*, definiu-se que as funcionalidades iam estar separadas por módulos, para que fosse possível ter uma estrutura organizada e escalável.

Figura 4.1: Especificação inicial dos módulos.

Na Figura 4.1, são apresentados, como exemplo, três módulos diferentes em que cada um executa uma ou mais funções diferentes. No caso em que a *framework* ou uma aplicação tenha

problemas na autenticação dos utilizadores, o programador sabe que o problema está no módulo 1, tornando assim possível alterar apenas esse módulo 1 sem que nada interfira com o módulo 2 e 3, que executam funções completamente diferentes.

Foi definido também, que os módulos poderiam comunicar entre si, sendo necessário criar um *eventbus*, como mostra a Figura 4.2.

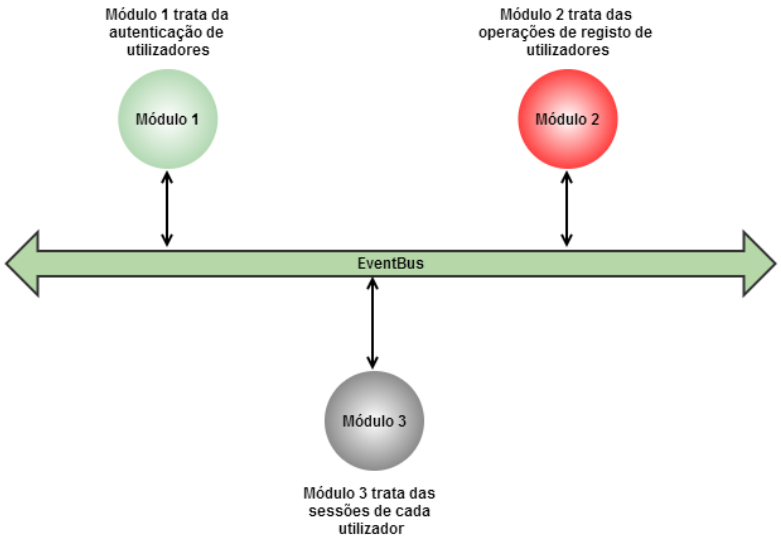


Figura 4.2: Especificação da interação entre módulos.

Num cenário em que os utilizadores precisem de fazer autenticação e ao mesmo tempo seja criada uma sessão associada a essa autenticação, o módulo 1 trata de toda a parte de verificação da autenticação, garantindo que existe aquele utilizador e que a sua *password* está correta. Depois disso, irá comunicar com o módulo 3 que trata da parte das sessões, isto é, sempre que um utilizador se autentique no sistema o módulo 1 irá comunicar com o módulo 3, a partir do *eventbus*, para que seja criada uma sessão associada aquele determinado utilizador.

Qualquer cliente *web* que seja desenvolvido não pode comunicar diretamente com o módulo da base de dados. É necessário que haja sempre um módulo intermédio, para o cliente interagir indiretamente com o módulo da base de dados.

Como no exemplo da Figura 4.3, se o cliente se quiser autenticar terá de comunicar com o módulo 1, que por sua vez se irá preocupar em comunicar com o módulo de base de dados, para verificar a autenticidade daquele cliente. No fim dependendo do que se encontra implementado,

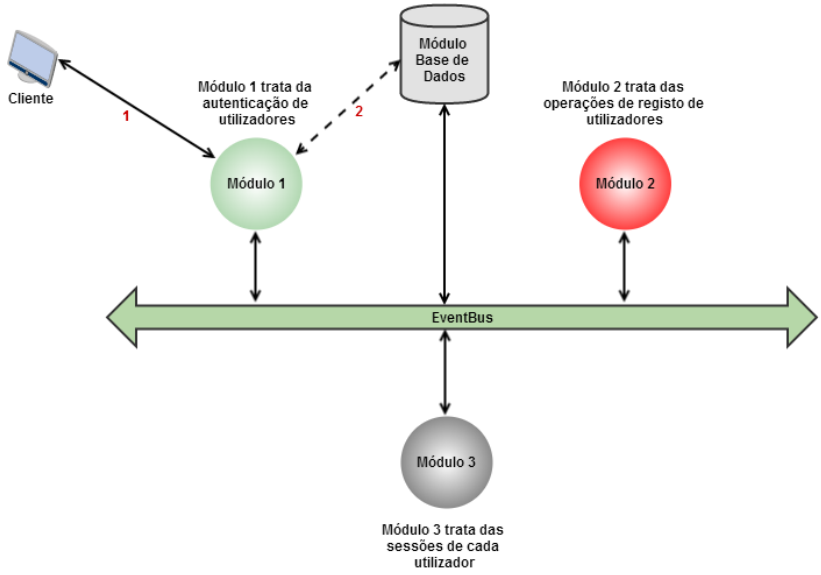


Figura 4.3: Especificação da interação dos clientes com módulo base de dados.

o cliente irá receber como resposta se a sua autenticação foi executada com sucesso ou não. Os módulos irão comunicar sempre entre si pelo *eventbus* existente.

4.2.2 Estrutura arquitetural

A arquitetura deste sistema consiste numa estrutura totalmente *web*, isto é, envolve a « existência de um servidor *web* ao qual clientes *HTTP* acedem para ter acesso aos recursos desse servidor. Podem-se considerar duas camadas, a camada da parte do cliente e a camada da parte do servidor, como está representado na Figura 4.4:

A parte do cliente tem por objetivo principal a comunicação com os módulos registados no *eventbus*, assim, poderá garantir uma melhor interação em tempo real entre cliente e servidor, de forma a que toda a informação se encontre sempre atualizada. Contudo, um cliente poderá fazer o registo de endereços/*handlers* no *eventbus* como qualquer módulo, para que possa existir interação direta entre clientes, sem que seja necessário a informação passar sempre por um módulo do servidor.

A forma como os clientes interagem com os módulos, para conseguirem executar as funções pretendidas é apresentada na secção 4.4.

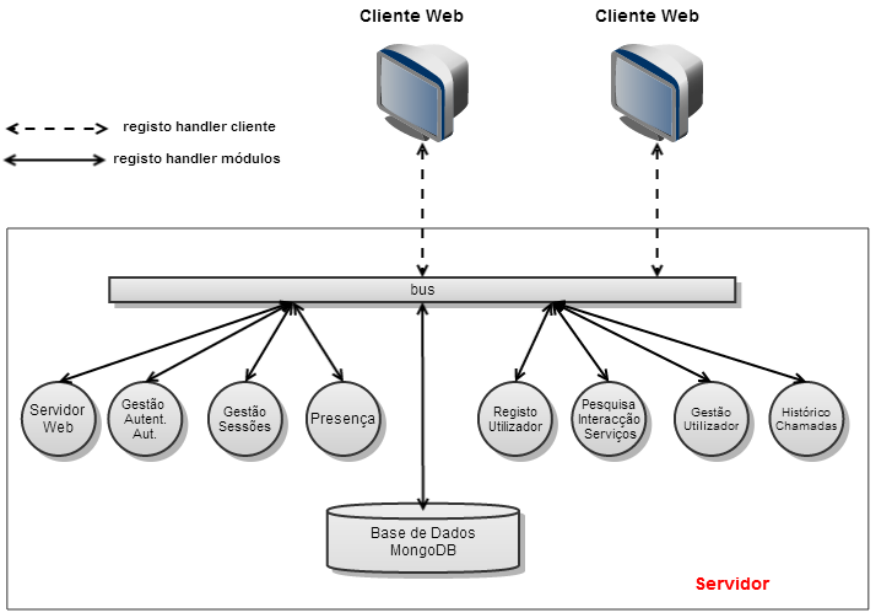


Figura 4.4: Arquitetura do sistema.

4.3 Descrição dos Módulos

Depois de definidas as especificações iniciais foi definida a arquitetura tanto a nível de componentes, como módulos e mensagens trocadas. Esta secção descreve os módulos escolhidos e a seguinte descreve a interação entre cada um deles.

4.3.1 Módulo servidor *web*

Vert.x é uma plataforma que nos consegue abstrair de plataformas como *Apache*, *PHP*, sistemas de gestão de base de dados *Mysql*, entre outros. O módulo servidor *web* cria um servidor onde os ficheiros são facilmente acedidos. Tem uma configuração bastante completa, onde são indicados os campos mais importantes para conseguir aceder ao servidor e aos ficheiros que lá se encontram. Os parâmetros de configuração passam por:

- *web_root*: pasta onde se encontram os ficheiros a que queremos aceder;
- *index_page*: a página índice de um sítio, ou seja, página inicial ao qual se deve aceder;
- *host*: endereço onde o servidor se encontra a ser executado;

- *port*: porto para escuta de pedidos e ligações;
- *ssl*: para suporte *https*, de forma a criar aplicações *web* mais seguras e confiáveis;
- *key_store_password*: *password* do ficheiro *Java Keystore*, que guarda o certificado do servidor;
- *bridge*: permite dizer se o servidor atua ou não como um *eventbus*;
- *inbound_permitted*: só é utilizado na existência do *eventbus* e permite dar autorização de entrada a objetos *JSON* no *eventbus*;
- *outbound_permitted*: só é utilizado na existência do *eventbus* e permite dar autorização de saída a objetos *JSON* no *eventbus*.

O programador é que define como estes campos devem estar configurados, pois só este sabe quais os requisitos ao qual o servidor deverá responder.

4.3.2 Módulo de base de dados

Como já foi referido anteriormente, este projeto passa pelo desenvolvimento de uma *framework*, que será testada ao mesmo tempo com o desenvolvimento de uma aplicação. Tanto a *framework* como a aplicação, teriam de interagir com uma base de dados. Foi escolhida um tipo de base de dados não relacional, *MongoDB* onde já existe implementado um módulo por parte da comunidade do *vert.x* e executa as seguintes funções:

- *save*: função do módulo que permite guardar documentos na base de dados, numa determinada *collection*;
- *update*: função do módulo que permite atualizar um determinado documento;
- *find*: função para encontrar um ou mais documentos na base de dados. Neste caso é necessário ter um campo chamado *criteria*, que corresponde ao que queremos encontrar na base de dados;
- *count*: função que recebe como resposta o número de documentos de um determinado tipo, que se encontre na base de dados;

- *findone*: função que encontra um documento específico na base de dados, ou seja, apenas é devolvido um documento;
- *delete*: função para apagar determinados documentos da base de dados.

Este tipo de base de dados implementa *collections*, que correspondem a tabelas de bases de dados relacionais e documentos que são o mesmo que registos de base de dados. Para além disso, os módulos que se pretendem usar têm te ter uma configuração prévia, caso contrário irão ser usados os valores por defeito. No caso deste módulo, pode ter como configuração os campos: *address* (endereço), *host*, *port* (porto), *pool_size, db_name* (nome da base de dados). Todas as funções que devem ser executadas neste módulo devem ser enviadas para o endereço que foi especificado na configuração, pois é o que se encontra registado como *handler* no *eventbus* usado.

4.3.3 Módulo de gestão de autenticações e autorizações

Este módulo faz uma gestão básica de autenticação de utilizadores no sistema, verificando se o utilizador e a *password* introduzidas estão corretas e, por conseguinte, tem de ter associado um módulo de base de dados, pois é onde se encontram registados os utilizadores. Quando a autenticação é executada, é devolvida como resposta um identificador de sessão que pode ser passado pelo *eventbus*, para outros módulos. Tem três operações simples:

- *login*: para executar o *login* é necessário enviar uma mensagem com o *username* e a *password*. Se for executado com sucesso é retornado um identificador da sessão;
- *logout*: para executar o *logout* de um utilizador é apenas necessário enviar o identificador da sessão e o módulo irá encarregar-se de executar o *logout*;
- *authorised*: envia o identificador da sessão para verificar se é ou não uma sessão autorizada.

Para executar estas três operações é necessário enviar uma mensagem para os seus endereços, ou seja, na configuração inicial do módulo são configuráveis os seguintes campos: endereço, *user_collection* (tabela dos utilizadores registados no sistema), *persistor_address* (endereço do módulo de base de dados) e o *session_timeout* (temporizador do fim de sessão). Por exemplo, se o endereço deste módulo for *"test_myauthmod"* então, para executar *login*, seria necessário

enviar uma mensagem para o endereço *"test_myauthmod.login"*, no caso do *logout* seria para o endereço *"test_myauthmod.logout"* e *authorised "test_myauthmod.authorise"*.

4.3.4 Módulo de gestão de sessões

Este módulo permite ao programador criar sessões e guardar informação nessas sessões. Para além disso, cada sessão tem um *timeout* associado e, sempre que expirar, toda a informação que se encontre nessa sessão será apagada juntamente com a sessão. Pode ser usado tanto com *SharedData* como com base de dados (*MongoDB*) para guardar a informação. Como os módulos apresentados anteriormente, apresenta vários campos configuráveis:

- *address*: endereço do módulo, para registar como *handler* no *eventbus*;
- *timeout*: o tempo que as sessões devem ficar guardadas;
- *cleaner*: endereço para o qual é enviada informação quando uma sessão é destruída;
- *prefix*: prefixo onde os clientes podem estar à escuta do seu identificador da sessão;
- *map-timeouts*: nome do *shared-map* para guardar o *timer* de identificadores;
- *map-sessions*: nome do *shared-map* para guardar informação sobre a sessão;
- *mongo-sessions*: configuração para quando é usada a base de dados (*MongoDB*);
- *address*: endereço do módulo do *MongoDB*;
- *collection-name*: nome da *collection* onde a informação sobre as sessões fica guardada.

Para ser possível criar, apagar e destruir sessões, é necessário executar algumas das seguintes operações:

- *start*: permite iniciar uma sessão recebendo como resposta um indentificador de sessão (*sessionId*) caso a sessão seja criada com sucesso;
- *destroy*: permite destruir uma sessão de imediato, a informação da sessão será enviada para o *cleaner* e o utilizador receberá uma mensagem de que a sua sessão foi destruída;
- *clear*: operação que destrói todas as sessões que estejam ativas;

- *heartbeat*: envia *heartbeats* periódicos para o servidor de forma a reiniciar o *timeout*, para que a sessão não seja destruída automaticamente;

- *get*: possibillita recolher informação sobre uma determinada sessão, enviando-lhe apenas o identificador de sessão e os campos da informação que se quer recolher;

- *put*: necessário para quando se quer guardar informação na sessão. Neste tipo de mensagem devem ir identificados os dados que se querem guardar na sessão;

- *status*: este tipo de mensagem tem dois subtipos de *reports*;

- *connections*: recebe como resposta o número de sessões que estão guardadas e ativas;

- *matches*: devolve em resposta o identificador de uma sessão, ao executar a pesquisa de uma determinada informação.

