Framework e Cliente WebRTC

Vasco 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

Setembro, 2013

Agradecimentos

I would like...

I also...

Abstract

WebRTC is a standard technology, wich allows real time communications between browsers, without install additional plugins. In this way for each device (computer, smartphones, etc) has a installed browser, it's possible to do peer-to-peer real time comunications natively [1]. For example it's possible to do video and voice communications, people talk in a chat, share files, share screen, etc.

This is a recent technology that has grown exponentially, such in implemented solutions and in the browsers compatibility. WebRTC has begun to be a evolutionary technology with a strong growth, where could appear more solutions Over-The-Top (OTT) and the Telecommunications Operators could invest by create their own solutions.

There is no standard implementation for the comunication between WebRTC endpoints, this project is an approach to WebRTC, to identify the its potentiality and in which way could cause some impact in the world of telecommunications. With this project is intended to create a Framework that could help some entities, to create WebRTC applications and services in a higher level. In the same way it's intended to create a WebRTC client that will test the services implemented by the Framework, so it will be the proof of concept.

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 [1]. Exemplo disso são as comunicações de voz, vídeo e também a possibilidade de falar por chat, partilhar ficheiros, partilha de ecrã, etc.

É uma tecnologia relativamente recente mas tem vindo a crescer exponencialmente, tanto a nível de soluções implementadas, como também a nível de compatibilidade de *web browsers*. Desta forma *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 gerar um grande impacto ao nível de oferta de serviços.

Ainda não está definida uma implementação normalizada para a comunicação entre *endpoints WebRTC*, este projeto consiste numa abordagem à tecnologia *WebRTC*, de forma a identificar as potencialidades desta tecnologia e de que forma podem ter um impacto no mundo telecomunicações. Pretende-se ainda desenvolver uma Framework que terá como objetivo, tornar mais fácil a criação e implementação de serviços *WebRTC*, que servirá como uma solução de comunicação entre vários clientes. A título de exemplo será desenvolvida, paralelamente, uma aplicação com a implementação de alguns desses serviços.

Conteúdo

Lista de Figuras

Lista de Tabelas

List of Acronyms

API	Application Programming Interface
WebRTC	Web Real Time Communications
IETF	Internet Engineering Task Force
W3C	World Wide Web Consortium
P2P	Peer-to-Peer
IP	Internet Protocol
	...

Capítulo 1

Introdução

1.1 Introdução

A comunicação entre sistemas começou desde cedo a ser pensado, 1945, onde foi apresentado 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, no mesmo ano é falado no *hypertext* [2]. Desde então e devido à criação da ARPANET começaram a surgir mais ideias e protocolos de comunicação, onde foi apresentado também um sistema de email. Em 1989 Tim Berners-Lee apresenta a proposta *Information Management: A Proposal*, que foi aceite. Então no ano seguinte Tim começa a trabalhar na implementação de um *browser* e num editor que permiti-se o uso de hyperlinks [2]. Todo este processo foi evoluindo ao longo dos anos e nos dias de hoje a W3 ou simplesmente a Web, é imprescindível para o dia a dia das pessoas, seja no telemóvel ou num computador. Toda esta grande evolução tem gerado várias discussões sobre aplicações nativas *"versus"* aplicações web. 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 browser (Gmail, Drive Google, PlayStore, etc). Para além desses serviços e com a ajuda da W3C e da IETF, foi possível criar um projecto que permiti-se a comunicação áudio e vídeo entre browsers, a tecnologia WebRTC. Com esta tecnologia é possível a criação de aplicações, que permitam realizar comunicações em tempo real entre várias pessoas, como em algumas aplicações nativas (Skype). Esta tecnologia surgiu recentemente, sendo apresentada em 2012 pela Google. Desde então tem evoluído de uma forma bastante rápida, permitindo o surgimento de várias soluções utilizando as suas APIs. Prevê-se um futuro próximo uma evolução exponêncial do uso desta tecnologia não só ao nível de computadores,

mas também em tablets e smartphones, figura 1.1.

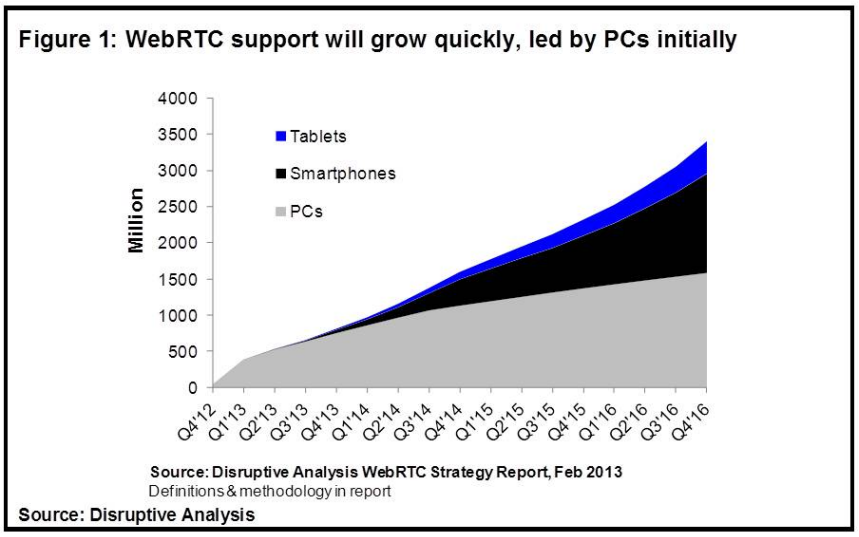


Figura 1.1: Estimativa da evolução da tecnologia WebRTC [3].

1.2 Motivação e Objectivos

A tecnologia *WebRTC* tem vindo a evoluir desde que a Google e a W3C começaram a trabalhar na sua normalização. Têm vindo a ser desenvolvidas várias aplicações WebRTC, que serão apresentadas no decorrer deste projecto. O maior desafio dessas soluções passa por saber de que forma é feita toda a parte da sinalização, isto é, que tecnologias são usadas na troca de sinalização entre peers.

A figura 1.2 apresenta o exemplo da principal motivação deste projecto. Na camada aplicacional não existe um mecanismo normalizado para a troca de informação de sinalização entre peers. Na tecnologia WebRTC é necessário haver uma negociação e sinalização entre os diferentes peers, desta forma é possível criar ligações peer entre eles, para que a informação media e data possa ser trocada entre eles. Este projecto consiste numa análise mais aprofundada à tecnologia WebRTC e também a tecnologias que ajudem na troca de informação de sinalização entre os diferentes peers, como por exemplo, WebSockets.

Este projecto tem como objectivos o estudo de várias tecnologias WebSockets e fazer uma análise comparativa entre essas diferentes tecnologias. Desta análise comparativa serão tomadas

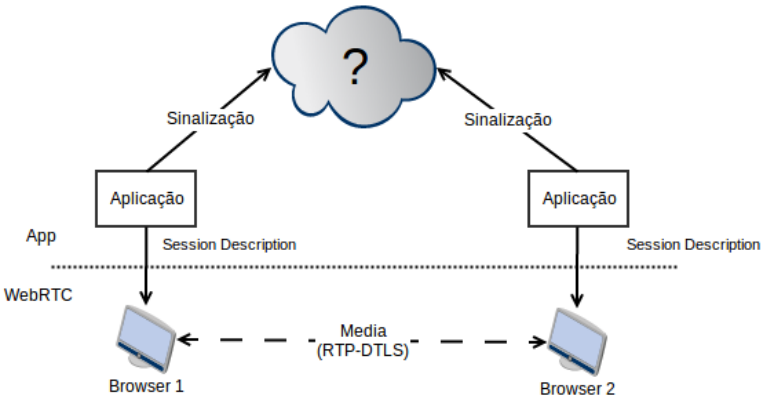


Figura 1.2: Exemplo de motivação.

algumas decisões sobre as tecnologias que devem ser usadas, desta forma será desenvolvida uma Framework que tem por objectivo principal, facilitar a outras entidades o desenvolvimento de serviços WebRTC (presença, histórico de chamadas, conferência, etc), a um alto nível. Para isso foram definidas fases de desenvolvimento:

- Estudo da tecnologia WebRTC e de soluções que permitam a comunicação entre browsers;
- Definir casos de uso para a Framework e prova de conceito;
- Desenvolvimento da Framework e Prova de Conceito;
- Testes e análises de resultados.

A prova de conceito servirá essencialmente para verificar que os serviços desenvolvidos na Framework se encontram operacionais e também servirá como um cliente WebRTC na empresa em que o projecto se encontra inserido. Os testes que serão realizados serão mais ao nível de memória e CPU, tanto nos clientes como nos servidores. Desta forma poder-se-á verificar qual o peso computacional da tecnologia WebRTC nas diferentes máquinas. Desta forma é possível 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.

1.3 Principais contribuições

Tem surgido uma discussão sobre as aplicações Web "versus"aplicações nativas, onde maior parte das análises realizadas são em termos de performance e de que forma compensa a criação de aplicações Web. Neste sentido existem cada vez mais aplicações Web que fazem o mesmo, que várias aplicações nativas, onde a única diferença, é que as aplicações Web correm sobre um browser. Neste sentido surgiu a tecnologia WebRTC que permite realizar comunicações entre diferentes clientes em tempo real. Como este projecto está a ser desenvolvido num âmbito empresarial, deve tirar-se partido da WebRTC ser uma tecnologia recente e estudá-la, de forma a saber que vantagens poderá trazer no futuro das telecomunicações. Com este projecto pretende-se realizar uma Framework que permita a outra entidades desenvolver soluções WebRTC, a um mais alto nível e de uma forma mais simplificada. Para além disso será desenvolvido um cliente WebRTC, para verificar as funcionalidades da Framework desenvolvida. Desta forma é possível explorar e estudar mais aprofundadamente a tecnologia WebRTC, analisando as vantagens e desvantagens desta tecnologia, como também estar ocorrente de todas as actualizações e modificações feitas na tecnologia.

1.4 Dissertation layout

In the present Chapter 1 - ...

In Chapter 2- ...

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, os protocolos usados e ainda os *codecs*. Serão descritos os métodos de sinalização que são normalmente usados neste tipo de soluções.

2.1 WebRTC

As *Real Time Communications* – mencionadas como *RTC* durante este projeto – 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*) [1].

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*. Na secção seguinte é apresentada a *API WebRTC* e os conceitos que a constituem, a compatibilidade atual desta tecnologia nos *web browsers* e quais os protocolos usados para comunicação e para sinalização.

2.1.1 WebRTC API

Esta tecnologia foi desenvolvida para Web browsers, o que fez com que a linguagem adotada fosse JavaScript.Desta forma 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 [4, 5].

PeerConnection

PeerConnection permite que dois utilizadores comuniquem directamente, browser-to-browser. Para estabelecer esta ligação e haver uma negociação de sinalização, é necessário que haja um canal de sinalização. Este é fornecido por um script implementado num servidor Web, utilizando WebSockets ou XMLHttpRequest. Este mecanismo usa o protocolo 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 NAT e firewalls [4, 5, 6].

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. 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, como mostra o seguinte exemplo Para a transmissão das streams são usados protocolos como SRTP, RTP e RTCP para monitorização de transmissão de media. O Protocolo DTLS é usado como uma chave de SRTP e para gestão de associações [4, 5, 6].

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 encap-

sulado em DTLS. Desta forma existe uma solução para NAT com confidencialidade, autenticação da fonte e integridade dos dados que sejam necessários transferir. 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 [4, 5, 6].

2.2 Compatibilidade

Por WebRTC ser uma tecnologia Web, esta não depende apenas de si própria. Desta forma é necessário garantir que haja compatibilidade nos diferentes Web browsers. É então necessário garantir que os web browser suportem HTML5. O HTML5 é um novo standard para a syntax HTML, que suporta novas funcionalidades, como a redução da utilização de plugins adicionais como FLASH, novas tags que permitiem 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 [4].

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 apresentada permite perceber quais os browsers mais utilizados entre janeiro e maio de 2013.

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

Tabela 2.2: Percentagem dos *web browsers* usados [7].

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%

2.3 *Network Address Translate (NAT)*

Comparativamente com o protocolo SIP, WebRTC tem problemas NAT, 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.

2.3.1 *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 proncipais deste protocolo verificam conectividade entre dois agentes ou retransmitir pacotes de dados entre dois agentes. Este protocolo é usado na tecnologia WebRTC para estabelecer uma sessão entre dois clientes, onde o browser funciona como cliente STUN e o servidor web como servidor STUN. São enviados á 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 [8].

2.3.2 *Interactive Connectivity Establishment (ICE)*

ICE é uma solução para streams de media baseadas em UDP, que sejam estabelecidas num modelo pedido/resposta. Este protocolo permite a vários clientes trocarem informação media através de NAT. Esta informação tem incluídos endereços IP e portos, que por norma são problemáticos no que diz respeito sistemas com NAT. O protocolo ICE é um protocolo que usa uma técnica conhecida como “hole punching”, que foi definida para que os utilizadores possam trocar infor-

mação 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 DoS, isto acontece porque não é enviada media enquanto a informação ICE não seja trocada [9].

2.3.3 *Traversal Using Relays around NAT (TURN)*

É possível a partir de técnicas "hole punching" descobrir um caminho directo 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 directo 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 haja como um TURN server, de forma a que os pacotes que estão a ser transmitidos, possam ser reencaminhados para o seu destino. No caso do WebRTC o 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 [10].

2.4 *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. Para quem desenvolve este tipo de aplicações, terá de estar ciente da escolha dos protocolos de transporte que melhor se adequam à sua solução, como também dos melhores mecanismos de compressão de voz e de vídeo, por forma a obter o melhor desempenho fim-a-fim. Os codecs são hardware e software. Em termos de hardware são usados codificadores e descodificadores, que por sua vez poderão correr 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 PCMA/PCMU, telefone event e Opus como codecs de áudio. No que diz respeito a codecs de vídeo devem ser garantidos 10 frames por segundo, ter resolução 320x240 devendo suportar também resoluções de 1280x720, 720x480, 1024x768, 800x600,640x480 e 640x360 [11]. RTCWeb definiu dois codecs áudio obrigatórios:

- *Opus*;
- G.711.