Este módulo irá servir para guardar informação sobre uma sessão de autenticação de um utilizador, mas também poderá servir para guardar informação sobre sessões de chamadas, como o tempo da chamada, quem foram os participantes, mensagens trocadas, etc.

4.3.5 Módulo de registo utilizador

Este módulo é responsável pelo registo dos utilizadores no sistema, visto que a aplicação cliente não pode interagir diretamente com o módulo de base de dados é necessário existir uma espécie de *middleware*. Este módulo tem apenas dois campos na sua configuração inicial:

- *address*: endereço para identificação do módulo no *eventbus*;

- *port*: porto no qual a aplicação está à escuta para receber pedidos.

Este módulo irá receber um pedido da aplicação cliente com os dados sobre o registo do utilizador no sistema, a partir da operação *registuser*. Este módulo ao receber esse pedido irá encaminhá-lo para o módulo de base de dados, a partir do *eventbus*, para executar o registo.

4.3.6 Módulo de pesquisa e intersecção de serviços

Este módulo é usado em paralelo com o módulo de autenticação e o módulo de gestão de sessões, é configurável como o de registo de utilizador, podendo modificar os campos endereço e porto. Um utilizador ao fazer autenticação com sucesso irá verificar os serviços disponíveis no terminal, que de seguida serão enviados por este módulo a partir da operação *register*. Este módulo é também um *middleware* que ao receber informação sobre os serviços que o terminal do utilizador suporta, irá verificar à base de dados quais os serviços que o cliente tem subscritos. Depois disto é feita uma intersecção entre os serviços que o utilizador suporta e que o terminal suporta. Como exemplo simples, é possível imaginar dois conjuntos de serviços, conjunto A e conjunto B, em que:

- A = vídeo, áudio, *chat*, partilha de ficheiros (serviços suportados pelo terminal);
- B = vídeo, *chat* (serviços suportados pelo utilizador).

Ao fazer a intersecção entre o conjunto A e conjunto B ($A \cap B$), o resultado seria vídeo, *chat* sendo que, durante essa sessão do utilizador, o cliente apenas poderia usar os serviços de vídeo e de *chat*.

4.3.7 Módulo de gestão de utilizadores

Este módulo serve essencialmente para pesquisa de pessoas no sistema, em que os campos configuráveis são:

- *endereço*: é o endereço para registar o módulo no *eventbus*;
- *porto*: no qual estará à escuta para receber pedidos.

Para realizar uma pesquisa de uma pessoa é necessário que, a partir da aplicação cliente, seja enviada para o servidor uma mensagem do tipo *"userquery"*. Esta *query* também é configurável, pois é possível enviar vários campos como parâmetros de entrada, podendo pesquisar a pessoa por nome, *e-mail*, *URL* público, etc.

Para além disso, o cliente pode definir quais são os parâmetros que quer receber em relação aos utilizadores que está a procurar, podendo receber os seus endereços de cada serviço, nome,

e-mail, o estado de presença, etc. Caso um utilizador queira adicionar uma pessoa do sistema à sua lista de contactos, terá de fazer uma pesquisa prévia. O mesmo acontece se o utilizador quiser adicionar uma pessoa a uma sessão de conferência.

4.3.8 Módulo de presença

Este módulo tem como objetivo tratar de todas as funcionalidades de presença dos utilizadores. Existem dois campos configuráveis:

- *endereço*: é o endereço para registar o módulo no *eventbus*;
- *porto*: ao qual estará à escuta para receber pedidos.

É um módulo de troca de mensagens entre cliente e servidor, pelo qual é possível executar três operações distintas. Uma das operações é *"get"*, que serve para buscar toda a informação sobre os estados de presença que se encontrem registados na base de dados.

A operação *"update"* pode ser usada quando um utilizador muda o estado de presença (e.g. disponível para ocupado), e tem por objetivo notificar todos os contactos que se encontrem online da sua mudança e também o servidor para que este consiga atualizar informação na base de dados. Por fim a operação *"subscribe"*, que é usada em paralelo com a função de pesquisa de pessoas, ou seja, quando se faz uma pesquisa de pessoas é possível adicionar essa pessoa à lista de contactos.

4.3.9 Módulo de histórico de chamadas

Este módulo tem como objetivo guardar informação relativamente a chamadas ou conferências realizadas. Tem dois campos configuráveis:

- *endereço*: é o endereço para registar o módulo no *eventbus*;
- *porto*: ao qual estará à escuta para receber pedidos.

É também um módulo de troca de mensagens entre cliente e servidor, isto é, é um módulo intermediário que receberá mensagens dos clientes e as encaminhará para o módulo de base de dados para guardar informação.

Este módulo tem um *eventhandler* que estará à espera de receber três tipos de eventos: *registcall*, *unregistcall*, *getcalls*. A mensagem *registcall* permite a gravação de dados na base de dados, relativos à chamada/conferência como, duração, participantes, data e tipo. Existem três tipos distintos de chamadas *attended*, *notattended* e *rejected*. A mensagem *unregistcall* permite ao utilizador apagar uma chamada do seu histórico. Por fim a *getcalls* permite ao utilizador buscar todas as chamadas do seu histórico, que se encontrem associadas ao seu perfil.

4.3.10 Sinalização

Em termos de sinalização é usada a tecnologia *socket.io*, que é usado para aplicações em tempo real, visto que para a implementação de serviços como conferência áudio e vídeo, chat e partilha de ficheiro recorreu-se ao repositório *WebRTC-Experiment*, nomeadamente à biblioteca *RTC-MultiConnection*. Optou-se por usar uma solução já implementada, adaptando-se ao servidor que está a correr localmente. Desta forma, é necessária uma solução *Javascript*, que corre no servidor, para que todos os clientes se possam ligar a esse processo e possam fazer a negociação de sessões.

4.4 Mensagens

Para construir uma *framework* e aplicação funcionais é necessário que haja interação entre os clientes e os componentes dos servidores, de forma a que se consiga aceder aos serviços. Esta secção demonstra de que forma os clientes interagem com os módulos, para conseguirem executar determinadas ações.

4.4.1 Registo de cliente

Na aplicação é possível realizar o registo de utilizadores no sistema de base de dados, onde toda a informação é tratada. Um cliente, para realizar o registo necessita preencher um formulário simples com os campos: *Username*, *Password*, *SIP Address*, *SIP Server*, Serviços, *Email*, como mostra a Figura 4.5.

Estes campos são obrigatórios, isto é, necessitam de ser preenchidos para o cliente se poder registar no sistema. No caso do campo "Serviços", existe uma lista dos serviços que o sistema

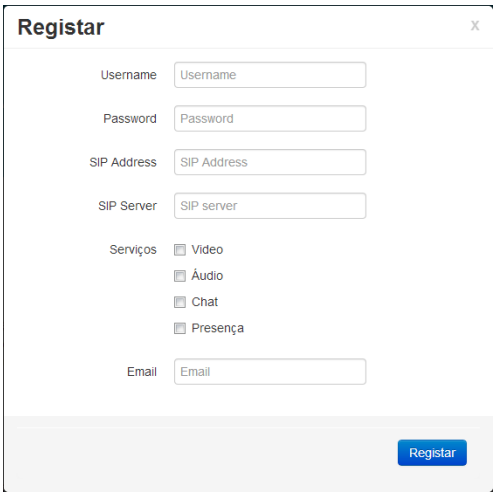


Figura 4.5: Formulário de registo no sistema.

suporta, ou seja, o cliente pode escolher entre estes quais é que deseja subscrever. De momento encontram-se disponíveis serviços como: Vídeo, Áudio, *Chat* e Presença.

Para um utilizador se registar, é indispensável o envio de uma mensagem do tipo *"registuser"* para "Módulo Registo de Utilizadores", como mostra a Figura 4.6, que se encontra registado no *eventbus*.

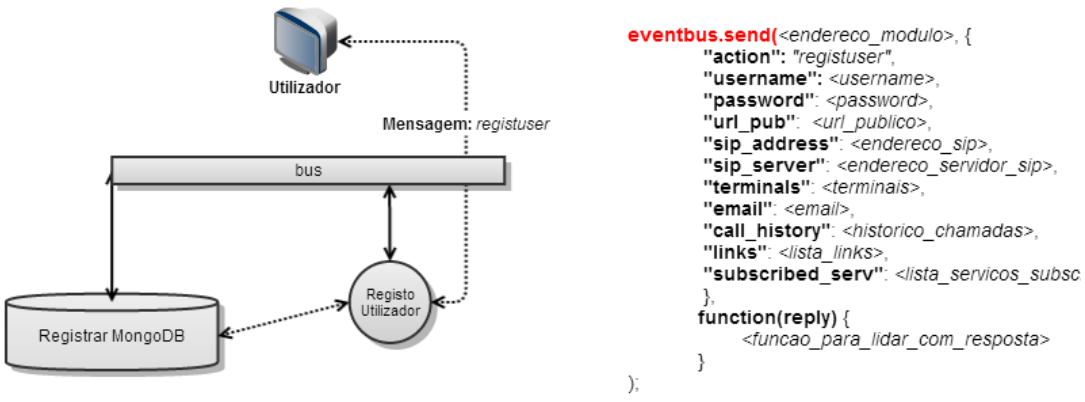
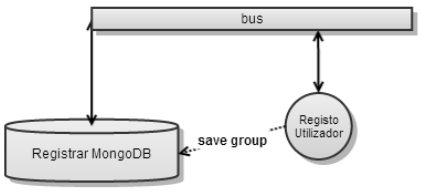


Figura 4.6: Mensagem de registo no sistema.

Depois do módulo "Registo de Utilizadores"receber a mensagem por parte da aplicação cliente, irá encaminhá-la para "Módulo de Base de dados"onde é guardado um novo utilizador. Ao receber a resposta de que o utilizador foi registado com sucesso, irá enviar outra mensagem para o "Módulo Base de dados"para que possa criar um grupo de perfil de utilizador, Figura 4.7.




```
eventbus.send(<endereco_modulo>, {
      "action": "save",
      "collection": <collection>,
      "name": <nome>,
      "groups_presence": {
            "authorised_presence": <password>,
            "presence_authorization_presence": <url_publico>,
            "denied_presence": <endereco_sip>,
      }
      "type": private,
      "group_elements": <collection>,
      "sessionId": <collection>,
      "photo": <collection>,
      "url_pub": <collection>,
      "admin": <collection>,
      "session_communication_group": <endereco_servidor_sip>,
      "terminals": <terminais>,
      "email": <email>,
      "call_history": <historico_chamadas>,
      "links": <lista_links>,
      "subscribed_serv": <lista_servicos_subscritos>
      },
      function(reply) {
          <funcao_para_lidar_com_resposta>
      }
);
```

Figura 4.7: Mensagem para gravar grupo de utilizador.

É neste grupo que se encontra todo o tipo de informação referente ao utilizador, desde os terminais que usa, o histórico de chamadas, os serviços subscritos, etc. Existe ainda um campo *"groups_presence"* onde se encontram grupos de subscrição:

- *authorised_presence*: grupo onde se encontram todos os utilizadores da lista de contactos que foram aceites;
- *presence_authorization_pendent*: grupo de utilizadores que pediram subscrição e estão em estado pendente;
- *denied_presence*: grupo de utilizadores que pediram subscrição e foram rejeitados.

Para além da execução destas operações é realizado ainda outro tipo de operação, que passa pela criação de objetos na *collection "UserServices"*. A Figura 4.8 mostra o tipo de mensagem enviado para "Módulo de Base de Dados".

Essa mensagem é enviada por cada serviço que o cliente tenha subscrito, ou seja, no registo o utilizador pode escolher os serviços que quer subscrever e ao executar essa operação são criados documentos na *collection "UserService"*, onde fica guardada a configuração associada àquele serviço a um determinado utilizador.

Para isso existem campos como *"service"* e *"username"* que identificam o utilizador e que serviço lhe foi subscrito. No campo de configuração (*configuration*) está especificado o campo *"address"* que identifica o endereço do serviço daquele utilizador e o campo *"automatic _authorisation _accpetance"*, que existe apenas no serviço de presença. Este campo pode tomar valores

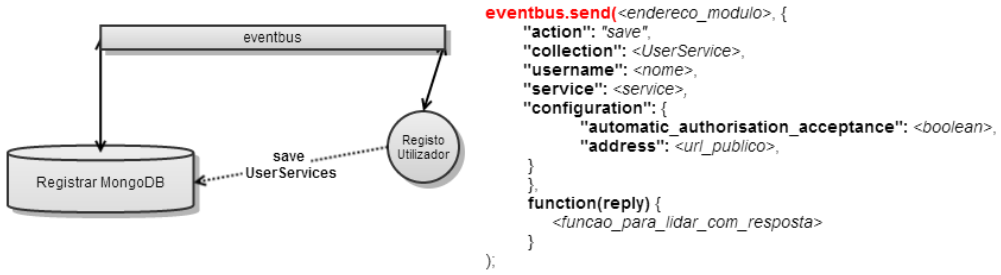


Figura 4.8: Gravar serviços subscritos por um utilizador.

booleanos, desta forma se estiver a *true* todos os pedidos de subscrição para aquele utilizador serão aceites automaticamente, caso contrário o utilizador receberá uma notificação referente ao pedido de subscrição feito, onde o mesmo poderá ser aceite ou rejeitado.

4.4.2 Autenticação de Cliente

Para poder ter acesso aos serviços subscritos no registo, é necessário aceder ao sistema internamente é, neste caso, que entra a autenticação de utilizadores. Para um utilizador poder executar essa operação, necessita de comunicar com o "Módulo de Gestão de Autenticações e Autorizações", consoante ilustrado na Figura 4.9.