Enquanto de vídeo não foram definidos quais os que são obrigatórios, por isso depende da implementação do browser. A tabela seguinte mostra quais os codecs que cada um dos browsers usa, tanto áudio como vídeo.

Tabela 2.3: Tabela de suporte codecs nos diferentes browsers [12].

	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

2.5 Protocolos DataChannel

Para além de media na tecnologia WebRTC, existe também a possibilidade de trocar media entre os utilizadores de uma sessão, para isso foram definidos canais de dados (Data Channels). Os Data Channels usam quatro tipo de protocolos, SCTP para controlo de streams, DTLS para segurança dos dados e UDP/ICE que são definidos como protocolos da camada de transporte.

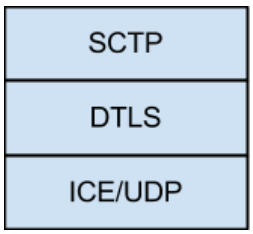


Figura 2.1: Pilha *Data Channel* [13].

A figura 2.1 representa a stack base para os Data Channels, em que SCTP se encontra encapsulado em DTLS e que por sua vez se encontram encapsulados em ICE/UDP [13].

2.5.1 Stream Control Transmission Protocol (SCTP)

SCTP é um protocolo que surgiu devido às necessidades das aplicações, em que o TCP era limitativo. TCP é um protocolo que tem uma natureza stream-oriented, permite a entrega fiável e ordenada de dados, não permite grandes transferências de dados e é vulnerável a ataques DoS. Todas estas caracteristicas são um bocado limitativas, no que diz respeito às exigências das aplicações [14]. SCTP é um protocolo de transporte fiável que oferece uma fragmentação de dados consoante o MTU definido. Está livre de erros e de dados duplicados, tem entrega de dados sequencial, envia múltiplas streams apenas num pacote SCTP, o que permite uma maior entrega de dados. É tolerante a falhas e para além disso tem um mecanismo de controlo de congestionamento de tráfego. Todas estas características permitem satisfazer as necessidades destas novas aplicações, sem os inconvenientes do TCP [14].

2.5.2 Datagram Transport Layer Security (DTLS)

DTLS é um protocolo usado para permitir a segurança de tráfego na rede. Este protocolo é designado como "TLS over datagram", por norma é usado para 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 UDP. Como este tipo de protocolos não permitem o uso TLS para protecção de dados, foi necessário desenhar uma solução. DTLS é uma variante do TLS, para protocolos que usam UDP. A sua definição permitiu usar maior parte da infraestrutura e código que já tinha sido definido no TLS [15]. DTLS foi então desenhado de forma a proteger dados em aplicações de comunicação, que não oferece 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 [15].

2.5.3 User Datagram Protocol (UDP)

User Datagram Protocol é um protocolo de comutação de pacotes num sistema de rede entre vários sistemas. É um protocolo que normalmente é usado para transportar pequenos pacotes de dados e também para trocar media em sessões RTP. As aplicações de comunicação em tempo real não são tolerantes a falhas ou a atrasos, logo é necessário garantir uma entrega rápida de pacotes, para diminuir ao máximo esses fatores. 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 [16]. Este protocolo é fornecido pelo sistema operativo sob o web browser.

2.5.4 SRTP - DTLS

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 (voz e vídeo). Atualmente é usado por diferentes aplicações fim-a-fim, tais como, Simple Multicast Audio, conferências áudio e vídeo, Mixers and Translators e Layered Encodings. RTP não garante QoS ou entrega de pacotes, consegue no entanto uma reconstrução temporal, detecção de perda, segurança e identificação de conteúdo [17, 18]. O SRTP é um protocolo do formato do RTP que fornece confidencialidade, autenticação de mensagens e a proteção de tráfego e de controlo para RTP. 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, pois SRTP não permite uma gestão de chaves nativo, desta forma é necessário recorrer a mecanismos externos. DTLS é um protocolo que integra essa gestão de chaves, como também negociação de parâmetros e transferência segura de dados. Desta forma é possível ter uma solução que combina a performance e encriptação do SRTP, com a gestão de chaves e associações [18]. Esta é a solução que WebRTC usa para a encriptação de dados RTP e gestão de chaves entre dois clientes.

2.5.5 Session Description Protocol (SDP)

Session Description Protocol é um protocolo, como o próprio nome indica, serve para a descrição de uma sessão entre dois clientes. Quando se inicia uma conferência multimédia, chamadas VoIP, streaming de vídeo, ou outro tipo de sessões, é necessário ter em conta algumas características, como detalhes de media (ex. codecs), endereços de transporte e outros dados da sessão que precisam de ser trocados entre os participantes [10]. É um protocolo completamente texto com uma grande quantidade de informação, mas que é necessário e essencial para estabelecer uma sessão de media entre os clientes. Este protocolo é usado numa grande quantidade de aplicações e diferentes tipos de redes. Apesar disto 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 informação sobre o contacto da pessoa que está responsável pela sessão [19]. No caso da tecnologia WebRTC, a informação é codificada num objeto designado por RTCSessionDescription [4, 6]. Este objeto serve para descrever a sessão entre dois clientes, 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 que faz a oferta suporta. No caso de answer, é uma resposta originada caso seja recebida a offer e leva a mesma informação, mas neste caso do cliente que cria a answer [4, 6].

2.6 Sinalização

Ao contrário do que acontece com uma infraestutura 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 necessário haver uma negociação de parâmetros da sessão, falados na secção SDP. A definição de um método de utilização 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.

2.6.1 SIP over WebSockets

O protocol WebSockets, secção 2.7, 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, que permite 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 [20]:

- SIP WebSocket Client: entidade capaz de abrir ligações de websockets saída para o servidor e comunicam 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.

2.6.2 Jingle over WebSockets

XMPP Jingle define um protocolo XMPP para a gestão de sessões media peer-to-peer e permite fazer a gestão de múltiplos conteúdos simultaneamente [21]. Este protocolo foi definido principalmente para comunicações em tempo real, como presença e chat. 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.

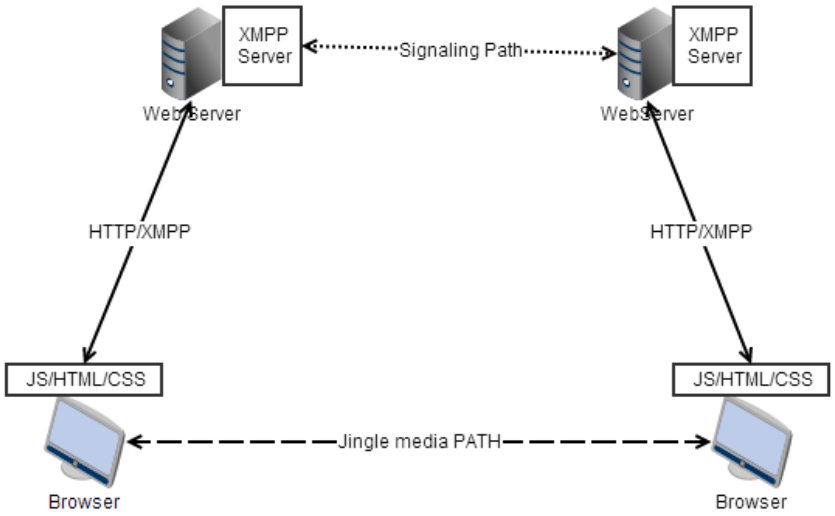


Figura 2.2: Processo Jingle [21].

Em comparação ao que acontece com a sinalização com outros protocolos (ex. SIP), a sinalização é feita a partir de um caminho XMPP e caminho de media é do tipo Jingle media. Como este tipo de técnicas são baseadas em HTTP long polling, que transporta demasiado overhead. Uma boa solução era evitar XMPP sobre HTTP, o que indica que seria necessário usar XMPP nativo. Porém os 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 permitira enviar informação XMPP entre os vários browsers e os servidores. Desta forma a comunicação e a troca de mensagens entre as várias identidades era

feita de uma forma mais robusta e eficiente [22].

2.6.3 *JavaScript Session Establishment Protocol (JSEP)*

JavaScript Session Establishment Protocol define de que forma as aplicações JavaScript interagem com as funções RTC. Este protocolo está responsável pela forma de como os clientes podem recolher informação sobre o tipo de media que suportam e que codecs usam, isto fica definido num objeto SDP RTCSessionDescription. JSEP fornece mecanismo de criação de offers e answers, como também de que forma podem ser aplicadas numa sessão [22]. 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 a cargo do utilizador.

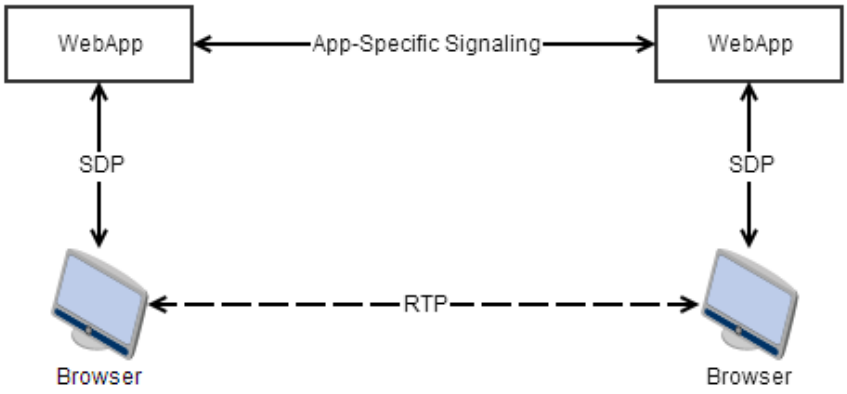


Figura 2.3: Modelo *JSEP* [22].

A imagem acima mostra o modelo utilizado pelo JSEP, onde as mensagens entre o browser a aplicação são objetos SDP, e o protocolo que é usado para transmitir essa informação é especificado na camada aplicacional.

2.7 WebSockets

A criação de aplicações Web tem sido 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 timer que atualizasse a página, para ver os dados atualizados. As tecnologias que permitem que o servidor envie dados atualizados mal os recebam, foram usadas durante algum tempo. Uma das soluções era a sondagem longa, que permite a abertura de uma ligação ao servidor, até que este receba uma resposta. 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 pelo várias ligações TCP para cada cliente, uma para enviar informação e várias para cada mensagem recebida [23, 24]. WebSocket Protocol foi desenhado para este tipo de problemas. É um protocolo que fornece uma comunicação bidirecional, 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 [23, 24] .

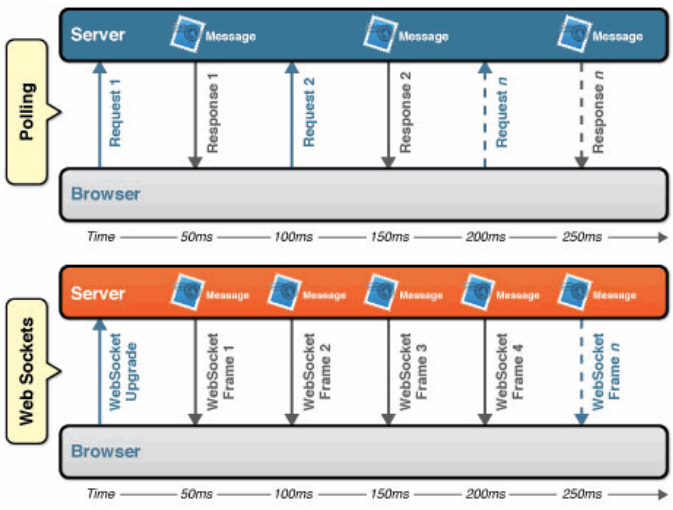


Figura 2.4: Comparação de latência entre *polling* e aplicações *Websockets* [25].

Na figura 2.4 estão representados dois tipos de ligações: polling (HTTP) e WebSockets. Uma ligação polling o browser necessita de estar sempre a fazer pedidos, pois o servidor só 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, sem ter que receber qualquer pedido adicional por parte do browser. Esta ligação entre o servidor e o browser é designada como uma ligação persistente, pois sempre que um deles precisar de enviar qualquer informação podem fazê-lo a qualquer momento.

2.8 Sumário

Este capítulo descreveram-se, de um modo geral, a arquitectura WebRTC que APIs constituem essa tecnologia (PeerConnection, Media Streams, RTC Data Channels) e de que forma podem ser usadas. Falou-se um pouco da compatibilidade desta tecnologia e também da tecnologia HTML5 nos browsers, visto que por norma são usadas em conjunto para desenvolver este tipo de aplicações. Foi discutido o problema de NAT nesta tecnologia 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 é importante, visto que os diferentes browsers podem usar diferentes codecs, e deve haver uma interecção 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. Descreveram-se ainda 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 descreveram-se o que são os WebSockets e para que servem e de que forma se diferenciam dos tradicionais servidores Web.

Capítulo 3

Soluções Existentes

Neste capítulo serão apresentadas algumas soluções WebRTC, onde que cada uma tem uma funcionalidade diferente, mostrando assim 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 ajudam à implementação de serviços de uma forma mais simples, e que podem ser integradas nesta solução. Serão ainda feitas introduções a plataformas de WebSockets, que têm algum potencial e poderão ser usadas para transportar informação de sinalização entre os diferentes clientes. Por fim são analisados alguns gateways SIP, que permitem a interoperabilidade entre este tipo de soluções e clientes SIP que não são soluções Web.