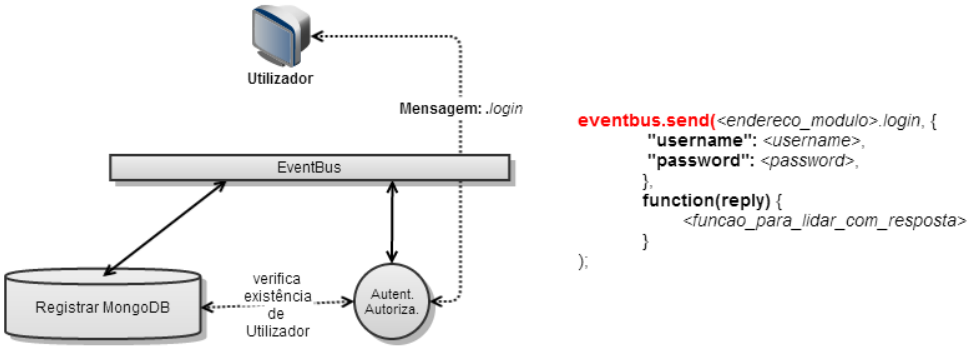


Figura 4.9: Autenticação de um utilizador no sistema.

É enviada uma mensagem do tipo *login*, onde deve ser identificado o *username* e a *password* e quando o cliente enviar a mesma, irá receber uma resposta. No caso dessa resposta ser de sucesso irá enviar outra mensagem, mas desta vez para o "Módulo de Gestão de sessões".

Essa mensagem serve para iniciar uma sessão para aquele utilizador, ou seja, sempre que um

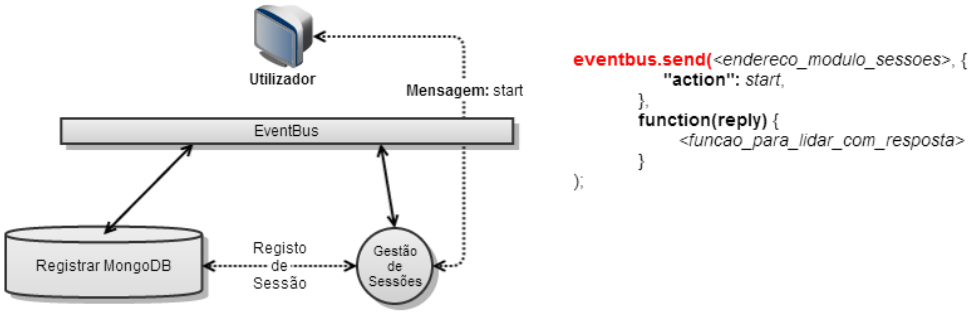


Figura 4.10: Iniciar sessão para um utilizador.

utilizador fizer autenticação no sistema vai enviar uma mensagem do tipo *start* para o "Módulo de Gestão de Sessões"para iniciar a sessão, como indicado na Figura 4.10. Depois de enviar essa mensagem irá receber outra resposta, indicando que o registo da sessão, foi executada com sucesso e, por conseguinte, ao receber essa mensagem terá de guardar informação referente à sessão.

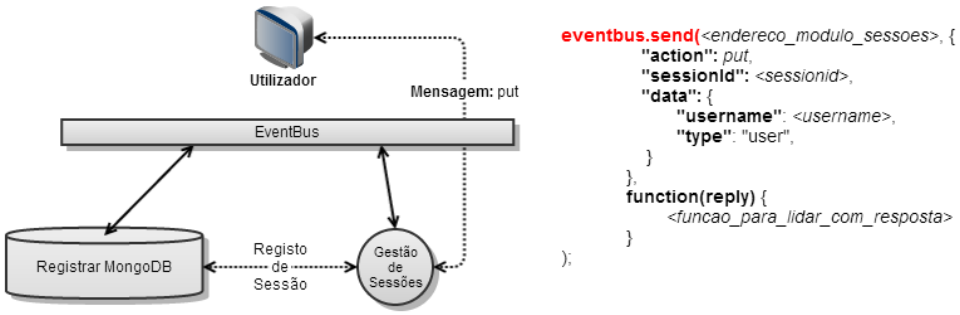


Figura 4.11: Guardar informação do utilizador na sessão associada.

Para guardar essa informação terá de enviar, mais uma vez, uma mensagem do tipo *put* para o "Módulo de Gestão de Sessões", como mostra a Figura 4.11. A informação que se pretende guardar neste momento é apenas o *username* e o tipo de sessão (*type*), que neste caso é do tipo *user*.

No decorrer da autenticação o utilizador será reencaminhado para a página *home* e, quando esta for carregada, serão executadas algumas funções, que permitem recolher mais informação sobre o utilizador que fez a autenticação, de forma a que o sistema o possa servir consoante os serviços que tem ou não subscritos. Como já foi referido anteriormente, o utilizador precisa

de saber quais os serviços que tem subscritos e quais são os que o seu terminal suporta. Para saber quais são os serviços que o terminal do utilizador suporta é usada a *API Media Streams* da tecnologia *WebRTC*. Sendo a função *getUserMedia()* executada com sucesso, é retornado um objeto que contém informação sobre as *streams* que o terminal suporta. Nesse objeto é possível identificar se o terminal suporta vídeo, áudio ou ambos, assumindo-se à partida que o resto dos serviços são suportáveis. De seguida, é criado um *array* que contém a indicação do que o terminal suporta e é enviado para o "Módulo de Pesquisa e Intersecção de Serviços", como ilustrado na Figura 4.12.

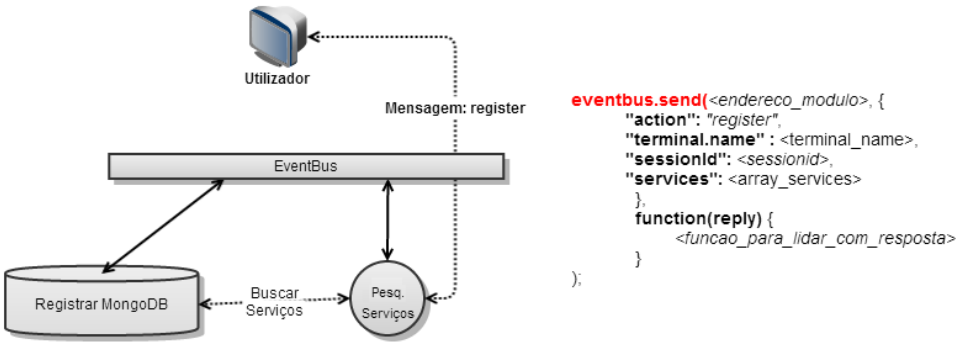


Figura 4.12: Mensagem de verificação de serviços de um utilizador.

Depois disso vai verificar quais são os serviços que estão associados a esse utilizador e é executada a intersecção entre os mesmos que foram recebidos por *array* e os que estão subscritos. Este módulo encarrega-se de criar um *array* com o resultado dos serviços e as suas configurações, como mostra o exemplo da Figura 4.13.

```
[
 {
    "service": <serviço>,
    "uri" : <biblioteca_para_carregar_no_cliente>,
    "initServ": <booleano>
    "name": <nome_do_servico>
    "configuration": {
                "address": <endereço_serviço>,
                ...
                }
 },
...
]
```

Figura 4.13: *Array* de serviços em resposta do servidor.

Nesta resposta vem a identificação dos serviços e a configuração de cada um, como também qual o *uri* do serviço. O cliente só vai carregar as bibliotecas correspondentes ao serviço, caso

tenha suporte a esse serviço, desta forma consegue-se um sistema mais flexível. No caso do campo *initServ*, este serve para indicar se o serviço deve ser ou não carregado ao início, ou se deve ser guardado em memória para ser apenas carregado quando for usado.

4.4.3 Módulo de presença e Interação no sistema

Depois de todo o processo de autenticação e o sistema verificar quais os serviços que o cliente suporta, para cada um deles irá será carregada a biblioteca correspondente. Neste caso será carregada a biblioteca *"presence.js"*, onde se encontram todas as funções e operações que devem ser executadas pelo cliente relativamente à presença. Este serviço está encarregue das atualizações de estado do cliente, verificar quem se encontra na lista de amigos e qual o seu estado atual, adicionar uma pessoa do sistema à lista de contactos e ainda estar atento a eventos que lhe digam respeito. É um serviço que se encontra disponível e funcional na *framework*. Já na integração com a aplicação é criado um objeto *JavaScript* desse serviço *"new PresenceService()"*. Para este serviço ficar completamente funcional é necessário carregar o módulo de presença para o servidor, para que possa existir interação entre servidor e cliente. Ao ser criado este objeto é possível ter acesso a várias operações, sendo elas:

- *getPresences*;
- *updatePresence*;
- *notifyFriendsOnline*;
- *registHandler*;
- *addAsFriend*.

A primeira função serve para o cliente recolher os estados de presença que foram definidos no sistema, que neste caso são:

- *Available*;
- *Busy*;
- *Away*;
- *Not Here*;

- *Offline/Invisible*.

Para esta informação ser recolhida é enviada uma mensagem do tipo *"get"* para o módulo de presença sobre o *eventbus*, como mostra a Figura 4.14.

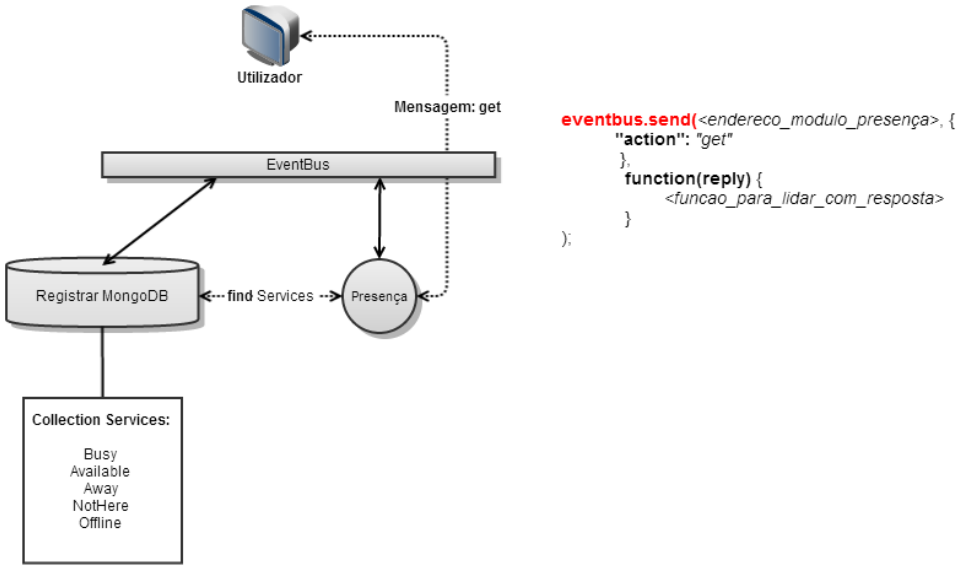


Figura 4.14: Mensagem para obter os estados de presença do sistema.

Na aplicação, o resultado a este pedido é um *array* com o nome dos serviços e o seu tipo, que posteriormente é carregado para um objeto *dropdown* de *HTML5*. Sempre que o utilizador quiser alterar o seu estado basta escolher o estado que quiser nesse *dropdown*.

Quando é feita a autenticação de um utilizador para além de recolher informação sobre os estados existentes no sistema, é necessário que o seu estado de presença mude para *Available*, isto porque anteriormente se encontrava *Offline*. Para isso é usada a função *"updatePresence"*, que envia para o módulo de presença no servidor uma mensagem do tipo *"update"*, como mostra a Figura 4.15.

O módulo de presença, ao receber a mensagem irá verificar o valor do campo *login*, pois indica se o utilizador está ou não a executar autenticação. Caso este campo se encontre com o valor *true*, o próprio servidor irá registar um *handler* com o endereço de presença desse utilizador (*presence.<username>*), de forma a estar sempre atento à mudança de estado do mesmo, e irá atualizar o estado de presença na sessão respetiva ao utilizador. Para além disso, irá verificar quem são os utilizadores que se encontram na lista de contactos desse utilizador, para que, desta forma, possam ser carregados no lado do cliente e possa ser registado um *handler* referente a

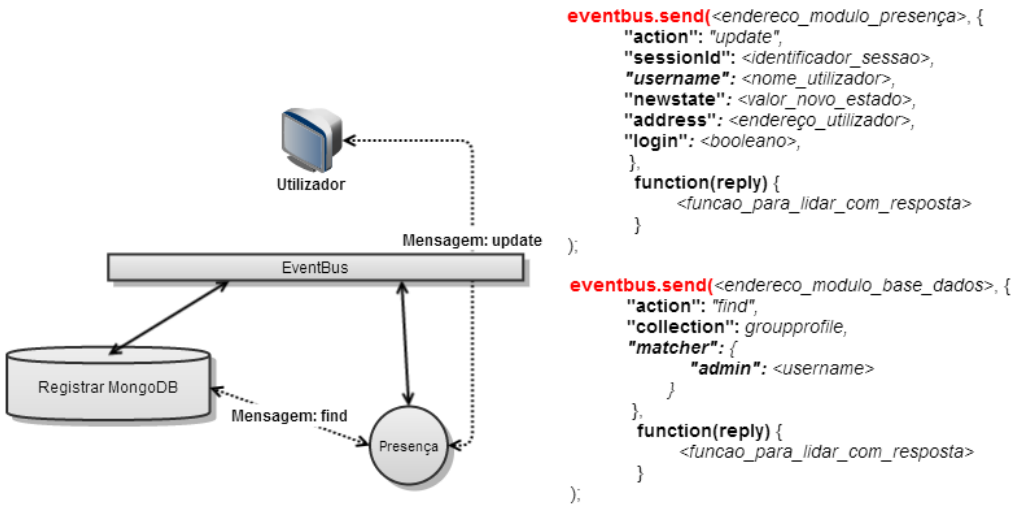


Figura 4.15: Mensagem para notificar e obter lista de contactos.

cada um dos utilizadores. A seguir, quando o utilizador atualizar o seu estado, irá ser enviada uma mensagem em *publish*, para o seu endereço de presença, como mostra a Figura 4.16.