3.1 Soluções existentes

3.1.1 PeerCDN

Esta é uma solução de entrega de conteúdos peer-to-peer. Esta solução tem por objectivo reduzir os custos da largura de banda e do servidor. É possível 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. Desta forma é reduzida em grande parte a largura de banda utilizada na parte do servidor, reduzindo assim os custos [26]. Neste solução é usada a API WebRTC DataChannel, para estabelecer a troca de dados entre os vários peers. Solução totalmente desenvolvida em JavaScript [26].

3.1.2 Conversat.io

Conversat.io é uma solução para realizar chamadas entre duas a seis pessoas. O funcionamento é muito simples, a aplicação cliente liga-se a uma porta especifica do servidor (socket.io) que permite a executar a parte de sinalização e negociação entre os peers. É pedida a permissão para aceder aos dispositivos internos do computador (câmara e vídeo). É criado um "room" caso não exista, caso contrário a pessoa junta-se a esse "room ", onde estarão outras pessoas com quem a pessoa em questão queira falar [27]. 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 [27].

3.1.3 BananaBread

É um jogo em 3D que corre totalmente em browser. É desenvolvido no 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 [28].

3.1.4 Responsive Web Typography with WebRTC

Esta solução tira proveito da tecnologia WebRTC para criar sítios web que sejam responsivos em termos de tamanhos dos objetos, principalmente letras. Os exemplos que existem até ao momento com capacidade de resposta, guiam-se apenas pelo tamanho dos ecrãs ou então pela densidade de pixels. Para além disso quem desenvolve este tipo de soluções, acaba por dizer aos utilizadores de que forma devem ser usadas nos diferentes dispositivos [29]. É preciso ter em atenção outros fatores como por exemplo, meio ambiente onde o utilizador se encontra, que tipo de necessidades o utilizador necessita, etc. Este exemplo em específico concentra-se principalmente na distância a que os utilizadores se encontram do dispositivo. Isto é, foi 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 [29].

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 IMS, permitindo realizar e receber chamadas vídeo ou voz [30]. 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 DMTF usando SIP INFO;

Encontra-se disponível para Chrome, Firefox Nightly, Firefox stable, 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 [30, 12], consegue-se ter uma ligação a uma rede SIP ou uma rede PSTN para comunicar com outros dispositivos.

3.2 Bibliotecas JavaScript

Apesar de WebRTC ser uma tecnologia recente, há quem se encontre de momento a trabalhar e a implementar algumas soluções. Algumas têm por base bibliotecas Javascript que foram desenvolvidas 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 objetivo ajudar no desenvolvimento de aplicações deste tipo. Existem quatro bibliotecas diferentes neste repositório RTCMultiConnection.js, DataChannels.js, RecordRTC.js e RTCall.js [31].

RCMultiConnection.js

Com esta biblioteca é possível desenvolver alguns serviços avançados de uma forma mais simples, como [32]:

- 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);
- Possibilidade de banir utilizadores;
- Detecção de presença (orientado a eventos onde é verificado quando um utlizador entra ou utilizador sai, sem vários estados de presença);

Esta biblioteca usa firebase por defeito, 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 actualizados é possível actualizar essa informação em todos os clientes que estão ligados [32]. 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 algumas destas funcionalidades [33]:

- Mensagens de texto directas entre utilizadores;
- Banir ou rejeitar qualquer utilizador;
- Sair de uma sessão ou encerrar uma sessão completa;

- O tamanho dos ficheiros é ilimitado;
- Quantidade de dados é ilimitada;
- Detecção de presença (orientado a eventos onde é verificado quando um utlizador entra ou utilizador 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. Stream de áudio é gravada no formato .wav, stream vídeo guardada em formato WebM e imagens animadas em .GIF. Este 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 objecto Blob [34]. Esta biblioteca pode ser útil para gravar chamadas entre clientes, ou até mesmo gravar uma mensagem de vídeomail 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. Exemplo disso será 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 [35].

3.2.2 SimpleWebRTC

É uma biblioteca Javascript que permite desenvolver aplicações de comunicações em tempo real em ambientes Web (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 camara e microfone deve ser feito

Tabela 3.1: Perfixos das interfaces nos diferentes *browsers* [36].

W3C Standard	Google Chrome	Mozilla Firefox
getUserMedia	Não Suporta	Suporta >v17
RTCPeerConnection	Não Suporta	Suporta >v22
RTCSessionDescription	Não Suporta	Suporta >v22
RTCIceCandidate	Não Suporta	Suporta >v22

automaticamente ou não. Para além disso é uma biblioteca que é orientada a eventos, para saber se o utilizador está pronto e se está à espera que alguém se junte à sessão. É necessário o uso de um servidor socket.io, que servirá como servidor de sinalização, que pode ser implementado localmente ou num servidor web.

3.2.3 Adapter.js

As chamadas aos sistema de cada Web browser é 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 que tipo de funções são usadas para desenvolver uma solução WebRTC em diferentes 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.

3.3 Sinalização

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, 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 falado na secção 2.7, é a solução mais aceitável e que maior parte das aplicações deste tipo usam, 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. Também será apresentada uma das soluções da google, App Engine.

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, são assíncronos o que permite diminuir o overhead e aumentar a escalabilidade [37]. 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 objectivo funcionar completamente em tempo real, como por exemplo, streaming de voz ou vídeo, ou então serviços de chat, partilha de ficheiros [37]. 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) [38]. 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 utilize uma API tipo WebSockets, é o socket.io. Permite que WebSockets e tempo real esteja presente em qualquer browser, para além disso ainda fornece multiplexing, escalabilidade horizontal e codificação e descodificação em JSON [38].

3.3.2 Vert.x

É uma framework da próxima geração, que corre em JVM, é orientada a eventos e assíncrona. Implementa um modelo de concorrência e 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, diz que é uma framework poliglota, suportando desenvolvimento em [39]:

- Java
- JavaScript
- Groovy
- Ruby
- Python

- Scala

Para além das características apresentadas anteriormente, esta framework aborda alguns conceitos que são importantes para o desenvolvimento de aplicações [40, 39].

- Verticle: Um verticle é definido por ter uma função principal (main), que corre um script especifico. 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 especifico 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: é uma caracteristica do Vert.x bastante importante. Este conceito 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. Pode-se verificar na figura 3.1 uma arquitetura específica do Vert.x.

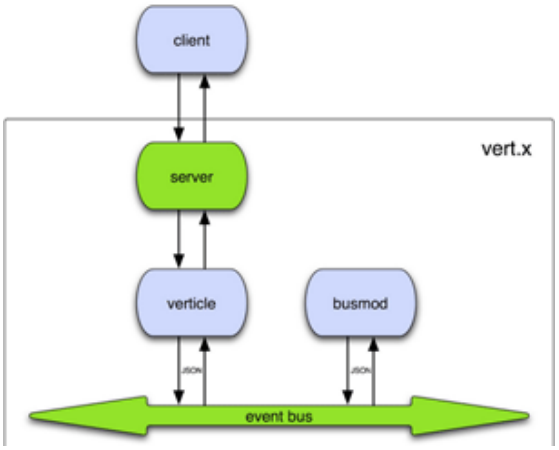


Figura 3.1: Funcionamento geral da aplicação Vert.x [41].

Esta imagem tem como objectivo mostrar que é possível a interacção entre os diferentes módulos e verticles, através do Eventbus

- 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.

Em termos de core, são fornecidos vários serviços que são chamados directamente em verticles e que podem ser implementados e módulos [40].

- 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 App Engine Google

Google App Engine é uma ferramenta que permite correr aplicações web na infraestrutura da Google. Normalmente quando se desenvolve uma aplicação é necessário exista um administrador que trate de manter os serviços sempre ativos, que garanta que o servidor não vai abaixo, etc. Com App Engine é requisitado um serviço pago, à empresa Google que garante redundância, gestão de serviços e basta fazer upload da aplicação para esta ficar a correr. Este serviço suporta várias linguagens de programação tais como: Java, Python, PHP e 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 [42]:

- Serviço de web dinâmico com suporte a tecnologias web comuns
- Armazenamento com persistência com consultas, transações e ordenação balanceamento de carga e escalável
- 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. Desta forma 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 do software e também da localização física do servidor web [42].

3.4 Gateways para SIP

Exisem Gateways que permitem a comunicação de clientes totalmente WebRTC com clientes SIP, isto é, é possível passar de um ambiente totalmente Web para um ambiente totalmente SIP e vice-versa. Este tipo de soluções podem 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 Internet Protocol (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. Usado em mais de 170 países. Permite 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, queira [43].

Na versão 11 do Asterisk foi adicionado o suporte à tecnologia WebRTC. Foi 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 e para além disso foi adicionado o suporte dos protocolos ICE, STUN e TURN a 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 iimplementada na versão 10 [44]. 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 [44]. Apesar destes módulos estarem implementados no Asterisk é necessário que sejam ativados e configurados devidamente.

3.4.2 WebRTCtoSIP (sipML5)

O Webrtc2sip serve como um gateway, permitindo assim que um browser funcione como um telefone/telemóvel, que permite realizar chamadas tanto áudio como vídeo e ainda enviar SMSs, sobre qualquer tipo de rede PSTN e SIP. 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 mídia das aplicações, comunica com dois componentes do webrtc2sip RTCWeb Breaker e Media Coder. RTCWeb Breaker trata de fazer a conversão de streams media, para que possam comunicar com dispositivos finais que não suportem caraterísticas e protocolos como ICE and DTLS/SRTP. A figura 3.3 mostra como é feita essa conversão.

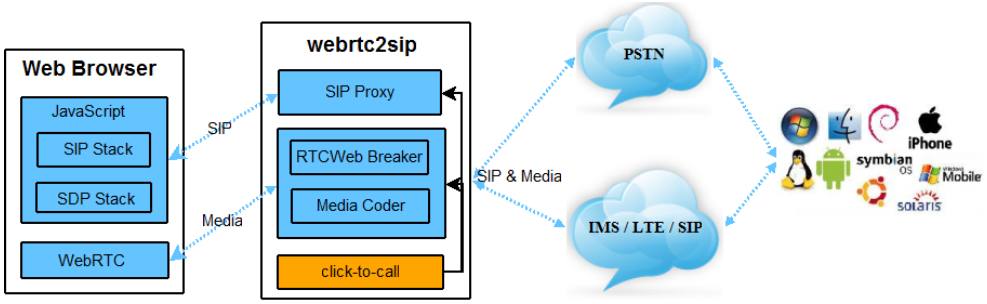


Figura 3.2: Arquitectura SipML5

Figura 3.3: RTCWeb Breaker SipML5 [12].

A normalização de RTCWeb definiu dois MTI (Mandatory To Implement) codecs áudio, opus e g.711. Apesar de ainda haver bastante discussão sobre quais os codecs de vídeo a usar, sujeita-se à escolha entre VP8 e H.264. Chrome, Mozilla, Opera provavelmente irão usar 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.

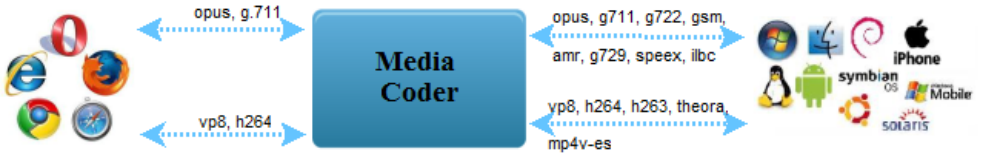


Figura 3.4: Media Coder SipML5 [12].

3.5 Experimentação e selecção de soluções existentes

Como todos os projetos deve haver uma fase de estudo, investigação e definição de requisitos, para desenvolver uma solução capaz de forma organizada. Este capítulo vem dar enfâse ao estudo das várias tecnologias analisadas. Irá ser feita 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. Será dada tanto à parte de servidor como cliente, analisando diferentes soluções que por fim levará à escolha da melhor.

3.5.1 Vert.x vs Node.js

Nesta secção serão analisadas soluções WebSockets que funcionará 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 capacidade de desenvolvimento, tipo de linguagem, escalabilidade, desempenho de servidores, etc.

Vert.x vs node.js simple HTTP benchmarks

Este estudo 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.

A imagem acima 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 quase mais 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.

O segundo teste, passa por testar 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.

A imagem x 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 performance do node.js.

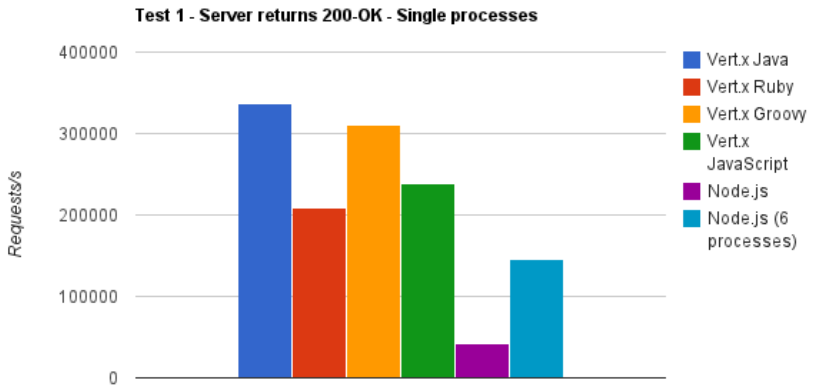


Figura 3.5: Resultados do teste de respostas 200-OK [45].

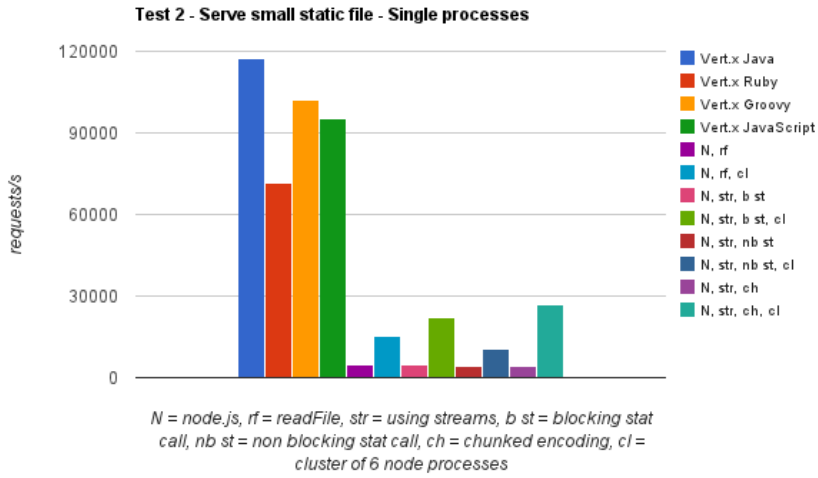


Figura 3.6: Resultados do teste de ficheiro 72bytes [45].

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. Apesar de existem mais essas duas soluções, irão ser apresentados apenas os resultados do node.js e do vert.x, pois são as plataformas escolhidas como candidatas a ser usadas neste projecto. Este teste usou 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.

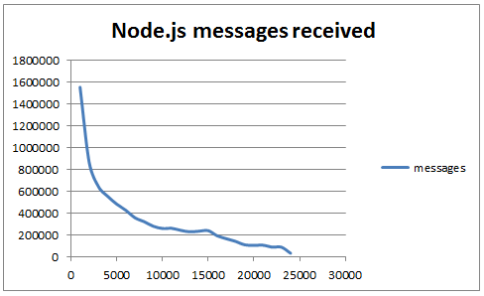


Figura 3.7: Mensagens recebidas [46].

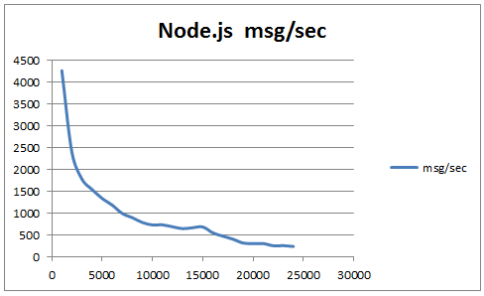


Figura 3.8: Mensagens recebidas por segundo [46].

Figura 3.9: Comportamento de node.js.

As imagens x e y, mostram os resultados que se obtiveram no Node.js conseguiu chegar às 24000 ligações, mas acabou por ser tornar lento e o cliente não conseguiu alocar mais memória para essas ligações.

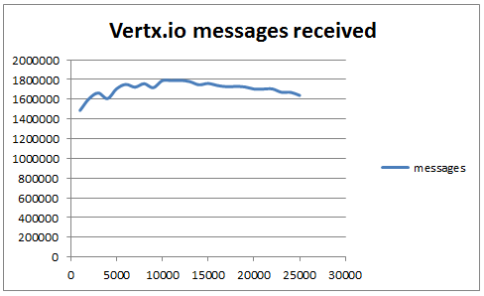


Figura 3.10: Mensagens recebidas [46].

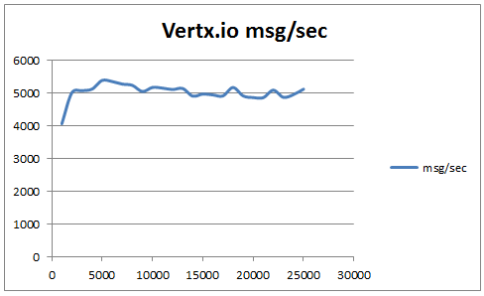


Figura 3.11: Mensagens recebidas por segundo [46].

Figura 3.12: Comportamento de vert.x.

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 [46]. No caso da plataforma Atmosphere não conseguiu lidar com mais de 3500 ligações, enquanto no netty a aplicação bloqueou nas 6500 ligações [46].

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
- Orientada a eventos

Este tipo de aplicações visa ter o melhor tipo de serviços possíveis e que sejam desenvolvidos da forma mais simples e mais 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 ter módulos para controlo de ligações a um WebServer, uma gestão de mensagens de base de dados ou gestão de autenticação. Cada um deles pode executar uma função diferente, isto implica uma melhor organização de código e da estrutura da arquitetura em si. Existe ainda a possibilidade de cada um destes módulos comunicarem sobre um eventbus, 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. Para além disso o vert.x permite ainda escolher entre várias linguagens de programação, suportando: 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.

3.5.2 Bibliotecas JavaScript WebRTC

Este tipo de bibliotecas têm por objetivo ajudar no desenvolvimento de aplicações que permitam um desenvolvimento a um alto nível. Foi feita uma análise comparativa entre duas bibliotecas, apresentadas na secção x, sobre os serviços que era possível fornecer e qual a escalabilidade de cada uma delas.

WebRTC Experiment vs SimpleWebRTC

Como já referido na secção x, 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, com estas bibliotecas é possível ter serviços como [34, 32, 33, 35]:

- 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);
- Possibilidade de banir utilizadores;
- Detecção de presença (orientado a eventos onde é verificado quando um utlizador entra ou utilizador sai, sem vários estados de presença);
- Possibilidade de 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 browsers. Este repositório permite uma comunicação entre browsers do mesmo tipo, como também browsers 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 é uma biblioteca que permite apenas fazer conferências áudio e vídeo com dois ou mais utilizadores e usa como base o módulo socket.io, node.js [47]. Em suma, WebRTC Experiment é o repositório mais completo, pois é mais escalável em termos de serviços, pois tem uma maior diversidade de serviços que podem ser fornecidos. Fornece uma API de desenvolvimento simples, criando objectos JavaScript que fornecem funções para tratar de negociações de media e permite uma API orientada a eventos. Usa como backend firebase para manter a informação sempre actualizada, apesar disso é possível a integração de WebSockets over socket.io e de node.js over socket.io.

Base de dados

Visto que seria necessário guardar a informação da aplicação num sistema de base de dados, optou-se por usar algo que já estivesse implementado. Visto que se escolheu vert.x para implementar uma solução WebSockets, optou-se por usar 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 ter muita escolha, visto que o WebRTC permite o desenvolvimento de aplicações web a partir da linguagem JavaScript. Em termos do servidor é que existe mais escolha, visto que foi escolhido para o servidor a plataforma vert.x. Como foi referenciado na secção x, a linguagem que mais se destacou mais nos testes realizados, foi Java. Desta forma optou-se por se usar Java para desenvolver a parte do servidor, visto que irá permitir uma maior performance 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 tiram proveito das funcionalidades das APIs referentes à tecnologia WebRTC, demonstrando como funcionam e de que forma podem ser utilizadas. Foi também uma abordagem a algumas bibliotecas que facilitam a implementação de uma solução, podendo desenvolver serviços a um alto nível de programação. Foram descritas algumas plataformas WebSockets que podem ser usadas tanto no servidor como no cliente. Foram descritos gateways SIP que podem ajudar a implementação de soluções com interoperabilidade entre clientes WebRTC e clientes SIP externos.

Capítulo 4

Arquitectura e Desenho

Depois de todo o estudo ter sido feito e das tecnologias analisadas e escolhidas, passou-se para a fase de implementação. Como já foi descrito no capítulo dos requisitos, definiram-se objetivos, casos de uso e requisitos, para que se pudessem implementar com sucesso. Neste capítulo vão ser apresentadas as especificações iniciais da arquitetura, depois serão apresentados os módulos que foram escolhidos e os que foram desenvolvidos para a execução de funções, de forma a descrever a finalidade de cada um. Por fim será apresentada a interação entre todo o sistema e de que forma foi estruturada a arquitetura.

4.1 Requisitos e Casos de Uso

O objetivo deste projeto era a análise da tecnologia WebRTC e o desenvolvimento de uma framework, que fosse testada numa aplicação. Este capítulo vai apresentar os requisitos e casos de uso que foram definidos inicialmente para esses testes. Para além disso em cada requisito vai ser descrito um resumo, que descreve cada um destes.

4.1.1 Chamada com controlo de estabelecimento de chamada

Este requisito/caso de uso é considerado como um requisito minimo obrigatório, visto que este projecto é referente à tecnologia WebRTC, que tem por objectivo 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. Isto é, necessário haver uma ligação inicial com

o servidor para que estes possam fazer a negociação da sessão, para estabelecer uma chamada. Depois da chamada estar estabelecida entre os clientes, ambos podem-se desligar 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 requisito anterior, isto é, 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 de 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 chamada deve ser criado simultaneamente um URL especifico. 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 com a outra. Video e Áudio Conferência Este requisito é uma extensão das anteriores, isto é, possível realizar uma conferência entre várias pessoas, quando todas as pessoas estiverem dentro dessa sessão se possam desligar do servidor e continuarem a falar. Também deve ser possível ter um controlo total desta sessão, ou seja, possível verificar a duração da chamada, como também quem se encontra nessa sessão. Para adicionar pessoas externas, deve funcionar da mesma forma que a chamada com pessoas externas, 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 em maior parte das aplicações de mensagens instantâneas e chamadas de voz e vídeo. É necessário garantir que uma pessoa tenha uma lista de contactos, para isto é necessário haver uma base de dados elaborada, onde fiquem guardadas as informações das pessoas que se encontram numa lista de contactos. Para além disso deve ser garantido um serviço de presença, ou seja, para cada pessoa numa lista de contactos deve ser possível saber os estados de presença em que uma pessoa se encontra (ex. Ocupado, Disponível, etc.). Sempre que o estado

de uma pessoa muda, deve garantir que todas as pessoas da sua lista de contactos, 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

Por norma um serviço de Chat ou Mensagens Instantâneas também é um serviço que se encontra em muitas aplicações deste tipo. Foi definido como um requisito para este projeto, sempre que é realizada uma chamada / conferência, este serviço deve estar presente. Isto é, sempre que alguém se encontre numa sessão de chamada, deve ter a possibilidade de enviar mensagens de texto para as pessoas que se encontrem nessa 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 RTC Data Channels, 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 e em desenvolvimento, é possível criar vários serviços com esta. Devido ao desenvolvimento dos browsers, estes já 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 projecto.

4.1.7 Gravação de Voz e Vídeo

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

- Ser possível a gravação total de uma chamada, isto deve ser feito tanto para chamadas de apenas 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. Desta forma 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 permite 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 Especificações Iniciais

Nesta fase de implementação foi necessário desenvolver uma arquitetura, para realizar uma framework organizada e funcional. Esta secção descreve toda arquitetura que foi especificada e as decisões que foram tomadas na realização da mesma. Como a tecnologia escolhida, para os clientes e o servidor comunicarem, foi vert.x decidiu-se que as funcionalidades deviam estar separadas por módulos. Assim seria possível ter uma estrutura bastante organizada e bastante escalável.

Figura 4.1: Especificação inicial dos módulos.

Na figura x, são apresentados, como exemplo, três módulos diferentes em que cada um executa uma ou mais funções diferentes. No caso em que framework ou uma aplicação tenha problemas na autenticação dos utilizadores, o programador sabe que o problema está no módulo 1. Assim é 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 cada um dos módulos poderia comunicar entre si, para isto é necessário criar um eventbus, como mostra a imagem seguinte.

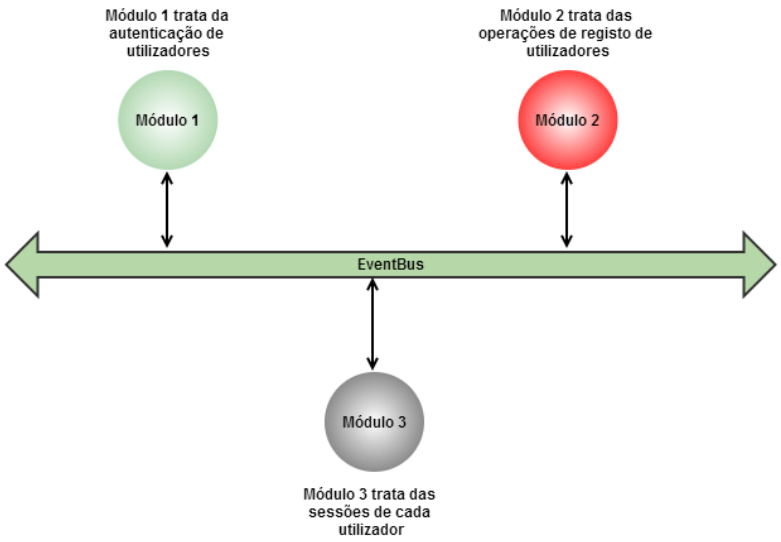


Figura 4.2: Especificação da interacção entre módulos.

Num cenário em que os utilizadores precisem de fazer autenticação e ao mesmo tempo seja criada uma sessão associada a essa autenticação. No exemplo da figura x, o módulo 1 trata de toda a parte de fazer a 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.

Outra especificação é que o cliente web não poderia 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.

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, 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.

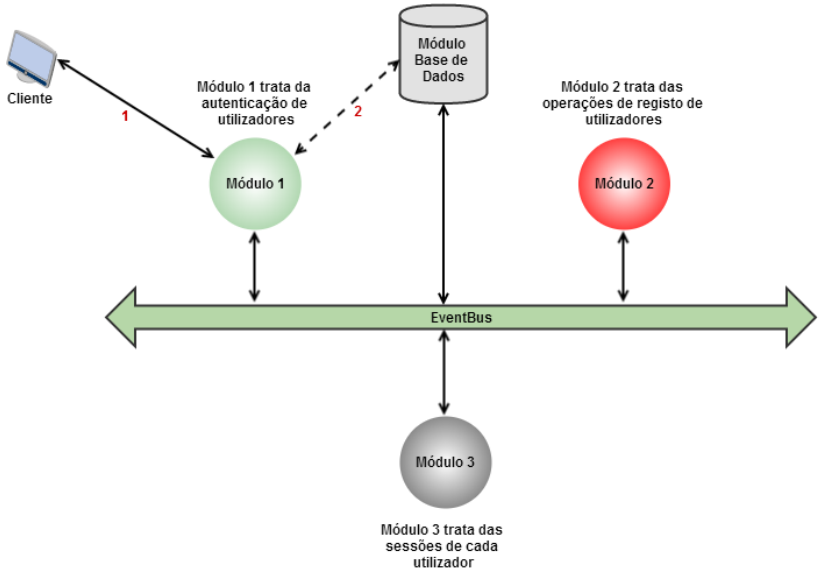


Figura 4.3: Especificação da interacção entre módulos.