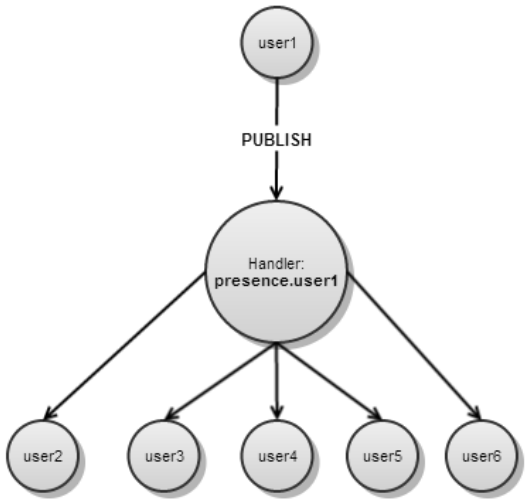


Figura 4.16: Notificação de mudança de estado.

Neste caso o campo *login* irá com o valor *false*, pois o utilizador já não se encontra a fazer autenticação.

Como foi dito anteriormente, existem três grupos de presença associados ao grupo de cada utilizador registado no sistema, sendo estes: *authorised_presence*, *presence_authorization_pendent*

e *denied_presence*. Estes grupos dizem respeito à lista de contactos autorizados, aos contactos que fizeram pedido mas se encontram pendentes e aos que foram rejeitados, respetivamente. Neste sistema inicial todos os pedidos executados serão automaticamente aceites, ou seja, sempre que seja executado um pedido, os contactos serão logo adicionados à lista *authorised_presence*. Para adicionar uma pessoa do sistema à lista de contactos é necessário enviar uma mensagem do tipo *subscribe* para o módulo de presença, como está representado Figura 4.17.

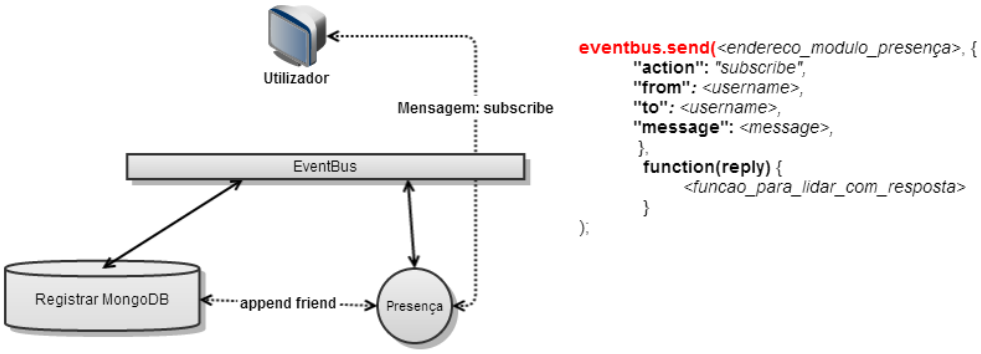


Figura 4.17: Mensagem para adicionar contacto.

O módulo de presença, ao receber a mensagem *subscribe*, irá verificar de quem é a mensagem a partir do campo *from* e quem é que essa pessoa pretende adicionar a partir do campo *to*. Este irá ainda verificar se o campo *"automatic_authorisation_acceptance"* se encontra a *true* ou *false*. É necessário garantir que não é adicionada a mesma pessoa duas vezes ao grupo *"authorised_presence"*.

Por fim, este módulo ficará encarregue de notificar os clientes de que foi adicionado uma nova pessoa à lista de contactos, para que possa haver uma atualização na parte do cliente.

4.4.4 Módulo de gestão de utilizadores

Este módulo permite a pesquisa de pessoas no sistema, desta forma é possível encontrar pessoas para adicionar à lista de contactos ou até mesmo acrescentar pessoas a uma conferência que esteja a decorrer, consoante o ilustrado na Figura 4.18.

No cliente existe um campo de pesquisa que permite, sempre que haja, a escrita de uma letra, que será enviada por mensagem para o "Módulo de Gestão de Utilizadores". Existe ainda a possibilidade de definir campos para a filtragem da pesquisa, este ficam definidos no *"querycri-*

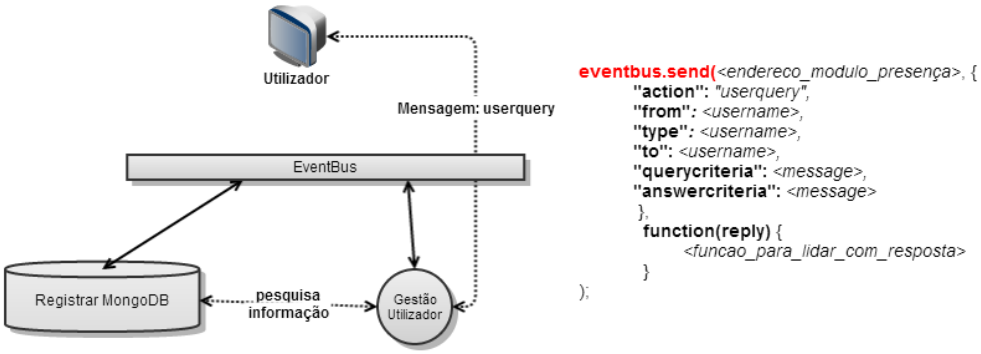


Figura 4.18: Mensagem para pesquisa de utilizadores do sistema.

teria". Da mesma maneira é possível definir qual a informação que se quer que seja devolvida por parte do servidor, em relação aos utilizadores que se querem procurar, esta informação fica definida no campo *"answercriteria"*.

Para ser feita uma pesquisa mais geral, existe a possibilidade de usar expressões regulares numa base de dados *Mongo*. Nesta caso é usada a expressão */"string"/i*, que permite que sejam devolvidos todos os resultados que tenham aquela *"string"*, não se importando se é *case sensitive*. Nesta especificação inicial o campo *"querycriteria"* contém apenas o *username*, isto implica que a pesquisa apenas é feita com esse campo. No caso do *"answercriteria"*, apenas está definido que deve ser devolvido o *"username"* na resposta.

4.5 Conferências áudio e vídeo

Anteriormente, foi referido que o repositório *WebRTC-Experiment* oferecia uma maior diversidade de serviços relacionados com *WebRTC*.

Conforme as necessidades da prova de conceito, foram escolhidas duas bibliotecas desse repositório: *RTCMultiConnection.js* e *RecordRTC.js*. *RTCMultiConnection* para sinalização usa, por defeito, a tecnologia *firebase*, mas permite o uso de outras tecnologias. Para iniciar uma conferência é necessário que o cliente na sua lista de contactos selecione pelo menos uma pessoa, dessa forma irá aparecer uma janela para criar uma conferência, como mostra a Figura 4.19.

Neste caso selecionou-se o utilizador "teste". Para iniciar uma sessão deve-se, não é obrigatório, modificar o assunto da chamada no campo “Subject Call...” e convidar as pessoas selecionadas. Será enviado uma mensagem *"invite"* para cada um dos contactos selecionados, como

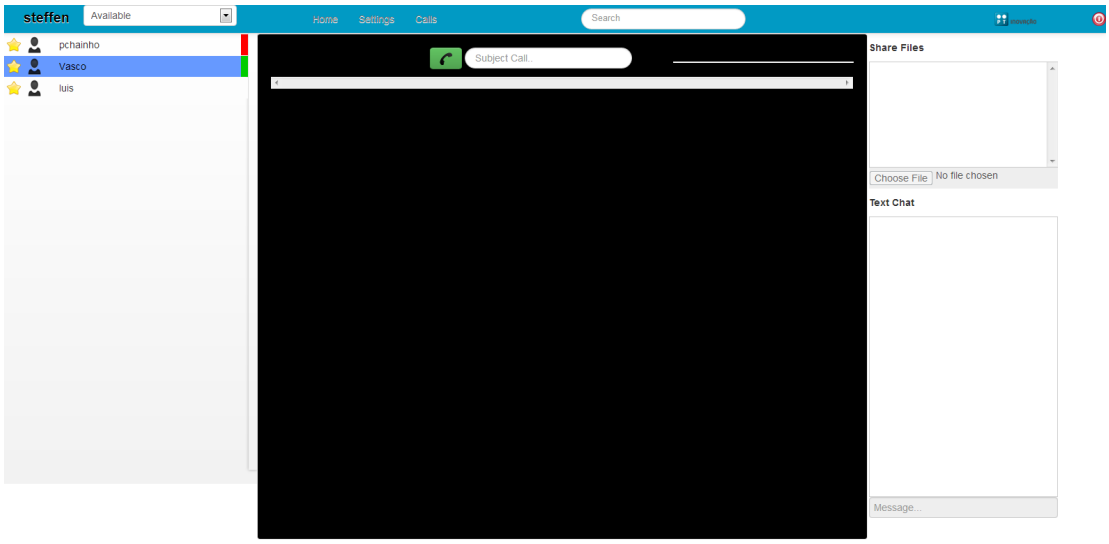


Figura 4.19: Convidar pessoas para uma sessão.

mostra a Figura 4.20.

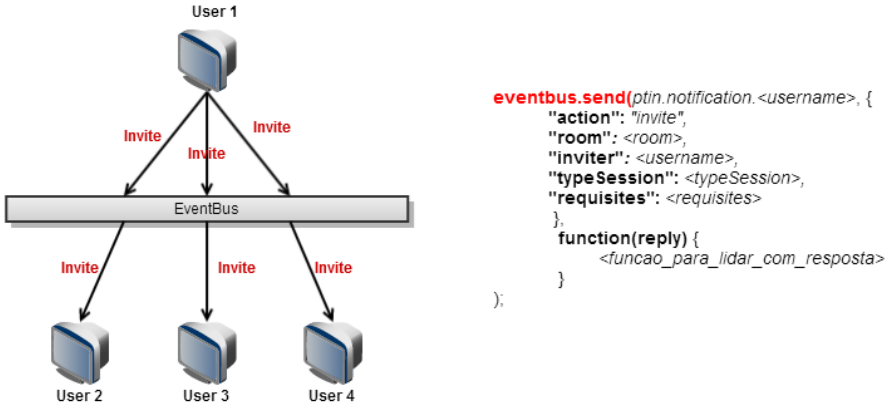


Figura 4.20: Mensagem de notificação para início de sessão.

Os utilizadores que receberem esse convite serão notificados com um *popup*, a identificar quem é que os está a tentar contactar e o assunto para a conferência, com possibilidade de aceitar ou rejeitar o mesmo, como mostra a Figura 4.21. Existe a possibilidade de quem envia este convite poder cancelar o pedido e, além disso, é definido um *timeout* específico para cada utilizador e, sempre que esse for excedido, o convite é cancelado.

Este *popup* aparece devido aos eventos que são tratados no "Módulo de notificações". Estas notificações podem dizer respeito a: convite para iniciar sessão; convite para adicionar à lista de

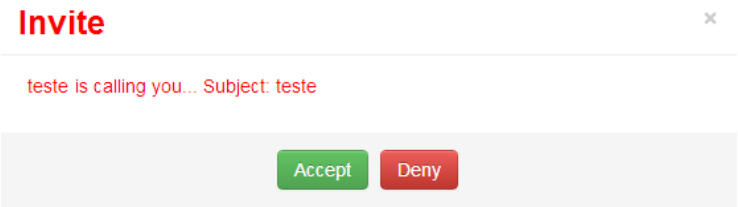


Figura 4.21: Notificação para início de sessão.

contactos; etc. Para um utilizador receber esse tipo de notificações é registado um *handler* no *eventbus* para cada cliente que se encontre com uma sessão de utilizador iniciada. Para a sessão ser criada é necessário que pelo menos uma pessoa aceite o convite inicial, caso contrário não existe troca de informação *SDP* entre utilizadores. À medida que os utilizadores forem aceitando juntar-se à conferência é feita uma negociação de *codecs*, faixas de áudio e vídeo e também de dados. Pode-se verificar no Anexo A as mensagens enviadas e recebidas durante a sinalização entre clientes, tanto a nível de *codecs* e da informação que se quer trocar durante a conferência, como também a negociação de candidatos *ICE*. No Anexo B é mostrado o fluxo de mensagens trocado entre os vários *peers* durante a negociação da sessão, para iniciar uma chamada. Se a negociação entre todos os *browsers* dos clientes for executada com sucesso e sem problemas irá ser iniciada uma sessão entre utilizadores, como mostra a Figura 4.22.

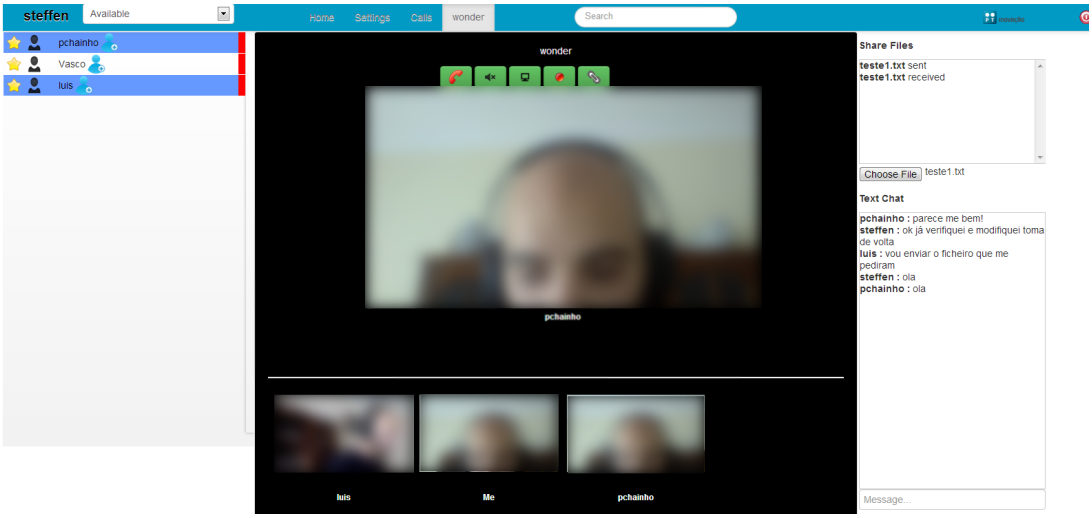


Figura 4.22: Conferência entre três pessoas.