4.3 Módulos

Depois de definir algumas especificações iniciais foi necessário começar a especificar a arquitetura tanto a nível de componentes, módulos e mensagens trocadas. Esta secção descreve os módulos escolhidos e a interação entre cada um deles.

4.3.1 Módulo Servidor Web

Vert.x é uma plataforma que nos consegue abstrair de serviços como apache, php, mysql entre outros. O módulo Web Server consegue criar um servidor web onde são facilmente fornecidos ficheiros que se encontrem nesse servidor. 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 encontra os ficheiros a que queremos aceder.
- index_page: a página indice 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 estar à escuta de pedidos e ligações.
- ssl: para ter suporte a https, de forma a criar sitios mais seguros e confiáveis.
- key_store_password: password do ficheiro Java Keystore, que guarda o certificado do servidor.
- bridge: permite dizer se o servidor actua ou não como um eventbus.
- inbound_permitted: só é usado se existir bridge e permite dar autorização de entrada a objectos json no eventbus.
- outbound_permitted: só é usado se existir bridge e permite dar autorização de saída a objectos 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. 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 funções:

- save: função do módulo que permite guardar documentos na base de dados, numa determinada collection.
- update: função do módulo que permite actualizar 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 nos queremos encontrar na base de dados.
- count: função que recebe como resposta o número de documentos de um determinado tipo, que se encontrem na base de dados.
- findone: função que encontra um único documento especifico na base de dados, ou seja, apenas é devolvido um único 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 por norma os módulos que se querem usar têm te ter uma configuração prévia, caso contrário irão ser usados os valores que estejam 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.

4.3.2 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 correctas. Este módulo 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-se encarregar de executar o logout.

- Authorised: Enviado 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. Isto é, na configuração inicial do módulo sao 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 enredeço ”test_myauthmod.login”, no caso do logout seria para o endereço “test_myauthmod.logout” e authorised “test_myauthmod.authorise”.

4.3.3 Módulo de gestão de sessões

Este módulo permite ao programador criar sessões e guardar informação nessa sessão. 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. É configurável como os módulos apresentados anteriormente, tendo para configurar os seguintes campos:

- address: endereço do módulo, para registar como handler no eventbus.
- timeout: o tempo que as sessões devem ficar guardadas.
- cleaner: endereço para quando uma sessão for destruida seja enviada informação.
- prefix: prefixo onde os clientes podem estar à escuta o 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 ficam guardadas.

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: serve para 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 destruida.
- clear: operação que destroi todas as sessões que estejam activas.
- heartbeat: envia heartbeats periódicos para o servidor de forma a reniciar o timeout, para que a sessão não seja destruida automaticamente.

- get: possibilidade de ir 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 activas
- 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.

4.3.4 Módulo de Registo Utilizador

Este módulo é encarregue pelo registo dos utilizadores no sistema, visto que a aplicação cliente não pode interagir directamente 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 indetificação do módulo no eventbus.
- port: porto para 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.

4.3.5 Módulo de Pesquisa e Intersecção de Serviços

Este módulo é usado em paralelo com o módulo de autenticação e módulo de gestão de sessões. É um módulo configurável como o de registo de utilizador onde se configura o endereço e o porto. Um utilizador ao fazer autenticação com sucesso irá verificar os serviços disponíveis no

terminal, que de seguida será enviado para o 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. Para exemplo simples, é possível imaginar dois conjuntos de serviços, conjunto A e conjunto B, em que:

- A = vídeo, audio, 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 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.

4.3.6 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: ao 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.

4.3.7 Módulo de Presença

Este módulo tem como objectivo 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", 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 (ex. Disponível para Ocupado), que 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.

4.3.8 Sinalização

Em termos de sinalização é usada a tecnologia socket.io, que é usado para aplicações que funcionem 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 RTCMultiConnection, optou-se por usar uma solução já implementada, adpatando-se ao servidor que está a correr localmente. Desta forma é necessária uma solução Javascript, que corre no servidor para que desta forma todos os clientes se possam ligar a esse processo e possam fazer a negociação de sessões.

4.4 Mensagens e Estrutura Arquitetural

4.4.1 Estrutura Arquitetural

A arquitetura do sistema consiste em num sistema geral de Web, isto é, 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 seguinte:

A parte do cliente tem por objetivo principal a comunicação com os módulos registados no eventbus, desta forma é possível garantir uma melhor interacção em tempo real entre cliente e o servidor, de forma a que toda a informação se encontre sempre actualizada. Contudo um cliente poderá fazer o registo de endereços/handlers no eventbus como qualquer módulo, para que possa

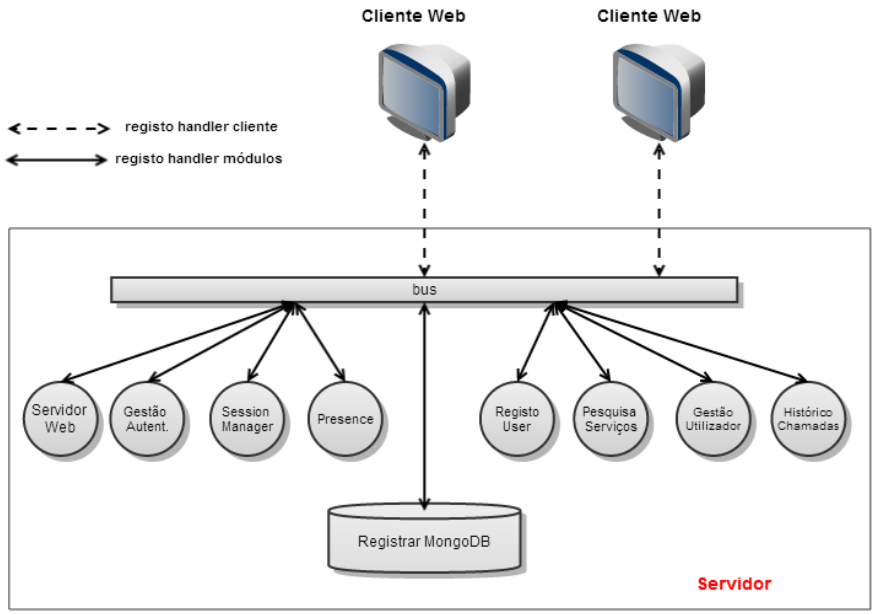


Figura 4.4: Arquitectura do Sistema.

existir interacção directa entre clientes, sem que seja necessário que a informação passe sempre por um módulo do servidor.

4.4.2 Mensagens

Para construir uma framework e aplicação funcionais é necessário que haja interacção 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.

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 seguinte figura.

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

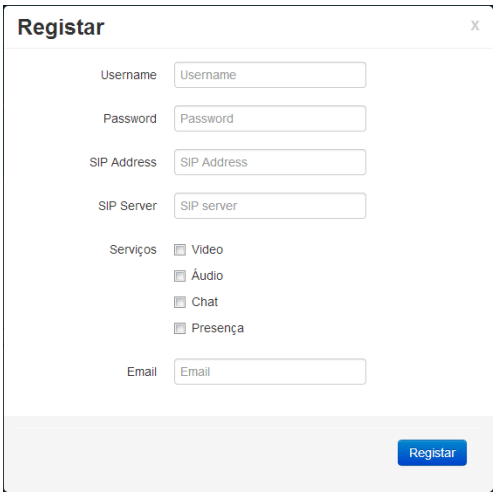


Figura 4.5: Formulário de registo no sistema.

porta, 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, necessita enviar uma mensagem do tipo "registuser" para "Módulo Registo de Utilizadores", como mostra a imagem x, que se encontra registado no eventbus.

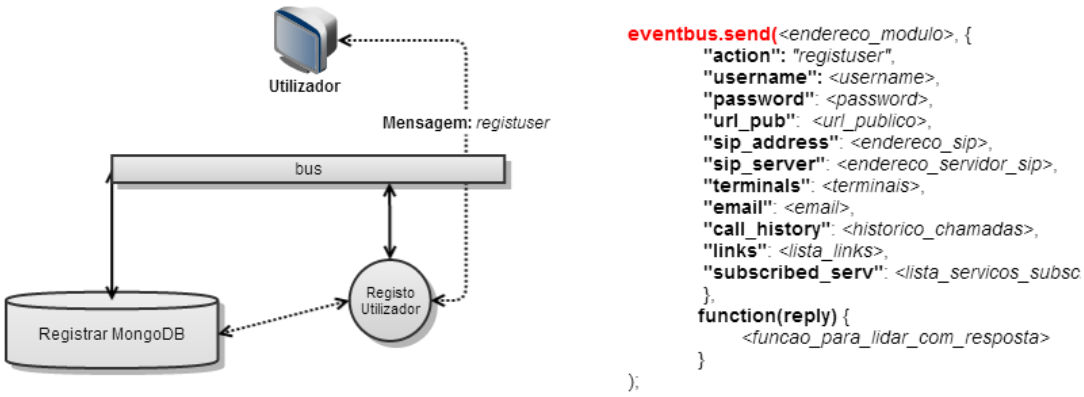
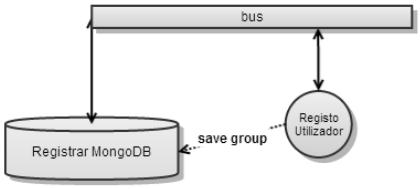


Figura 4.6: Mensagem de registo no sistema.

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 registado o 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 registar um grupo de perfil de utilizador, figura x.

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



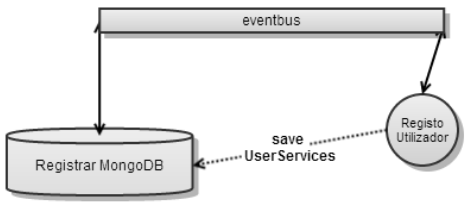

```
eventbus.send(<endereco_modulo>, {
      "action": "save",
      "collection": <collection>,
      "name": <nome>,
      "groups_presence": {
            "authorised_presence": <password>,
            "presence_authorization_presence": <url_publico>,
            "denied_presence": <endereco_sip>,
      }
      "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.

"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 realizada ainda outro tipo de operação, que passa pela criação de objetos na collection “UserServices”, a figura mostra o tipo de mensagem enviado para “Módulo de Base de Dados”.




```
eventbus.send(<endereco_modulo>, {
      "action": "save",
      "collection": <UserService>,
      "username": <nome>,
      "service": <service>,
      "configuration": {
            "automatic_authorisation_acceptance": <boolean>,
            "address": <url_publico>,
      }
      },
      function(reply) {
          <funcao_para_lidar_com_resposta>
      }
);
```

Figura 4.8: Gravar serviços subscritos por um utilizador.

Esta 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 que utilizador é e que serviço 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 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 que foi feito um pedido de subscrição onde o mesmo poderá ser aceite ou rejeitado.

Autenticação de Cliente

Para poder ter acesso aos serviços são 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”, figura x.

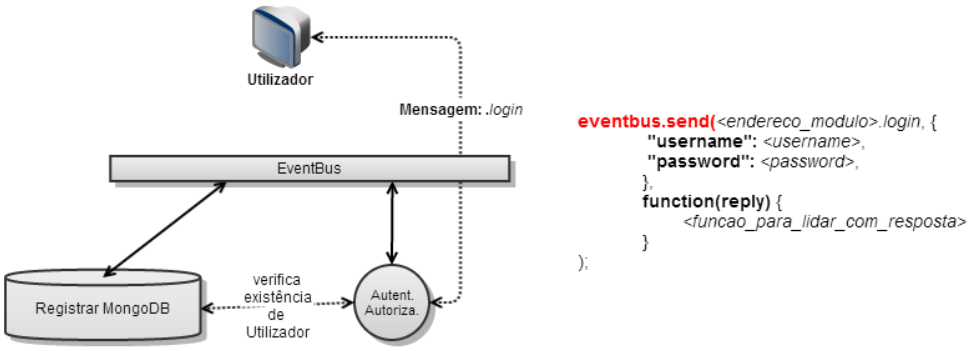


Figura 4.9: Autenticação de um utilizador no sistema.

A mensagem enviada para esse módulo, é a mensagem de login, onde deve ser identificado o username e a password. Quando o cliente enviar este tipo de mensagem 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”.

Esta mensagem serve para iniciar uma sessão para aquele utilizador, ou seja, sempre que um utilizador fizer autenticação no sistema vai enviar uma mensagem do tipo start para o “Módulo

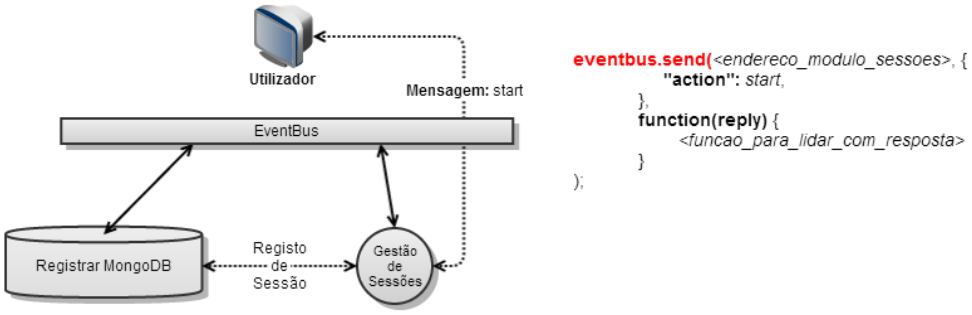


Figura 4.10: Iniciar sessão para um utilizador.

de Gestão de Sessões" para iniciar a sessão, como indicado na figura x. Depois de enviar essa mensagem irá receber outra resposta indicando que o registo da sessão foi executada com sucesso e ao receber essa mensagem terá de guardar informação referente à sessão.

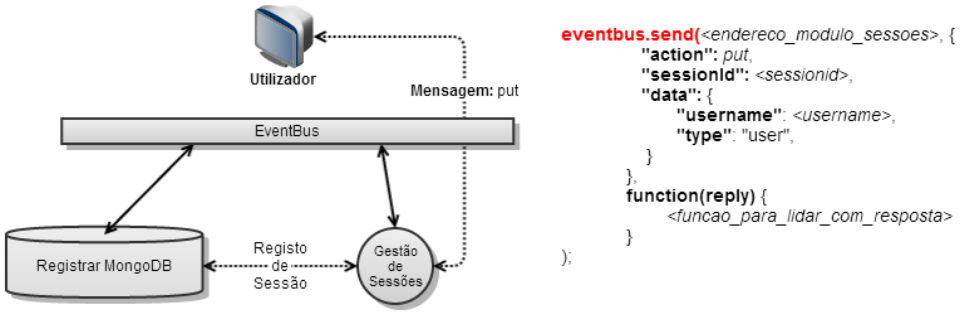


Figura 4.11: Guardar informação do utilizador na sessão associada.

Para guardar essa informação terá de enviar, mais uma vez, uma mensagem do tipo put para o "Módulo de Gestão de Sessões". 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 essa página é carregada são executadas algumas funções. Essas funções servem para 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 ou áudio ou ambos, assumindo-se à partida que o resto dos serviços são suportáveis. De seguida é criado um array que contém o que o terminal suporta e é enviado para o "Módulo de Pesquisa e Intersecção de Serviços".

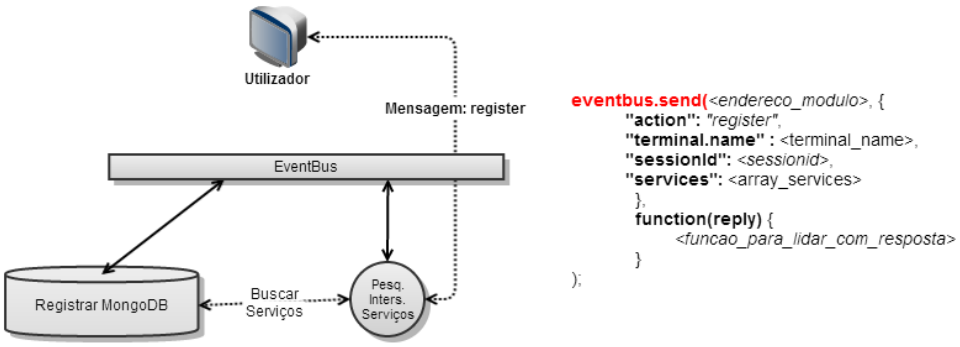


Figura 4.12: Mensagem de verificação de serviços de um utilizador.

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

```
[
 {
    "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 do serviço e a sua configuração, como também qual o uri do serviço, isto é, 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, 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.

Módulo de presença e Interacção no sistema

Depois de todo o processo de autenticação e o sistema verificar quais os serviços que o cliente suporta, um deles pode ser o serviço de presença. 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, na integração com a aplicação é criado um objecto JavaScript desse serviço "new PresenceService()". Para este serviço ficar completamente funcional é necessário carregar o módulo de presença para o servidor, para que possa haver 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 neste caso são:

- Available
- Busy
- Away
- Not Here
- Offline/Invisible

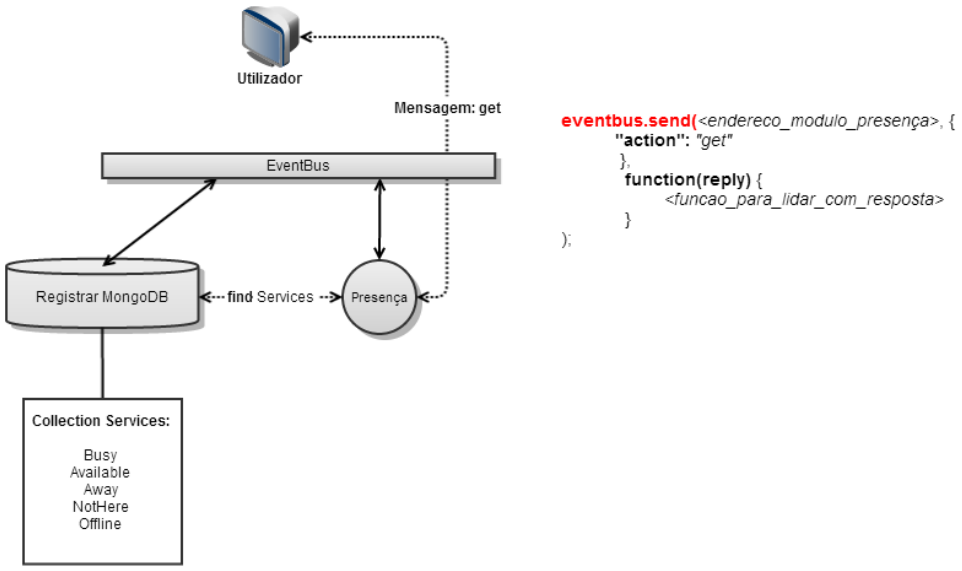


Figura 4.14: Mensagem para obter os estados de presença do sistema.

Para esta informação ser recolhida é enviada uma mensagem do tipo get par ao módulo de presença sobre o eventbus, como mostra a figura x.

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 x.

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 também irá actualizar o estado de presença na sessão respectiva 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 registar um handler referente a cada um dos utilizadores. Depois disso o utilizador sempre que fizer actualizar o seu estado irá ser enviado uma mensagem em publish, para o seu endereço de presença, como mostra a figura y.

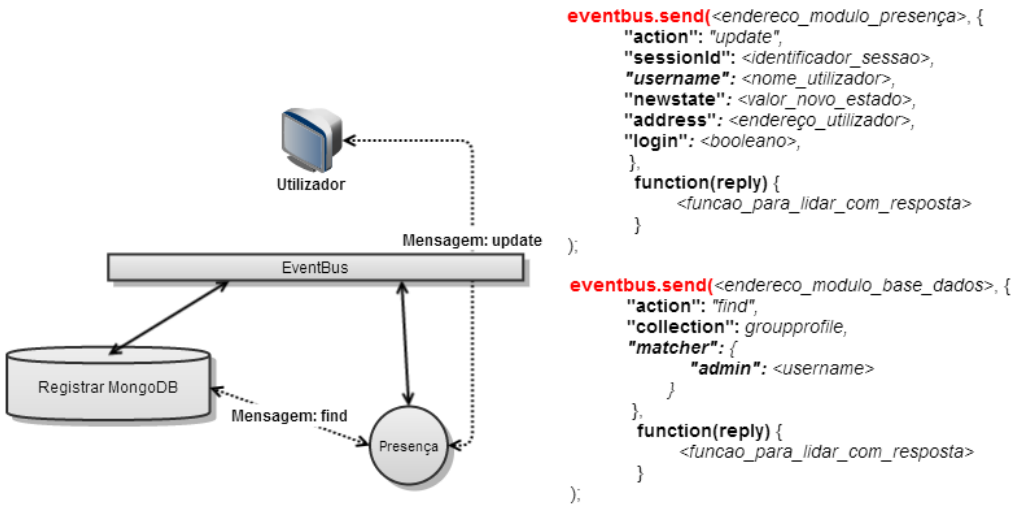


Figura 4.15: Mensagem para notificação e obter lista de contactos.

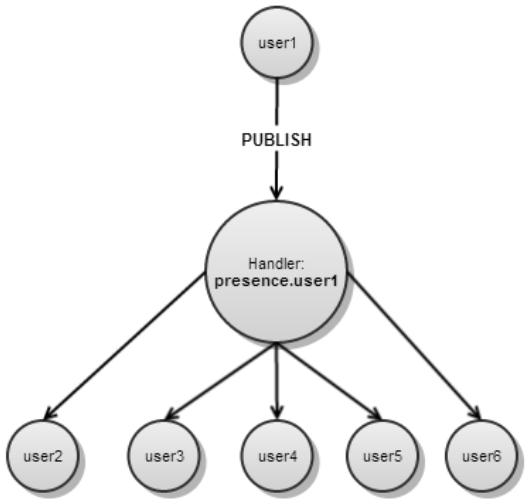


Figura 4.16: Notificação de mudança de estado.

Neste caso o campo login irá com o valor false, pois o utilizador já não se encontra fazer autenticação. Como foi dito na secção x, 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. 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 mostra figura x.

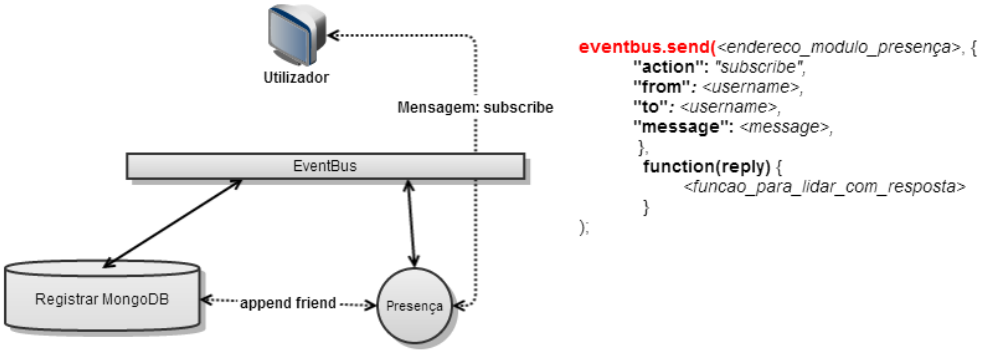


Figura 4.17: Mensagem para adicionar contacto.

O módulo de presença ao receber 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”. O módulo de presença irá 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 que foi adicionado uma nova pessoa à lista de contactos, para que possa haver uma actualização na parte do cliente.

Módulo de gestão de utilizadores

Este módulo é o que permite a pesquisa de pessoas no sistema. Desta forma é possível encontrar pessoas para adicionar à lista de contactos ou até mesmo adicionar pessoas a uma conferência que esteja a decorrer.

No cliente existe um campo de pesquisa, que permite que sempre que se escreva uma letra seja enviada uma mensagem para o módulo de gestão de utilizadores. É possível ainda definir quais sao os campos que se querem para a pesquisa, ficando isso definido “querycriteria”, enquanto que também é possível definir qual a informação que o servidor deve devolver em relação aos utilizadores que se querem procurar, ficando definido 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

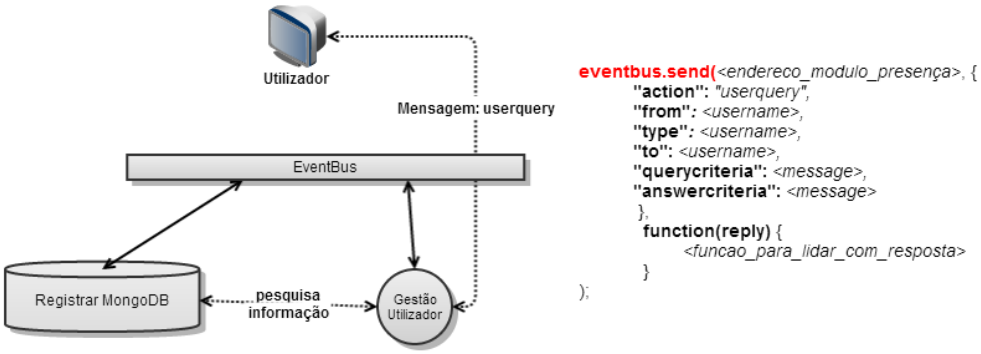


Figura 4.18: Mensagem para pesquisa de utilizadores do sistema.

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", também apenas está definido que deve ser devolvido o "username", na resposta.

Conferências áudio e vídeo

Como já foi referido na secção x, o repositório WebRTC-Experiment oferecia uma maior diversidade de serviços relacionados com WebRTC. Conforme as necessidades desta aplicação, 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 seleccione pelo menos uma pessoa, dessa forma irá aparecer uma janela para criar uma conferência, como mostra a figura x.

Neste caso seleccionou-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 seleccionadas. Será enviado uma mensagem "invite" para cada um dos contactos seleccionados, como mostra a figura x.

Os utilizadores que receberem esse convite irão receber um convite e serão notificadas com um popup, a identificar quem é que o está a tentar contactar, com possibilidade de aceitar ou rejeitar o mesmo, como mostra a figura x.