É possível adicionar pessoas a uma conferência que esteja a decorrer, ou seja, numa situação em que uma pessoa num momento inicial não se encontre disponível para iniciar uma conferência, poderá mais tarde juntar-se a essa conferência. Esse convite pode ser executado por qualquer

pessoa que se encontre na conferência.

Pode ser adicionada também qualquer pessoa do sistema a uma conferência, mesmo que não pertença à lista de contactos, bastando fazer uma pesquisa rápida no sistema e convidá-la para se juntar. No caso de se querer convidar uma pessoa externa ao sistema, ou seja, que não se encontre registada, no momento em que a conferência é iniciada é criado ao mesmo tempo um *URL* especifico (`https://<dominio>:<port>/session.html?<nome_sessÃčo>#<session_id>`), que deve ser enviado por *e-mail* ou por outro tipo de método para a pessoa em questão.

Durante uma sessão de conferência nesta aplicação, encontram-se ativos os seguintes serviços:

- *chat*;
- partilha de ficheiros;
- conferência áudio e vídeo;
- partilha de ecrã;
- *mute*;
- gravação.

Estes serviços foram implementados com a ajuda das bibliotecas do repositório *WebRTC-Experiment*. Para a sinalização e a negociação entre vários *peers*, usou-se tanto no servidor como no cliente a tecnologia *socket.io*, que permite a criação de aplicações que funcionem em tempo real e além disso é uma das soluções implementadas com as bibliotecas escolhidas. Para cada utilizador é criada uma ligação sempre que é necessário iniciar uma sessão de conferência.

Para garantir que as sessões criadas têm um identificador único é usado o algoritmo *Universally Unique Identifier* versão 4. Este algoritmo garante que a probabilidade de criar identificadores iguais é muito pequena. Desta forma é possível que duas ou mais sessões tenham o mesmo assunto, mas os seus identificadores vão ser diferentes [52].

No fim de cada sessão de conferência deve ser gravado no sistema as características que dizem respeito a cada sessão. Numa fase inicial são gravados apenas os seguintes dados:

- utilizador que inicia a conferência;

- duração da conferência;
- participantes dessa conferência;
- hora inicial da conferência.

No histórico de chamadas podem existir três tipos para as chamadas serem gravadas: *rejected*, *accepted* ou *notattended*. As chamadas são guardadas como *accepted*, sempre que é iniciada uma chamada entre utilizadores. As conferências rejeitadas são guardadas como *rejected*, caso todos os utilizadores que foram convidados rejeitem o convite recebido, sendo também gravado na base de dados este tipo de chamada na pessoa que enviou o convite. No caso das *notattended*, são gravadas sempre que o tempo de *timeout* é ultrapassado. Para guardar esta informação é necessário enviar uma mensagem para o "Módulo de histórico", assim esta informação é armazenada no perfil de utilizador. Esta informação é enviada numa mensagem do tipo *"registcall"*, como mostra a Figura 4.23.

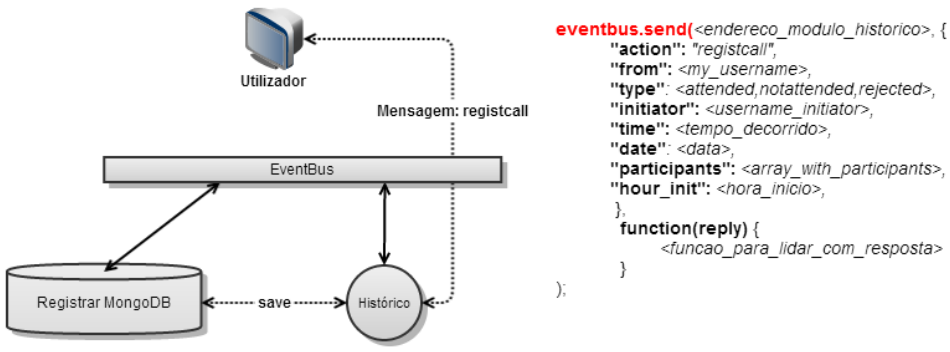


Figura 4.23: Registo de chamada no histórico de chamadas.

Desta forma é possível ter um histórico sobre todas as chamadas que foram executadas ou foram recebidas, podendo-se consultar posteriormente algumas características de cada uma das conferências.

4.6 Sumário

Neste capítulo foi descrita a parte funcional da *framework* e da prova de conceito. Abordaram-se as decisões arquiteturais tomadas inicialmente e, de seguida, passou-se para a descrição de cada módulo envolvido no desenvolvimento para esta solução, onde todas configurações que cada um

pode ter foram caracterizadas. Em relação ao funcionamento da *framework* e da aplicação foram descritas todas as mensagens trocadas entre os diferentes clientes, clientes e servidor e também entre os módulos que se encontram disponíveis no servidor.

Capítulo 5

Cenários de teste e resultados obtidos

Depois da implementação da *framework* e do cliente *WebRTC*, é indispensável existir uma fase de validação e análise de desempenho. Este capítulo descreve o trabalho realizado nesse sentido.

5.1 Testes de validação

Todo o processo de validação e teste de serviços foi contínuo, acompanhando a implementação dos módulos de suporte. Desta forma, foi possível verificar e validar as funcionalidades, de forma a poder avançar para o desenvolvimento de outros módulos. Foi feita uma validação final, onde foram testadas todas as funcionalidades implementadas na aplicação cliente.

Num primeiro passo, os utilizadores registam-se no sistema e posteriormente fazem a autenticação no sistema. No passo seguinte, foi testada a funcionalidade de presença, onde se faz a mudança de estados e se verifica se toda a lista de contactos de um cliente recebe a notificação de mudança de estado. Neste caso concluiu-se que se encontrava funcional. Ainda no mesmo serviço, verificou-se a funcionalidade de adicionar outros clientes à lista de contactos. Os clientes eram sempre adicionados à lista com sucesso. No entanto, por vezes quando se queria remover o contacto, nem sempre essa função era executada com sucesso. Isto devia-se ao facto de a implementação do evento *"onRemoveContact"*, não se encontrar bem implementado, pois nem sempre é despoletado, quando recebe uma mensagem de remoção do servidor.

Dentro do serviço de chamadas / conferência verificou-se que as notificações de convite e de aceitação eram realizadas com sucesso e, a partir do momento que um cliente aceitava o convite de outro, a chamada / conferência era inicializada. Verificou-se algum atraso na troca de

informação *SDP*, e na inicialização de chamadas. Depois disto, e dentro da mesma chamada / conferência, foi enviado um convite para outro cliente subscrito no sistema e outro a um cliente que não estava subscrito no sistema, para serem adicionados a essa chamada. Foi tudo realizado com sucesso, verificando-se que a troca de ficheiros e de mensagens *chat* eram enviadas com sucesso, como também o áudio e vídeo. Por vezes havia ligeiros atrasos no áudio e quebras de vídeo.

Por fim, analisaram-se as funcionalidades do histórico de chamadas, em que se verificou que toda a informação relativa a uma chamada / conferência era guardada e apresentada com sucesso, tanto para as atendidas, não atendidas e rejeitadas.

No final deste desenvolvimento foram feitas duas apresentações dentro do ambiente empresarial, em que se mostrou todo o processo realizado no desenvolvimento da *framework* e uma *demo* da aplicação com a *framework* a funcionar.

5.2 Testes de desempenho computacional

Nestes testes realizados foram analisados, essencialmente, o desempenho em termos de utilização de *Central Processing Unit (CPU)* e memória usada em cada sessão. Para estes testes foram definidos dois cenários diferentes, focando um ambiente empresarial e um ambiente doméstico. A Tabela 5.1 representa de uma forma resumida os cenários, os testes de cada um e as características relacionadas com os mesmos.

Tabela 5.1: Percentagem dos *web browsers* usados [11].

Cenário 1	**Tipo de Rede**	**Nº de Utilizadores**
Teste1	Empresarial	2
Teste2	Empresarial	4
Cenário 2	**Tipo de Rede**	**Nº de Utilizadores**
Teste1	Doméstica	2
Teste2	Doméstica	4

O primeiro cenário diz respeito a uma rede empresarial, em que os clientes se ligam por *Virtual Private Network (VPN)* à mesma. Para verificar a conectividade dos clientes a este tipo de rede foi executado o comando *ping -n 30 <ip_servidor >*, de forma a verificar o *Round Trip Time (RTP)* e se existiam perdas de pacotes. A Figura 5.1, ilustra os resultados retirados depois

de executar o comando.

```
C:\Users\10056914>ping -n 30 10.112.67.93

Pinging 10.112.67.93 with 32 bytes of data:
Reply from 10.112.67.93: bytes=32 time=391ms TTL=62
Reply from 10.112.67.93: bytes=32 time=368ms TTL=62
Reply from 10.112.67.93: bytes=32 time=409ms TTL=62
Reply from 10.112.67.93: bytes=32 time=412ms TTL=62
Reply from 10.112.67.93: bytes=32 time=374ms TTL=62
Reply from 10.112.67.93: bytes=32 time=416ms TTL=62
Reply from 10.112.67.93: bytes=32 time=334ms TTL=62
Reply from 10.112.67.93: bytes=32 time=441ms TTL=62
Reply from 10.112.67.93: bytes=32 time=448ms TTL=62
Reply from 10.112.67.93: bytes=32 time=433ms TTL=62
Reply from 10.112.67.93: bytes=32 time=490ms TTL=62
Reply from 10.112.67.93: bytes=32 time=441ms TTL=62
Reply from 10.112.67.93: bytes=32 time=449ms TTL=62
Reply from 10.112.67.93: bytes=32 time=458ms TTL=62
Reply from 10.112.67.93: bytes=32 time=508ms TTL=62
Reply from 10.112.67.93: bytes=32 time=471ms TTL=62
Reply from 10.112.67.93: bytes=32 time=543ms TTL=62
Reply from 10.112.67.93: bytes=32 time=560ms TTL=62
Reply from 10.112.67.93: bytes=32 time=549ms TTL=62
Reply from 10.112.67.93: bytes=32 time=477ms TTL=62
Reply from 10.112.67.93: bytes=32 time=582ms TTL=62
Reply from 10.112.67.93: bytes=32 time=581ms TTL=62
Reply from 10.112.67.93: bytes=32 time=598ms TTL=62
Reply from 10.112.67.93: bytes=32 time=605ms TTL=62
Reply from 10.112.67.93: bytes=32 time=623ms TTL=62
Reply from 10.112.67.93: bytes=32 time=616ms TTL=62
Reply from 10.112.67.93: bytes=32 time=629ms TTL=62
Reply from 10.112.67.93: bytes=32 time=568ms TTL=62
Reply from 10.112.67.93: bytes=32 time=526ms TTL=62
Reply from 10.112.67.93: bytes=32 time=453ms TTL=62

Ping statistics for 10.112.67.93:
    Packets: Sent = 30, Received = 30, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 334ms, Maximum = 629ms, Average = 491ms
```

Figura 5.1: *Ping* ao servidor com VPN.

Para a rede empresarial verificou-se um *RTT* mínimo de 334 milissegundos e um máximo de 629 milissegundos, perfazendo uma média de todos os pacotes de 491 milissegundos. Quanto a perda de pacotes não se verificou qualquer perda.

Num segundo cenário os clientes ligam-se a uma rede doméstica, em que o servidor também é um computador normal. Executou-se o mesmo processo que na rede empresarial, onde se podem verificar os resultados na Figura 5.2.

Quanto a este cenário verificou-se que o mínimo de *RTT* é de aproximadamente 0 milissegundos, enquanto o máximo atinge os 1 milissegundos, fazendo assim uma média de aproximadamente 0 milissegundos. Neste caso também não se verificaram perda de pacotes.