Este popup aparece devido à biblioteca de notificações criadas, que tem por objectivo lidar

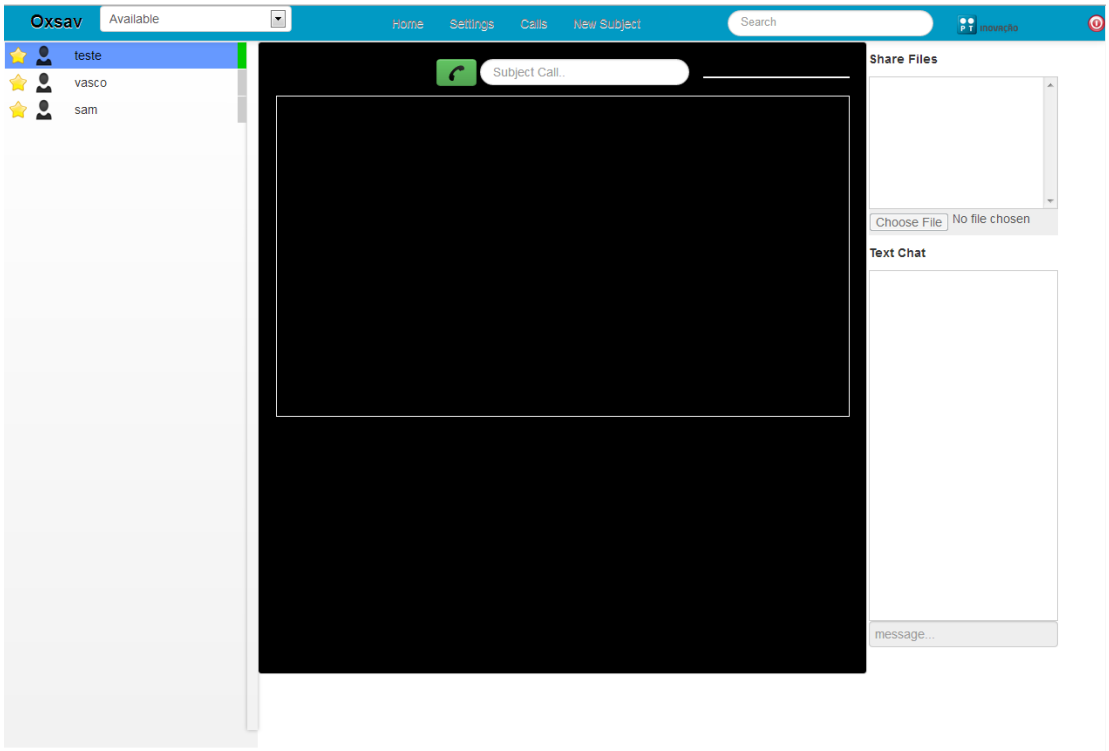


Figura 4.19: Convidar pessoas para uma sessão.

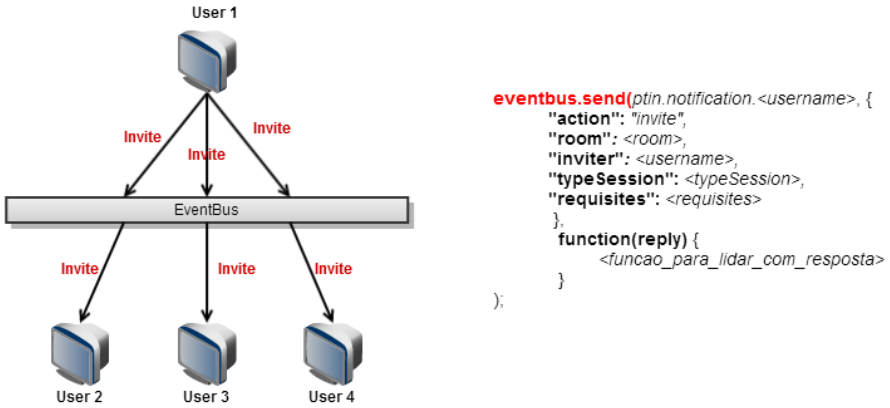


Figura 4.20: Mensagem de notificação para início de sessão.

com as várias notificações dos vários tipos como, convite para iniciar sessão, convite para adicionar à lista de contactos, etc. Para uma pessoa 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

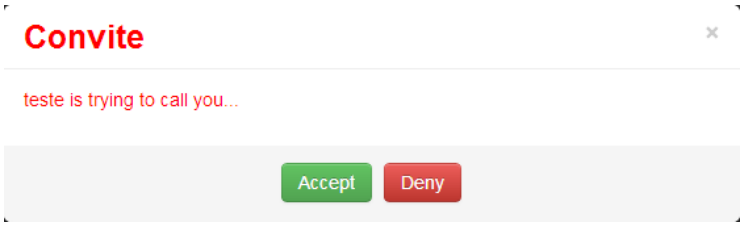


Figura 4.21: Notificação para início de sessão.

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 x.

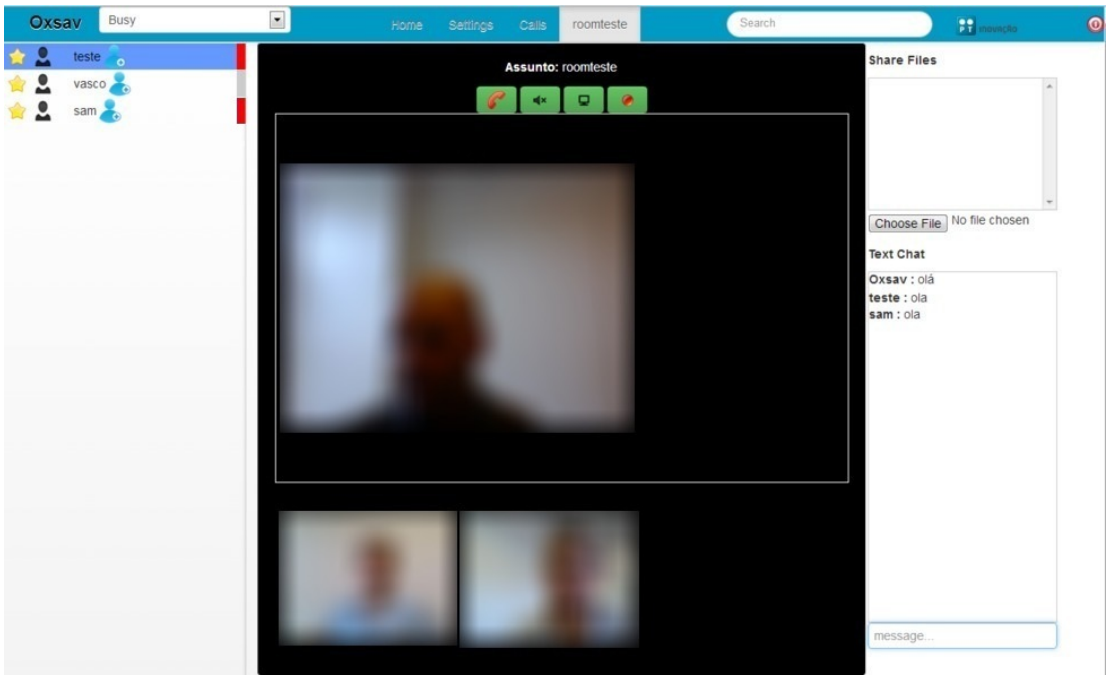


Figura 4.22: Conferência entre três pessoas.

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, basta fazer uma pesquisa rápida no sistema e convidá-la para se juntar. No caso de 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 mail ou por outro tipo de método para a pessoa em questão. Durante uma sessão de conferência neste sistema, estão activos os seguintes serviços:

- Chat
- Partilha de Ficheiros
- Conferência Áudio e Vídeo
- 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 socket.io. É uma tecnologia que permite a criação de aplicações que funcionem em tempo real e era uma das soluções implementadas com estas bibliotecas. 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 versão 4 de Universally Unique Identifier, é um algoritmo que garante que a probabilidade de criar identificadores iguais seja muito pequena. Desta forma é possível que duas ou mais sessões tenham o mesmo assunto, mas os seus identificadores vão ser diferentes [48].

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

Utilizador que inicia a conferência; Tempo que decorreu durante a conferência; Participantes dessa conferência; Hora inicial da conferência;

Para guardar esta informação para o módulo de histórico, de forma a que este consiga guardar essa informação no grupo associado ao utilizador. Esta informação é enviada numa mensagem do tipo “registcall”, como mostra a figura x.

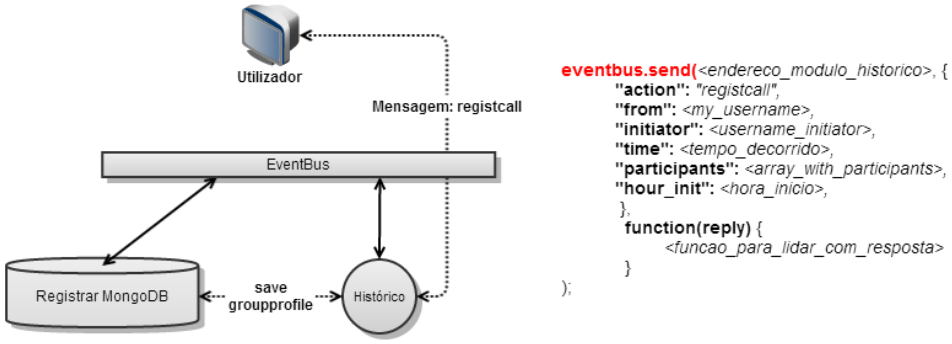


Figura 4.23: Registo de chamada no histórico de chamadas.

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

4.5 Sumário

Neste capítulo foi descrita toda a parte funcional da framework e aplicação. Começou-se por se descreverem as decisões que foram tomadas inicialmente. De seguida passou-se a descrição de cada módulo usado e desenvolvido para esta solução, onde são descritas as configurações que cada um pode ter. 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

Testes e Resultados

Depois de toda a implementação é necessário haver uma validação e uma fase de testes. Nos testes realizados foram analisados essencialmente performance do CPU, a memória que era usada em cada sessão, tanto ao nível do servidor como ao nível do cliente. Para estes testes foram definidos dois cenários diferentes, principalmente para analisar o comportamento da rede.

5.1 Cenário 1

Neste primeiro cenário os clientes ligavam-se por VPN à rede empresarial, de forma a que se pudessem ligar ao servidor que se encontrava nessa mesma rede. Num primeiro teste apenas dois clientes iriam realizar uma conferência áudio, video e dados entre eles, como mostra a figura 5.1.

Estes clientes encontravam-se numa rede externa à rede empresarial e como o servidor se encontra na rede empresarial, é necessário que haja uma ligação VPN para essa rede. Desta forma os clientes teriam acesso aos serviços fornecidos pelo servidor e conseguem realizar uma chamada entre eles. As especificações de cada máquina podem ser encontradas nos Anexos.

Neste exemplo o PC1 enviou um convite de chamada para o PC2, e verificaram-se os seguintes resultados no PC2, figura 5.2.

O ponto um 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 MB, estabilizando depois de toda a negociação ser feita entre os peers. No momento em que a

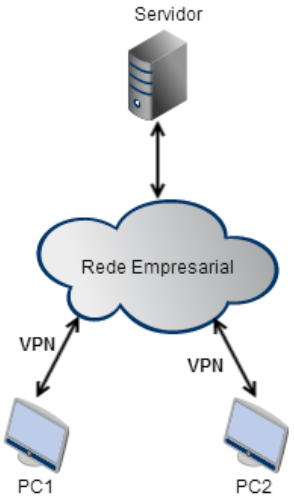


Figura 5.1: Cenário 1 e teste 1.

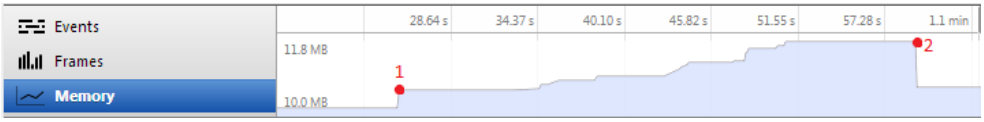


Figura 5.2: Gráfico de utilização de memória numa chamada.

chamada é desligada nota-se um decréscimo repentino na utilização de memória, fazendo com que desça mais de 1 MB de utilização de memória.

Em termos de vídeo, áudio e dados não se notaram grandes atrasos nem perdas de pacotes, todas as mensagens de chat foram trocadas, todos os ficheiros foram enviados e recebidos. No que diz respeito a vídeo e áudio não se notaram atrasos nem quebras de vídeo e de áudio.

Num mesmo cenário foi executado outro teste, onde neste caso em vez de serem apenas dois clientes, fez-se uma conferência com as mesmas caracteristicas, mas entre 4 pessoas, figura 5.3.

Neste cenário foram retirados apenas resultados de um dos computadores, visto que as caracteristicas de cada um são bastante semelhantes. Neste caso foram retirados resultados tanto ao nível de CPU e de memória.

No que diz respeito ao CPU foram retirados resultados tanto no inicio como no final da conferencia, figuras 5.4 e 5.5.

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

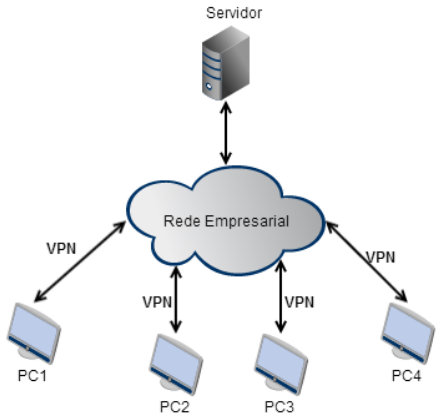


Figura 5.3: Cenário 1 e Teste 2.

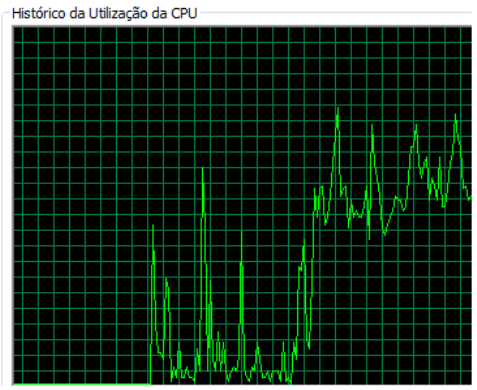


Figura 5.4: Utilização de CPU no inicio da conferência.