Com os resultados retirados pode-se verificar que a rede doméstica tem um *RTT* bastante mais pequeno que o da rede empresarial. Para aplicações que fornecem serviços em tempo real, não devem existir atrasos elevados, pois consequentemente irão causar uma degradação da qualidade de serviço.

```
C:\Users\10056914>ping -n 30 192.168.1.6

Pinging 192.168.1.6 with 32 bytes of data:
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time=1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time=1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time=1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time=1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time=1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64
Reply from 192.168.1.6: bytes=32   time<1ms TTL=64

Ping statistics for 192.168.1.6:
    Packets: Sent = 30, Received = 30, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

Figura 5.2: *Ping* ao servidor na rede doméstica.

5.2.1 Cenário 1 - Rede empresarial

Neste primeiro cenário, os clientes ligam-se por uma *Virtual Private Network (VPN)* à rede empresarial, de forma a aceder ao servidor que se encontra nessa mesma rede. Num primeiro teste apenas dois clientes realizam uma conferência áudio, vídeo e de dados entre eles, como mostra a Figura 5.3.

Os clientes têm acesso aos serviços fornecidos pelo servidor e conseguem realizar uma chamada entre eles. As especificações de cada máquina podem ser encontradas no Anexo C.

Neste exemplo, o PC1 envia um convite de chamada para o PC2. Na Figura 5.4 ilustram-se os resultados obtidos relativos à utilização de memória no PC2, figura 5.4.

O ponto 1 identifica o momento em que o PC2 recebe o convite e o aceita. A partir desse momento temos um aumento gradual de utilização de memória. Esse aumento é de quase 2 *MegaBytes (MB)*, estabilizando depois de toda a negociação ser feita entre os *peers*. No momento em que a chamada é desligada nota-se um decréscimo repentino na utilização de memória,

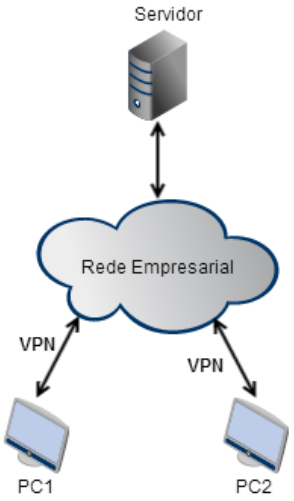


Figura 5.3: Cenário 1 e Teste 1.

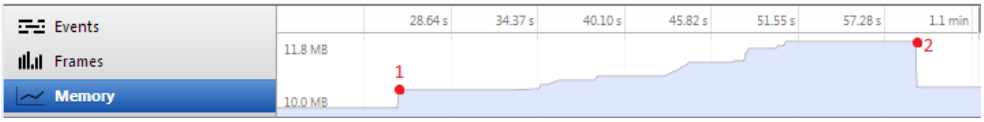


Figura 5.4: Gráfico de utilização de memória numa chamada.

fazendo com que desça mais de 1 *MB*. Não foi possível identificar o motivo pelo qual a memória não voltou a estabilizar, como no início.

Em termos de vídeo e áudio não se notaram grandes atrasos nem quebras de sequência. Em relação a dados não houve perda de pacotes, permitindo que todas as mensagens de *chat* fossem trocadas, todos os ficheiros fossem enviados e recebidos.

Neste mesmo cenário foi executado outro teste (Teste 2), neste caso em vez de serem apenas dois utilizadores, fez-se uma conferência com as mesmas características mas entre 4 pessoas, consoante ilustrado na Figura 5.5.

Foram retirados resultados tanto ao nível de CPU e de memória, mas apenas num dos computadores, visto que as características de cada um são bastante semelhantes.

Os resultados do *CPU* dizem respeito ao início e ao fim da conferência, podendo-se verificar o resultado nas Figuras 5.6 e 5.7.

Verificou-se um aumento significativo da utilização do *CPU* a partir do momento que se iniciou a conferência (Figura 5.6) chegando a atingir 50% de utilização do *CPU*. Quando a chamada se desligou a sua utilização voltou a estabilizar num uso normal do computador, descendo con-

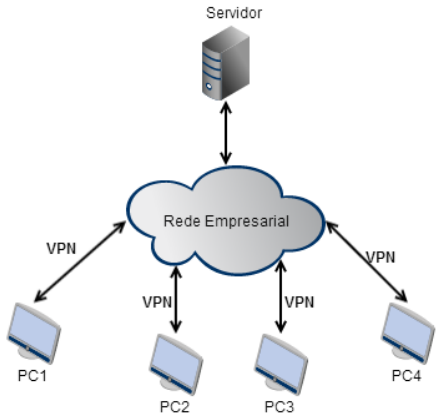


Figura 5.5: Cenário 1 e Teste 2.

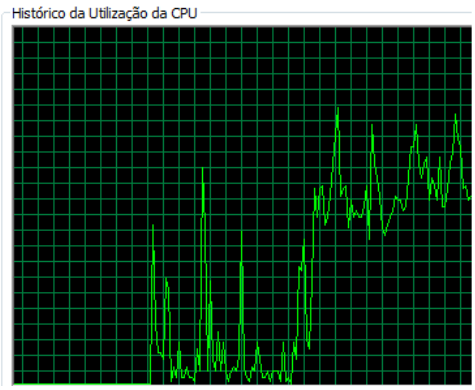


Figura 5.6: Utilização de *CPU* no inicio da conferência.

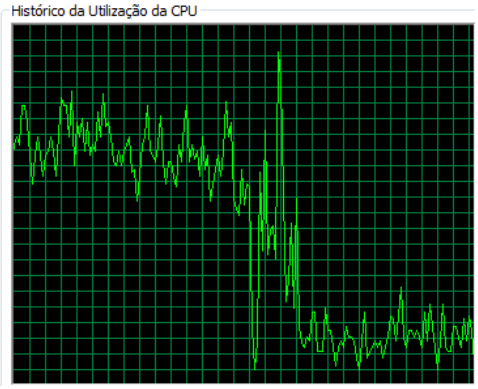


Figura 5.7: Utilização de *CPU* ao finalizar conferência.

Figura 5.8: Comportamento da utilização de *CPU*.

sideravelmente - cerca de 40%.

Na utilização de memória do computador os resultados foram os representados na Figura 5.9.

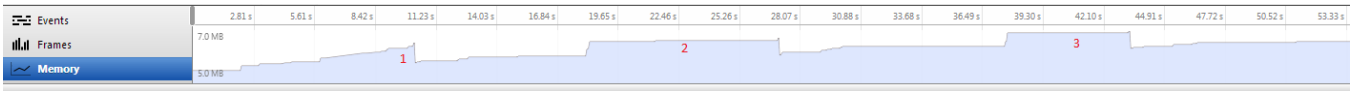


Figura 5.9: Utilização de memória ao iniciar a conferência.

Neste exemplo verifica-se um aumento de utilização de memória sempre que alguém entra na conferência como se pode verificar no ponto 1, 2 e 3, da Figura 5.9. Neste exemplo a utilização de memória estabilizou nos 9.2*MB*, ou seja, houve um aumento de 4.2*MB* durante a conferência

entre 4 pessoas. Quando a conferência é desligada entre todas as pessoas, existe um decréscimo significativo e linear, sendo isso visível na Figura 5.10.

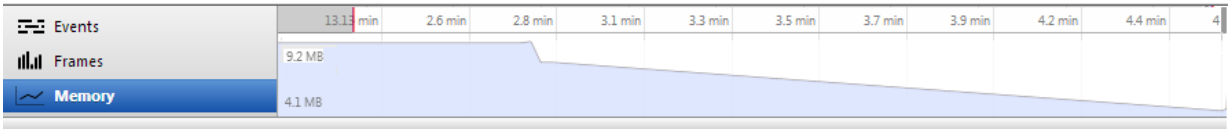


Figura 5.10: Utilização de memória ao finalizar conferência.

Em termos de qualidade da conferência verificaram-se grandes quebras em termos de vídeo e um atraso significativo no áudio. Os ficheiros que eram partilhados nem sempre eram recebidos no destino, isto é, ou recebia apenas um utilizador ou nenhum utilizador recebia o ficheiro que tinha sido enviado. O mesmo acontecia com as mensagens de *chat*, que nem sempre eram recebidas corretamente. Começou-se a notar uma maior falha em termos de pacotes trocados e entregues entre os diferentes *peers*.

No que diz respeito ao servidor conseguiram-se tirar algumas conclusões fazendo 2 testes distintos. Num teste os clientes lançam eventos para o servidor de forma a que o servidor os tivesse de processar para responder. Estes eventos são lançados exaustivamente por parte de dois clientes, fazendo pedidos ao servidor, numa média de 5 em 5 segundos.

No outro teste foi verificado se realmente o servidor é usado ou requisitado, enquanto os clientes se encontram numa conferência. Como o servidor é um computador com sistema operativo de distribuição *Linux (CentOS)*, foi usado o comando *sar*, que permite medir várias unidades do processador, em que cada resultado deste comando é dado em percentagem. É possível medir a percentagem de utilização do CPU tanto ao nível do utilizador como ao nível do sistema (*kernel*), a percentagem em que não é usado, etc.

Os resultados referentes ao teste dos utilizadores lançarem eventos para que o servidor tivesse de responder encontram-se na Tabela 5.2.

Tabela 5.2: Utilização do CPU no servidor durante alguns eventos entre cliente e servidor.

Tempo	CPU	% user	% nice	% system	% iowait	% steal	% idle
02:14:28 PM	all	1.71	0.00	0.22	0.12	0.00	97.94
02:15:28 PM	all	4.04	0.00	0.24	0.15	0.00	95.56
Average	all	2.88	0.00	0.23	0.14	0.00	96.75

Foi um teste realizado apenas por 2 minutos, onde se verifica uma utilização ao nível do

utilizador de 1.71% no primeiro minuto e ao nível do sistema uma utilização de 0.22%. No segundo minuto houve um aumento considerável ao nível do utilizador, subindo para 4.04% enquanto ao nível do sistema o aumento foi apenas para 0.24%.

Durante a conferência realizada, com 4 utilizadores, os resultados que foram retirados encontram-se na Tabela 5.3.

Tabela 5.3: Utilização de CPU no servidor durante uma conferência de 4 pessoas.

Tempo	CPU	% user	% nice	% system	% iowait	% steal	% idle
02:28:28 PM	all	0.12	0.00	0.02	0.09	0.00	99.78
02:29:28 PM	all	0.14	0.00	0.03	0.10	0.00	99.73
02:30:28 PM	all	0.26	0.00	0.03	0.09	0.00	99.62
02:31:28 PM	all	0.14	0.00	0.02	0.17	0.00	99.68
Average	all	0.16	0.00	0.03	0.11	0.00	99.70

São resultados referentes a 4 minutos. Verificou-se que a utilização de *CPU* é mínima, pois a tecnologia *WebRTC* é uma tecnologia *peer-to-peer*, onde apenas necessita do servidor para estabelecer ligações entre *peers*. A partir do momento que essas ligações estão criadas não é necessário ter um servidor a correr para os *peers* poderem falar entre si.

5.2.2 Cenário 2 - Rede doméstica

Este cenário 2 assemelha-se ao cenário apresentado na secção anterior. O servidor é um computador que se encontra na rede não empresarial mas sim doméstica. Esta rede doméstica ao contrário de uma rede empresarial tem menos utilizadores e é uma rede mais controlada. Neste cenário são realizados dois tipos de testes diferentes, com o objetivo de verificar como se comporta um computador com poder de processamento (*CPU*) inferior e de que forma se comportam os serviços vídeo e áudio numa rede com menos interferências e menos tráfego a circular. Os detalhes dos sistemas usados encontram-se no Anexo C.

Num primeiro teste (Teste 1) foram usados apenas dois computadores (Figura 5.11) sendo realizada uma chamada simples entre eles.

Em termos de clientes notaram-se algumas diferenças entre cada um, visto que eram computadores diferentes e com poderes de processamento diferentes. No computador com menor poder de processamento (PC4) verificou-se o que está representado na Figura 5.12.

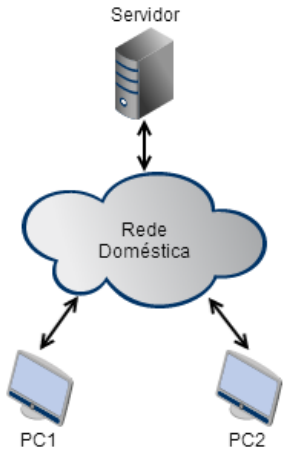


Figura 5.11: Cenário 2 e teste 1.

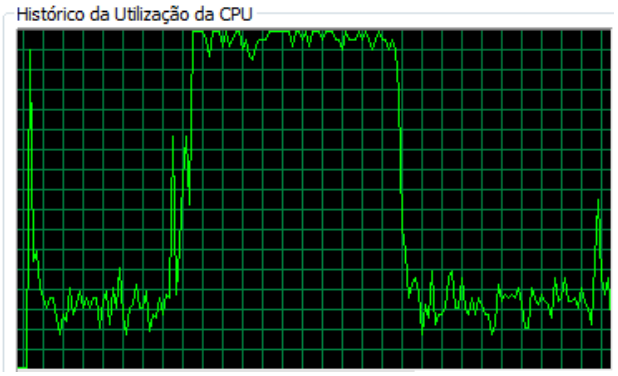


Figura 5.12: Resultados de *CPU* no computador com menor poder computacional.

Consegue-se verificar que a partir do momento que a conferência se iniciou a utilização do CPU subiu consideravelmente atingindo os 100% algumas vezes. A partir do momento que a conferência termina existe uma descida bastante considerável por parte da utilização do CPU. Neste computador notaram-se grandes atrasos no que diz respeito a áudio e bastantes quebras de vídeo. No que diz respeito ao computador com maior capacidade de processamento (PC3), verificou-se que a percentagem de utilização não era tão alta mas que também subiu consideravelmente (Figura 5.13).

No momento em que a conferência termina a utilização volta a estabilizar, mantendo-se basicamente constante tendo apenas picos que dizem respeito a outros processos.

Visto que os testes realizados anteriormente já permitiam tirar conclusões relacionadas com a memória e *CPU*, decidiu-se verificar como é que o vídeo, o áudio, a entrega de ficheiros e *chat*

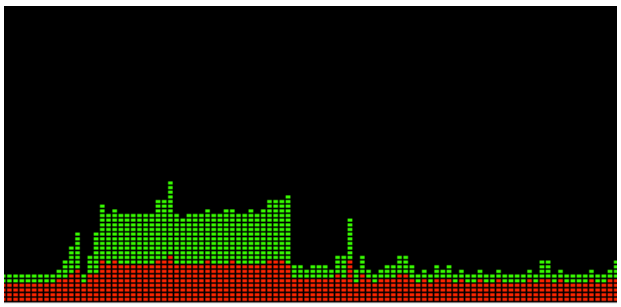


Figura 5.13: Resultados de *CPU* PC4.

se comportavam numa rede mais estável (Teste 2). Usaram-se computadores com características bastante semelhantes para este teste, num cenário igual ao da figura 5.14.

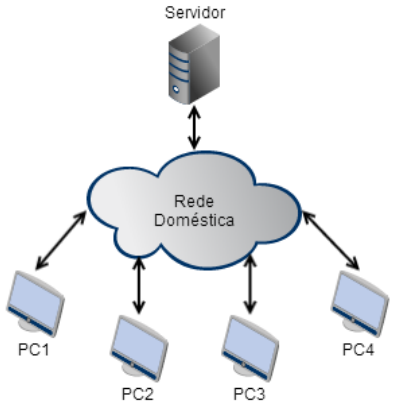


Figura 5.14: Cenário 2 e teste 2.

Foi realizada uma conferência entre os quatro clientes, em que era feita a chamada para todos os clientes ao mesmo tempo. Neste teste verificou-se que havia poucas quebras de vídeo correndo de uma forma fluente e não havia atrasos de áudio. Em relação ao envio e receção de ficheiros já eram enviados e recebidos com sucesso por todos os clientes, o mesmo se passando em relação às mensagens de *chat*.

5.3 Sumário

Neste capítulo foi apresentada a metodologia utilizada para a validação de funcionalidades da *framework*. Foram ainda, descritos os testes de desempenho realizados referentes ao trabalho de-

senvolvido. Foram definidos dois cenários diferentes e em cada um foram executados dois tipos de testes, primeiro com apenas dois computadores e depois quatro computadores. No primeiro cenário utilizou-se uma rede empresarial, enquanto no segundo cenário utilizou-se uma rede doméstica. Os resultados apresentados focam a utilização de *CPU* e de memória nas diversas situações de teste.

Capítulo 6

Conclusões e trabalho futuro

Neste capítulo é efetuada uma síntese do trabalho desenvolvido e apresentado ao longo desta dissertação. São descritas algumas conclusões e é feita uma análise crítica sobre o trabalho que foi realizado e de que forma poderá contribuir para a implementação de novos serviços no futuro. São apresentadas, também, as ideias para o trabalho futuro relacionado com este projeto.

6.1 Resumo do trabalho desenvolvido

As comunicações em tempo real, cada vez mais, são um requisito no dia-a-dia das pessoas. O uso da *Internet* para a realização de chamadas e o envio de mensagens de texto, tornou-se recorrente, devido ao facto de não existir um custo adicional.

Este projeto foi desenvolvido nesse âmbito, visto que a tecnologia *WebRTC* permite aos utilizadores comunicarem com outras pessoas a partir de um *web browser*. Assim definiu-se como objetivo desenvolver uma *Framework* e um cliente, que permitisse utilizar os serviços base do *WebRTC* e serviços de comunicação como presença, lista de contactos, chamadas vídeo e áudio, etc.

Antes de todo este processo de desenvolvimento tecnológico foi necessário haver uma fase de estudo prévio e de análise comparativa entre diferentes tecnologias. Recorreram-se a estudos realizados por outras pessoas, no que diz respeito à tecnologia *WebSockets*, havendo também uma fase de testes inicial, entre as diferentes tecnologias. *WebRTC* foi a tecnologia mais estudada, pois seria o foco principal deste projeto. Foi necessário entender toda a lógica associada à mesma, de como a sinalização era feita, como é que os utilizadores poderiam criar ligações

peer, etc. Para além disso, sendo esta uma tecnologia recente era necessário estar atento às novas atualizações, exemplo disso são os *RTCDataChannels*, que no inicio do desenvolvimento deste projeto não estavam funcionais em qualquer *web browser*. Com o decorrer do tempo foi possível implementar uma solução com *RTCDataChannels* que permitisse aos utilizadores enviar mensagens de texto e partilhar ficheiros, tirando partido dessa *API*. Desta forma, foi possível ter uma solução totalmente funcional com *WebRTC*.

Como explicado no decorrer da dissertação, foram definidos requisitos e casos de uso para a *Framework* e cliente, os quais foram cumpridos com sucesso. Existe neste momento uma solução onde todos os serviços que foram definidos inicialmente como requisitos, se encontram funcionais. É necessário salientar que *WebRTC* é uma tecnologia recente e que não tem ainda muita documentação que possa ser consultada, o que dificultou um pouco o desenvolvimento deste projeto. As principais dificuldades surgiram no desenvolvimento de alguns serviços, visto que todas as tecnologias que foram usadas neste projeto são tecnologias relativamente novas e que nunca tinham sido usadas no ambiente de trabalho. Houve também uma dificuldade inicial, em termos de objetivos, pois o projeto ainda estava a ser definido, logo não estavam definidos casos de uso e requisitos para este projeto.

Inicialmente não foram definidos objetivos em termos de resultados e métricas que tivessem de ser testadas, no entanto, com o decorrer do projeto, tornou-se relevante tirar conclusões, sobre o comportamento de sistemas em termos de memória e de processador, em máquinas diferentes. Utilizaram-se dois cenários idênticos, mas com diferentes características. Num primeiro cenário verificou-se que uma chamada normal entre duas pessoas funciona na perfeição, mesmo estando ligados por *VPN* a uma rede privada. Surgiram mais problemas a partir do momento que entram mais do que 2 pessoas numa chamada. Começaram-se a notar bastantes perdas, os vídeos paravam, o áudio chegava bastante atrasado, os ficheiros não eram partilhados devidamente e as mensagens de *chat* nem sempre eram entregues. Em termos de *CPU* e memória verificou-se que há um aumento considerável da sua utilização a partir do momento que é iniciada uma conferência, pois cada computador tem de aceder aos componentes internos (microfone e *webcam*), como também cada cliente numa conferência tem de criar um novo objeto para reproduzir áudio e vídeo. Num segundo cenário onde foi utilizado um computador mais fraco em termos computacionais, verificou-se que o uso do *CPU* por vezes chegava aos 100%. Isto indica que, para usar a tecnologia *WebRTC*, é necessário ter um computador que tenha uma boa capacidade de processamento. No último teste pretendia-se apenas verificar se o problema da conferência entre quatro pessoas permanecia, isto é, se realmente as quebras de vídeo e o áudio atrasado deixavam de ocorrer e o *chat* e a partilha de ficheiros funcionavam bem. Utilizando uma rede doméstica

verificaram-se algumas melhorias, visto que o vídeo corria mais fluentemente sem grandes quebras e o áudio já não chegava tão atrasado. Em termos de partilha de ficheiros e *chat* verificou-se que maior parte das mensagens e que os ficheiros que eram partilhados, eram recebidos com sucesso por todos os utilizadores. Com isto, pode-se concluir que para usar esta tecnologia convém usar uma rede estável, para que se possa ter uma boa comunicação entre *peers* e que convém ter um computador com maior poder computacional.

6.2 Principais contribuições

As principal contribuição do trabalho desenvolvido, com os objetivos cumpridos nesta fase inicial, foi o estudo mais aprofundado da tecnologia *WebRTC* na empresa e permitir o acesso a essa informação às pessoas da mesma. Para além disso, este projeto permitirá num futuro próximo a implementação de muitos mais serviços de telecomunicações, de uma forma mais facilitada e ao mais alto nível. Pretende-se num futuro próximo e com a evolução da *WebRTC*, criar uma *framework* mais estável, com menos problemas em termos de funcionalidades, e também realizar um cliente interno na empresa que permita a comunicação entre os colaboradores da empresa, de uma forma simples e eficaz. Visto que é uma tecnologia relativamente recente, é necessário tirar proveito disso e desenvolver uma solução num âmbito diferente, com o objetivo de ajudar entidades externas e até mesmo internas.

Como fruto deste trabalho, foi elaborado e submetido um artigo científico na "13ª Conferência de Redes de Computadores"[5], já aceite para publicação. Foi também realizado um artigo sendo publicado na Revista Saber e Fazer Telecomunicações, da empresa em questão.

6.3 Trabalho futuro

Como trabalho futuro a *Framework* terá de ser mais completa e melhorada, isto é, será necessário estar sempre em atualização, modificar e verificar eventuais erros que possam surgir. Será necessário ainda implementar todos os serviços com o *vert.x*, existindo ainda mais serviços a implementar no cliente e na *framework* como, por exemplo, *Voice Activity Detection (VAD)*, implementação de um serviço de desenho para partilha de imagens e eventuais esquemas, por exemplo, numa reunião em conferência de vídeo. Será também implementada uma solução para que haja interoperabilidade entre *web browsers*, isto é, para que utilizadores que usem *web*

browsers diferentes possam comunicar entre eles. No âmbito deste projeto, irá ser desenvolvida uma *API* que permita a interoperabilidade entre diferentes domínios, tornando possível que os domínios totalmente *web*, possam comunicar com domínios *IMS* e até de redes legadas.

Apêndice A

Anexo A - Mensagem SDP