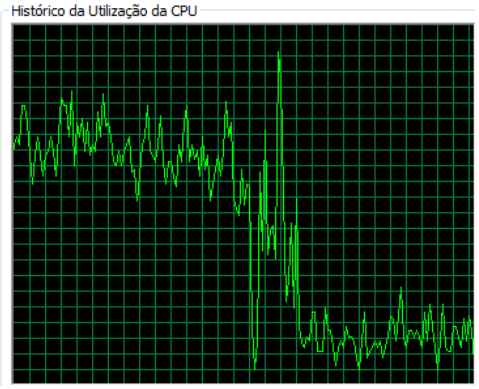


Figura 5.5: Utilização de CPU ao finalizar conferência.

Figura 5.6: Comportamento da utilização de CPU.

consideravelmente - cerca de 40% - a percentagem de uso do CPU.

No que diz respeito à utilização de memória do computador os resultados foram os seguintes, figura 5.7.

Figura 5.7: Utilização de memória ao iniciar a conferência.

Neste exemplo verifica-se um aumento de memória sempre que alguem entra na conferência como se pode verificar no ponto 1, 2 e 3, da figura x. Ou seja sempre que alguem entra na

Tabela 5.1: 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

conferência existe um aumento de utilização de memória. Neste exemplo a utilização de memória estabeleceu nos 9.2MB, ou seja, houve um aumento de 4.2MB de utilização de memória durante a conferência entre 4 pessoas. A partir do momento que se desligou a conferência entre todas as pessoas, houve um decrescimo significativo e linear, sendo isso visivel na figura 5.8.

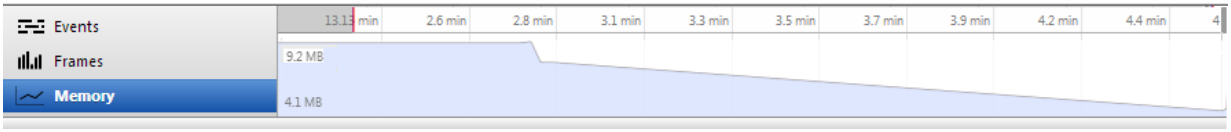


Figura 5.8: Utilização de memoria ao finalizar conferência.

Em termos de 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 correctamente. Começou-se a notar uma maior falha em termos de pacotes entregues e trocados entre os diferentes peers.

No que diz respeito ao servidor conseguiram-se tirar algumas conclusões fazendo 2 testes distintos. Num teste tinhamos vários clientes a lançar eventos para o servidor de forma a que o servidor os tivesse de processar para responder. O outro teste foi apenas para verificar se realmente o servidor é usado ou requisitado, enquanto os clientes se encontram numa conferência. Como era um computador com sistema operativo linux (CentOS), foi usado o comando sar, que permite medir várias unidades do processador, 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 5.1.

Tabela 5.2: 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

Foi um teste realizado apenas para 2 minutos, onde se verificou uma utilização ao nível do utilizador de 1.71% no primeiro minuto e ao nível do sistema uma utilização de 0.22%. Enquanto 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 os resultados que foram retirados foram os da tabela 5.2.

Foram resultados tirados durante 4 minutos, verificou-se que a utilização de CPU era minima, 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 Cenário 2

Este cenário 2 assemelha-se ao cenário apresentado na secção anterior. O servidor neste caso é um computador normal que se encontra na rede, a rede não é uma rede empresarial mas sim uma rede doméstica, que é mais controlada. Neste cenário são realizados dois tipos de testes diferentes, com o objectivo de verificar como se comporta um computador com um poder de processamento (CPU) e de que forma se comportam vídeo e áudio numa rede com menos

Num primeiro teste foram usados apenas dois computadores, figura x, onde foi realizada uma chamada simples entre cada um deles.

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 verificou-se o seguinte, figura 5.10.

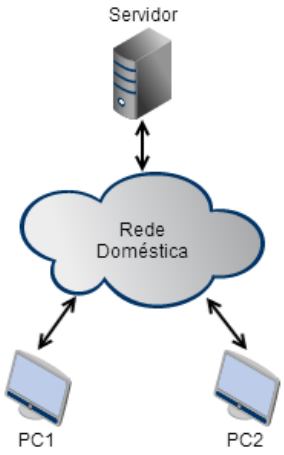


Figura 5.9: Cenário 2 e teste 1.

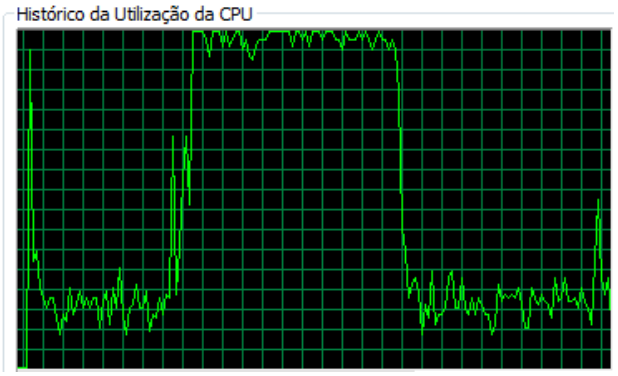


Figura 5.10: Resultados de CPU no computador com menor poder computacional.

Consegue-se verificar que a partir do momento que a conferência iniciou a utilização do CPU subiu consideravelmente atingindo os 100% algumas vezes. A partir do momento que a conferência termina existem uma descida bastante considerável por parte 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, verificou-se que a percentagem de utilização não era tão alta mas que também subiu consideravelmente, figura **??**.

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

Visto que os testes realizados anteriormente já davam para tirar algumas conclusões relacionadas com a memória e CPU, decidiu-se verificar como é que vídeo, áudio, entrega de ficheiros e chat se comportavam numa rede mais estável. Usaram-se computadores com car-

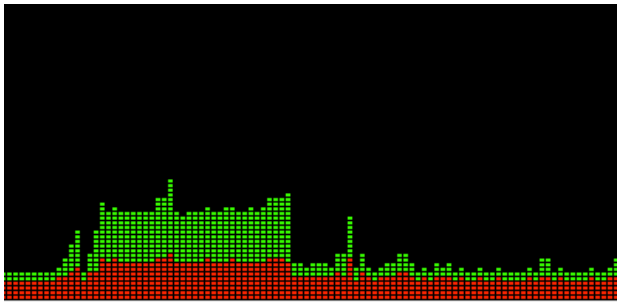


Figura 5.11: Resultados de CPU num MacBook.

acteristicas bastante semelhantes para este teste, com um cenário igual ao da figura 5.12.

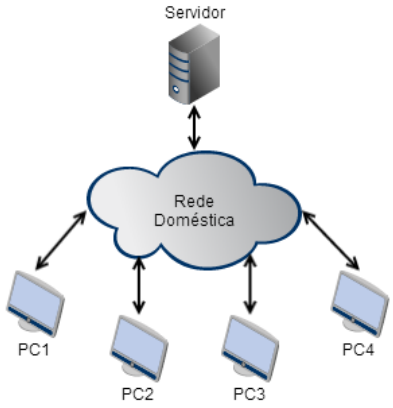


Figura 5.12: Cenário 2 e teste 2.

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 e em relação ao chat as mensagens eram todas recebidas e enviadas com sucesso.

Capítulo 6

Conclusions

Neste capítulo é efetuada uma síntese do trabalho desenvolvido e apresentado ao longo desta dissertação. São apresentadas algumas conclusões e análise critica sobre o trabalho que foi realizado e de que forma poderá contribuir para o futuro. Neste capítulo são apresentadas as ideias para o trabalho futuro relacionado com este projecto.

6.1 Resumo de Trabalho Desenvolvido

As comunicações em tempo real cada vez são um requisito para o dia a dia das pessoas. Vêm-se cada vez mais pessoas a recorrerem à Internet para realizarem chamadas, ou enviarem mensagens de texto para outras pessoas, sem que nisso haja um custo adicional. Este projecto foi desenvolvido nesse âmbito, visto que a tecnologia WebRTC, permite a que os utilizadores consigam comunicar com outras pessoas a partir de um browser. Este é um projecto mais tecnológico do que cientifico, visto que o pretendido é desenvolver uma Framework e um cliente que permita utilizar os serviços base do WebRTC e serviços de comunicação (presença, lista de contactos, chamadas vídeo e áudio, etc.). Não foram definidos objectivos em termos de resultados e métricas que tivessem de ser testadas, para chegar a uma conclusão em relação a esta tecnologia. Antes de todo este processo tecnológico teve de haver uma fase de estudo e de análise comparativa entre diferentes tecnologias. Recorreram-se a estudos realizados anteriormente por outras pessoas, como também houve uma fase de testes inicial entre as diferentes tecnologias. WebRTC foi a tecnologia que mais se estudou pois era o foco principal deste projecto. Foi necessário entender toda a lógica associada a ela, de como a sinalização era feita, como é que os utilizadores pode-

riam criar ligações peer, etc. Para além disso sendo esta uma tecnologia recente era necessário estar atento às novas actualizações, exemplo disso eram o RTCDataChannels, que no inicio do desenvolvimento deste projecto não estavam funcionais em qualquer browser. Com o decorrer do tempo foi possivel implementar uma solução com RTC Data Channels que permiti-se aos utilizadores enviar mensagens de texto e partilhar ficheiros, tirando partido dessa API. Desta forma foi possível ter uma solução totalmente funcional com WebRTC em que não é necessário recorrer ao servidor para ter alguns destes serviços. Como dito no decorrer da dissertação foram definidos alguns requisitos e casos de uso para a framework e aplicação, os quais foram cumpridos com sucesso. Existe neste momento uma solução, onde todos os serviços que foram definidos inicialmente como requisitos / casos de uso, se encontram funcionais. É necessário citar 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 projecto. As principais dificuldades existiram no desenvolvimento de alguns serviços, visto que todas as tecnologias que foram usadas ouve algumneste projecto são tecnologias relativamente novas e que não eram conhecidas, nem nunca tinham sido usadas pelo estudante. Ha dificuldade inicial pois o projecto ainda não estava bem definido inicialmente e ainda nem se sabia o que realmente ia ser desenvolvido e em que contexto. Foi necessário um tempo inicial para definir o que realmente ia ser feito e de que forma poderiam ser usadas todas as tecnologias analisadas. Apesar de ser um projecto mais tecnológico decidiram-se fazer alguns testes que pareceram relevantes. Os testes realizados foram principalmente em termos de CPU e de memória das diferentes máquinas usadas. Como referido na secção dos testes, usaram-se dois cenários idênticos, mas com diferentes caracteristicas. 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 consideravel a partir do momento que é iniciada uma conferência, pois cada computador tem de aceder aos componentes internos (microfone e webcam), como também para cada cliente numa conferência tem de criar um novo objecto 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%, com isto indica que para usar a tecnologia WebRTC, é necessário ter um computador que tenha uma boa capacidade de processamento. No último teste foi apenas para se verificar se o problema da conferência entre quatro pessoas permanecia, isto é, se realmente as quebras de vídeo, o áudio atrasado, o

chat e a partilha de ficheiros funcionavam bem. Utilizando uma rede mais estável verificaram-se algumas melhoras, 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 objectivos cumpridos nesta fase inicial, foi a de haver um estudo mais profundo da tecnologia na empresa e permitir o acesso a essa informação às pessoas da mesma. Para além disso este projecto permitirá num futuro próximo a implementação de muitos mais serviços de telecomunicações, de uma forma mais facilitada e ao mais alto nível. Com a evolução desta tecnologia pretende-se realizar uma solução bem mais estável e que seja da confiança das várias entidades que a usem. Visto que é uma tecnologia relativamente recente é necessário tirar proveito disso e realizar uma solução num âmbito diferente, com o objectivo de ajudar entidades externas e até mesmo internas.

6.3 Trabalho Futuro

Como trabalho futuro a framework terá de ser mais completa e melhorada, isto é, será necessário estar sempre em actualização modificar e verificar eventuais erros existentes. Será necessário ainda implementar todos os serviços com o vertx, 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 inter-operabilidade entre browsers, isto é, para que utilizadores que usem browsers diferentes possam comunicar entre eles. Existe ainda a possibilidade de implementar serviços SIP, para que possa haver uma interoperabilidade entre clientes WebRTC e clientes SIP.

Apêndice A

Name of the Appendix

Bibliografia

[1] G. Inc., "Webrtc." http://www.webrtc.org/, 2011-2012. [Online; acedido Fevereiro-Setembro 2013].

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

[3] D. A. 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] G. Inc., "Getting started with webrtc." http://www.html5rocks.com/en/tutorials/webrtc/basics/, 2011-2012. [Online; acedido Fevereiro-Setembro 2013].

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

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

[7] W3schools, "Browser statistics and trends." http://www.w3schools.com/browsers/browsers_stats.asp, 2011-2012. [Online; acedido Fevereiro-Setembro 2013].

[8] 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].

[9] 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].

[10] 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].

[11] C. Bran, Plantronics, C. Jennings, Cisco, J. 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].

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

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

[14] 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].

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

[16] 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].

[17] 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].

[18] D. McGrew, Cisco, E. Rescorla, and I. 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].

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

[20] I. B. Castillo, J. M. Villegas, Versatica, V. Pascual, and A. 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].

[21] Y. Suzuki, N. Ogashiwa, M. K. G. College, and D. 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].

[22] J. Uberti, G. 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].

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

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

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

[26] I. Instant IO, "peercdn." https://peercdn.com/, 2013. [Online; acedido Fevereiro-Setembro 2013].

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

[28] M. D. Network, "Bananabread." https://developer.mozilla.org/pt-PT/demos/detail/bananabread, 2012. [Online; acedido Fevereiro-Setembro 2013].

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

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

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

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

[33] 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].

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

[35] 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].

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

[37] J. Inc., "Nodejs." http://nodejs.org/, 2012. [Online; acedido Fevereiro-Setembro 2013].

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

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

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

[41] G. 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].

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

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

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

[45] 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].

[46] S. 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].

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

[48] P. Leach, Microsoft, M. Mealling, L. Refactored Networks, R. Salz, and I. 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].