```
remotesdpv=0
   o=-37198933532INIP4127.0.0.1
   s=-
   t=00
   a=group:BUNDLEaudiovideodata
   a=msid-semantic:WMSkqYFk49Vbbwes9dMMSHEThTTqDrDynqHzF8O
   m=audio1RTP/SAVPF111103104081071061051312 6
   c=INIP40.0.0.0
   a=rtcp:1INIP40.0.0.0
   a=ice-ufrag:QEsYYRAb4dutduc+
   a=ice-pwd:dpNwdS2ah7S2OPekpUnccoEh
   a=extmap:1urn:ietf:params:rtp-hdrext:ssrc-audio-level
   a=sendrecv
   a=mid:audio
   a=rtcp-mux
   a=crypto:1AES_CM_128_HMAC_SHA1_80inline:jDmqe2LT0zesIRcx0e4DCdFpld6GdQbF4
   a=rtpmap:111opus/48000/2
```

```
a=fmtp:111minptime=10
a=rtpmap:103ISAC/16000
a=rtpmap:104ISAC/32000
a=rtpmap:0PCMU/8000
a=rtpmap:8PCMA/8000
a=rtpmap:107CN/48000
a=rtpmap:106CN/32000
a=rtpmap:105CN/16000
a=rtpmap:13CN/8000
a=rtpmap:126telephone-event/8000
a=maxptime:60
a=ssrc:3437727872cname:1nazuC/eXhYsFanX
a=ssrc:3437727872msid:kqYFk49Vbbwes9dMMSHEThTTqDrDynqHzF8OkqYFk49Vbbwes9d
a=ssrc:3437727872mslabel:kqYFk49Vbbwes9dMMSHEThTTqDrDynqHzF8O
a=ssrc:3437727872label:kqYFk49Vbbwes9dMMSHEThTTqDrDynqHzF8Oa0
m=video1RTP/SAVPF100116117
c=INIP40.0.0.0
a=rtcp:1INIP40.0.0.0
a=ice-ufrag:QEsYYRAb4dutduc+
a=ice-pwd:dpNwdS2ah7S2OPekpUnccoE
a=extmap:2urn:ietf:params:rtp-hdrext:toffset
a=sendrecv
a=mid:video
a=rtcp-mux
a=crypto:1AES_CM_128_HMAC_SHA1_80inline:jDmqe2LT0zesIRcx0e4DCdFpld6GdQbF4
a=rtpmap:100VP8/90000
```

```
  a=rtcp-fb:100ccmfir
  a=rtcp-fb:100nack
  a=rtpmap:116red/90000
  a=rtpmap:117ulpfec/90000
  a=ssrc:1348122523cname:1nazuC/eXhYsFanX
  a=ssrc:1348122523msid:kqYFk49Vbbwes9dMMSHEThTTqDrDynqHzF8OkqYFk49Vbbwes9d
  a=ssrc:1348122523mslabel:kqYFk49Vbbwes9dMMSHEThTTqDrDynqHzF8O
  a=ssrc:1348122523label:kqYFk49Vbbwes9dMMSHEThTTqDrDynqHzF8Ov0
  m=application1RTP/SAVPF101
  c=INIP40.0.0.0
  a=rtcp:1INIP40.0.0.0
  a=ice-ufrag:QEsYYRAb4dutduc+
  a=ice-pwd:dpNwdS2ah7S2OPekpUnccoEh
  a=sendrecv
  a=mid:data
  b=AS:30
  a=rtcp-mux
  a=crypto:1AES_CM_128_HMAC_SHA1_80inline:jDmqe2LT0zesIRcx0e4DCdFpld6GdQbF4
  a=rtpmap:101google-data/90000
  a=ssrc:352621151cname:1+GP5YzCfsVxjPTm
  a=ssrc:352621151msid:RTCDataChannelRTCDataChannel
  a=ssrc:352621151mslabel:RTCDataChannel
  a=ssrc:352621151label:RTCDataChannel
  remotecandidate"a=candidate:19461855261udp2113937151193.136.92.
19155886typhostgeneration0\r\n"
  remotecandidate"a=candidate:29501536621udp211393715110.112.208.
4255887typhostgeneration0\r\n"
```

```
  remotecandidate"a=candidate:19461855261udp2113937151193.136.92.
19155886typhostgeneration0\r\n"

  remotecandidate"a=candidate:29501536621udp211393715110.112.208.
4255887typhostgeneration0\r\n"

  remotecandidate"a=candidate:19461855261udp2113937151193.136.92.
19155886typhostgeneration0\r\n"

  remotecandidate"a=candidate:29501536621udp211393715110.112.208.
4255887typhostgeneration0\r\n"

  remotecandidate"a=candidate:9817015741tcp1509957375193.136.92.
19159641typhostgeneration0\r\n"

  remotecandidate"a=candidate:37803932941tcp150995737510.112.208.
4259642typhostgeneration0\r\n"

  remotecandidate"a=candidate:9817015741tcp1509957375193.136.92.
19159641typhostgeneration0\r\n"

  remotecandidate"a=candidate:37803932941tcp150995737510.112.208.
4259642typhostgeneration0\r\n"

  remotecandidate"a=candidate:9817015741tcp1509957375193.136.92.
19159641typhostgeneration0\r\n"

  remotecandidate"a=candidate:37803932941tcp150995737510.112.208.
4259642typhostgeneration0\r\n"
```

Apêndice B

Anexo B

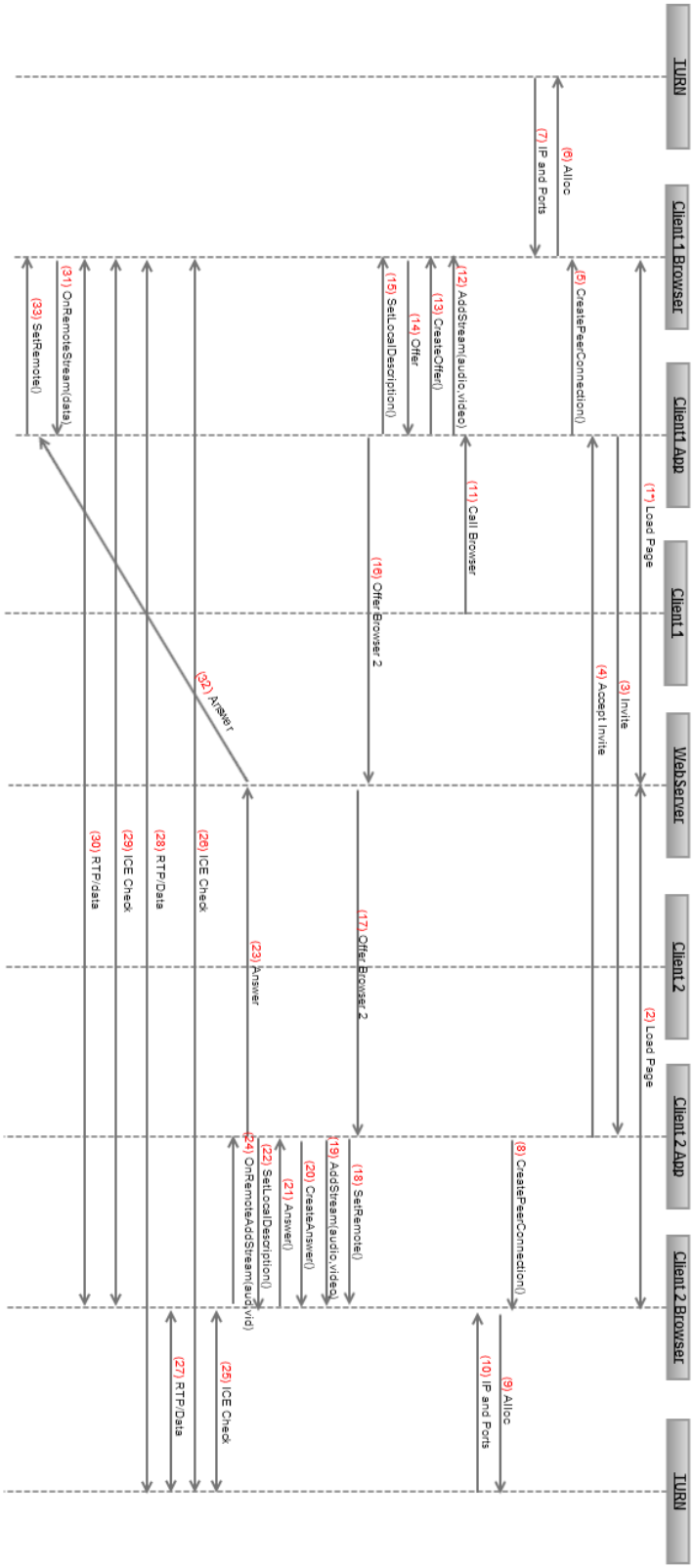


Figura B.1: Funcionamento de sinalização WebRTC.

Apêndice C

Anexo C

C.1 Cenário 1

- PC1
  - Modelo: HP EliteBook 8460p
  - Processador: Intel Core i5 2.5GHz
  - Memória: 4GB
  - Sistema Operativo: Windows 7 64-bits
- PC2
  - Modelo: HP EliteBook 8460p
  - Processador: Intel Core i5 2.5GHz
  - Memória: 4GB
  - Sistema Operativo: Windows 7 64-bits
- PC3
  - Modelo: HP EliteBook 8560p
  - Processador: Intel Core i5 2.5GHz
  - Memória: 4GB

  - Sistema Operativo: Windows 7 64-bits

- PC4
  - Modelo: HP EliteBook 8560p
  - Processador: Intel Core i5 2.5GHz
  - Memória: 4GB
  - Sistema Operativo: Windows 7 64-bits

C.2 Cenário 2

- PC1
  - Modelo: HP EliteBook 8560p
  - Processador: Intel Core i5 2.5GHz
  - Memória: 4GB
  - Sistema Operativo: Windows 7 64-bits

- PC2
  - Modelo: HP EliteBook 8560p
  - Processador: Intel Core i5 2.5GHz
  - Memória: 4GB
  - Sistema Operativo: Windows 7 64-bits

- PC3
  - Modelo: MacBookPro Pro
  - Processador: Intel Core i7 2,66 GHz
  - Memória: 8GB
  - Sistema Operativo: Mac OS X 10.8.5 (Mountain Lion) 64bits

- PC4
  - Modelo: HP Compaq Presario CQ57
  - Processador: AMD Fusion™ APU E-300 Dual Core
  - Memória: 4GB
  - Sistema Operativo: Windows 7 64-bits

Bibliografia

[1] W3C. *A Little History of the World Wide Web.*, 2012. [Online; acedido Fevereiro-Março 2013].

[2] W3C. Information management: A proposal. `http://www.w3.org/History/1989/proposal.html`, 2012. [Online; acedido Fevereiro-Março 2013].

[3] Disruptive Analysis Ltd. Webrtc market status and forecasts - the hype is justified it will change telecoms. `http://www.disruptive-analysis.com/webrtc.htm`, 2012. [Online; acedido Fevereiro-Maio 2013].

[4] Salvatore Loreto and Simon Romano. Real-time communications in the web - issues, achievements, and ongoing standardization efforts. *Internet Computing, IEEE*, 16(5):68–73, 2012.

[5] 12ª conferência de redes de computadores. `http://crc2013.dei.estg.ipleiria.pt/`, Novembro 2013.

[6] Google Inc. Webrtc. `http://www.webrtc.org/`, 2011-2012. [Online; acedido Fevereiro-Setembro 2013].

[7] Inc Google. Web real time communications svn. `https://code.google.com/p/webrtc/source/browse/#svn`, 2011-2013. [Online; acedido Fevereiro-Setembro 2013].

[8] Google Inc. Getting started with webrtc. `http://www.html5rocks.com/en/tutorials/webrtc/basics/`, 2011-2012. [Online; acedido Fevereiro-Setembro 2013].

[9] A.B. Johnston and D.C. Burnett. *Webrtc: APIs and Rtcweb Protocols of the Html5 Real-Time Web*. Digital Codex LLC, USA, 1 edition, 2012.

[10] W3C. Webrtc 1.0: Real-time communication between browsers. `http://www.w3.org/TR/webrtc/`, 2011-2012. [Online; acedido Fevereiro-Setembro 2013].

[11] W3schools. Browser statistics and trends. `http://www.w3schools.com/browsers/browsers_stats.asp`, 1999-2013. [Online; acedido Fevereiro-Setembro 2013].

[12] R. Jesup, Mozilla, S. Loreto, Ericson, M. Tuexen, and Muenster Univ. of Appl. Sciences. Rtcweb data channels. `http://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-04`, mar 2012. [Online; acedido Fevereiro-Setembro 2013].

[13] R. Mahy, P. Matthews, Alcatel-Lucent, and J. Rosenberg. Stream control transmission protocol. RFC 4960 (INTERNET STANDARD) `http://tools.ietf.org/html/rfc4960`, sep 2007. [Online; acedido Fevereiro-Setembro 2013].

[14] E. Rescorla, Inc. RTFM, N. Modadugu, and Stanford University. Datagram transport layer security. RFC 4347 (INTERNET STANDARD) `http://tools.ietf.org/html/rfc4347`, apr 2006. [Online; acedido Fevereiro-Setembro 2013].

[15] J. Postel and ISI. User datagram protocol. RFC 768 (INTERNET STANDARD) `http://tools.ietf.org/html/rfc768`, aug 1980. [Online; acedido Novembro 2012 - Setembro 2013].

[16] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. RFC 3550 (INTERNET STANDARD) `http://www.ietf.org/rfc/rfc3550.txt`, jul 2003. [Online; acedido Dezembro/Janeiro-2012/2013].

[17] D. McGrew, Cisco, E. Rescorla, and Inc. RTFM. Datagram transport layer security (dtls) extension to establish keys for secure real-time transport protocol (srtp). `http://tools.ietf.org/html/draft-ietf-avt-dtls-srtp-07`, feb 2009. [Online; acedido Fevereiro-Setembro 2013].

[18] M. Handley, V. Jacobson, and C. Perkins. Sdp: Session description protocol. `http://tools.ietf.org/html/rfc4566`, 2006.

[19] I. Fette, Google Inc., A. Melnikov, and Isode Ltd. The websocket protocol. RFC 6455 (INTERNET STANDARD) `http://tools.ietf.org/html/rfc6455`, dec 2011. [Online; acedido Dezembro/Janeiro-2012/2013].

[20] Google Inc. Introducing websockets: Bringing sockets to the web. http://www.html5rocks.com/en/tutorials/websockets/basics/, 2010 - 2012. [Online; acedido Fevereiro-Setembro 2013].

[21] Kaazing Corporation. Html5 web sockets: A quantum leap in scalability for the web. http://www.websocket.org/quantum.html, 2013. [Online; acedido Fevereiro-Setembro 2013].

[22] I. Baz Castillo, J. Millan Villegas, Versatica, V. Pascual, and Acme Packet. The websocket protocol as a transport for the session initiation protocol (sip). http://tools.ietf.org/html/draft-ietf-sipcore-sip-websocket-09, jun 2013. [Online; acedido Fevereiro-Setembro 2013].

[23] Y. Suzuki, N. Ogashiwa, Maebashi Kyoai Gakuen College, and D3 Communications. Real-time web communication by using xmpp jingle. http://tools.ietf.org/html/draft-suzuki-web-jingle-00, feb 2012. [Online; acedido Fevereiro-Setembro 2013].

[24] J. Uberti, Google Inc., C. Jennings, and Cisco. Javascript session establishment protocol (jsep). http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03, Aug 2013. [Online; acedido Fevereiro - Agosto 2013].

[25] J. Rosenberg. Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal for offer/answer protocols. RFC 5245 (INTERNET STANDARD) http://tools.ietf.org/html/rfc5245, apr 2010. [Online; acedido Fevereiro-Setembro 2013].

[26] J. Rosenberg, R. Mahy, P. Matthews, D. Wing, and Cisco. Session traversal utilities for nat (stun). RFC 5389 (INTERNET STANDARD) http://tools.ietf.org/html/rfc5389, oct 2008. [Online; acedido Fevereiro-Setembro 2013].

[27] R. Mahy, P. Matthews, Alcatel-Lucent, and J. Rosenberg. Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun). RFC 5245 (INTERNET STANDARD) http://tools.ietf.org/html/rfc5766, apr 2010. [Online; acedido Fevereiro-Setembro 2013].

[28] C. Bran, Plantronics, C. Jennings, Cisco, JM. Valin, and Mozilla. Webrtc codec and media processing requirements. http://tools.ietf.org/html/

draft-cbran-rtcweb-codec-02, mar 2012. [Online; acedido Fevereiro-Setembro 2013].

[29] Doubango Telecom. Smart sip and media gateway to connect webrtc endpoints. http://webrtc2sip.org/, 2011. [Online; acedido Novembro 2012 - Setembro 2013].

[30] Inc. Instant IO. peercdn. https://peercdn.com/, 2013. [Online; acedido Fevereiro-Setembro 2013].

[31] Andyet. Conversat.io. http://blog.andyet.com/2013/feb/22/introducing-simplewebrtcjs-and-conversatio/, 2013. [Online; acedido Fevereiro-Setembro 2013].

[32] Mozilla Developer Network. Bananabread. https://developer.mozilla.org/pt-PT/demos/detail/bananabread, 2012. [Online; acedido Fevereiro-Setembro 2013].

[33] Mozilla. Responsive web typography with webrtc. https://hacks.mozilla.org/2013/02/responsive-web-typography-with-webrtc/, 2013. [Online; acedido Fevereiro-Setembro 2013].

[34] Doubango Telecom. World's first html5 sip client. http://sipml5.org/, 2011. [Online; acedido Novembro 2012 - Setembro 2013].

[35] M. Khan. Realtime/working webrtc experiments. https://github.com/muaz-khan/WebRTC-Experiment, 2012. [Online; acedido Fevereiro-Setembro 2013].

[36] M. Khan. Rtcmulticonnection documentation. https://github.com/muaz-khan/WebRTC-Experiment/tree/master/RTCMultiConnection, 2012. [Online; acedido Fevereiro-Setembro 2013].

[37] M. Khan. Datachannel.js : A javascript wrapper library for rtcdatachannel apis. https://github.com/muaz-khan/WebRTC-Experiment/tree/master/DataChannel, 2012. [Online; acedido Fevereiro-Setembro 2013].

[38] M. Khan. Recordrtc: Webrtc audio/video recording / demo. https://github.com/muaz-khan/WebRTC-Experiment/tree/master/RecordRTC, 2012. [Online; acedido Fevereiro-Setembro 2013].

[39] M. Khan. Rtcall.js — a library for browser-to-browser audio-only calling. https://github.com/muaz-khan/WebRTC-Experiment/tree/master/RTCall, 2012. [Online; acedido Fevereiro-Setembro 2013].

[40] andyet. simplewebrtc. http://simplewebrtc.com/, 2012. [Online; acedido Fevereiro-Setembro 2013].

[41] Google Inc. Interop notes. http://www.webrtc.org/interop, 2012. [Online; acedido Fevereiro-Setembro 2013].

[42] Joyent Inc. Nodejs. http://nodejs.org/, 2012. [Online; acedido Fevereiro-Setembro 2013].

[43] Joyent Inc. Node packaged modules. https://npmjs.org/, 2012. [Online; acedido Fevereiro-Setembro 2013].

[44] Tim Fox. Vert.x - main manual. http://vertx.io/manual.html, 2012. [Online; acedido Fevereiro-Setembro 2013].

[45] Tim Fox. Vert.x. http://vertx.io/, 2012. [Online; acedido Fevereiro-Setembro 2013].

[46] Glyn Normington. Osgi case study: a modular vert.x. http://www.javacodegeeks.com/2012/07/osgi-case-study-modular-vertx.html, 2012. [Online; acedido Fevereiro-Setembro 2013].

[47] Google Inc. What is google app engine? https://developers.google.com/appengine/docs/whatisgoogleappengine, 2012. [Online; acedido Fevereiro-Março 2013].

[48] Digium Inc. What is asterisk? http://www.asterisk.org/get-started, 2012. [Online; acedido Fevereiro-Março 2013].

[49] Digium Inc. Asterisk webrtc support. https://wiki.asterisk.org/wiki/display/AST/Asterisk+WebRTC+Support, 2012. [Online; acedido Fevereiro-Março 2013].

[50] Vert.x. Vert.x vs node.js simple http benchmarks. http://vertxproject.wordpress.com/2012/05/09/

vert-x-vs-node-js-simple-http-benchmarks/, 2012. [Online; acedido Fevereiro-Março 2013].

[51] Shahzad Bhatti. Comparing server side websockets using atmosphere, netty, node.js and vertx.io. http://weblog.plexobject.com/?p=1698, 2012. [Online; acedido Fevereiro-Março 2013].

[52] P. Leach, Microsoft, M. Mealling, LLC Refactored Networks, R. Salz, and Inc. DataPower Technology. A universally unique identifier (uuid) urn namespace. RFC 4122 (INTERNET STANDARD) http://www.ietf.org/rfc/rfc4122.txt, jul 2005. [Online; acedido Dezembro/Janeiro-2012/2013].