Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Roberto Oliveira Rocha

WebRTC - Evolução na Web

Outubro de 2014

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Roberto Oliveira Rocha

WebRTC - Evolução na Web

Tese de Mestrado

Sistemas e Tecnologias de Informação para as Organizações

Professor Steven Abrantes

Professor Rui Almeida

Outubro de 2014

RESUMO

A forte e exponencial evolução da web faz com que esta seja cada vez mais o suporte para novas aplicações devido à sua extensibilidade, simplicidade, compatibilidade e facilidade de acesso. É então natural perceber que, ao mesmo tempo que se concentram esforços para criar novas aplicações em plataformas web, existe também o esforço de normalizar, simplificar e providenciar a web de novas e melhores ferramentas de desenvolvimento, mais capazes e mais simples. Vimos isto com a evolução do HTML, do JavaScript e até mesmo do CSS. Estas tecnologias e a sua evolução têm um impacto forte na forma como as pessoas utilizam a web. Basta lembrar que não assim a tantos anos o que existiam eram páginas simples, sem estilo, sem conteúdo dinâmico e com capacidades limitadas. Nos dias de hoje já assistimos à existência de páginas HTML5 com conteúdos dinâmicos, conteúdos multimédia, gráficos compostos, efeitos visuais, etc. Todas estas capacidades foram introduzidas com novas tecnologias web que foram surgindo e foram sendo utilizadas pela sociedade, acabando por se formarem standards pelos organismos competentes.

Porém, é previsível que esta evolução não se fique por aí, e constantemente vão surgindo novas tecnologias web, com novas capacidades. O WebRTC é uma destas tecnologias web que ainda se encontra em fase de desenvolvimento, estando neste momento a IETF (Internet Engeneering Task Force) e o consórcio W3C (World Wide Web Consorcium) a concentrar esforços para a normalizar, após o seu desenvolvimento ter sido iniciado pelo Google. O objetivo desta tecnologia é permitir a comunicação direta e em tempo real entre navegadores, sem necessidade de transmitir os dados através de um servidor intermédio e sem a necessidade de extensões. Com esta tecnologia é então possível que qualquer dispositivo equipado com navegador seja capaz de comunicar de forma direta com outro para transmissão de dados, seja transmissão de áudio, vídeo ou simples ficheiros.

A popularidade do WebRTC tem crescido exponencialmente nos últimos três anos, prova disso é o crescente número de soluções implementadas assim como a maior compatibilidade dos navegadores para com esta tecnologia, ainda que esta não se encontre totalmente normalizada.

Com esta dissertação pretende-se que seja apresentada de forma clara a tecnologia, que seja explicado o seu funcionamento, capacidades e funcionalidades. Adicionalmente à tecnologia WebRTC, são obrigatoriamente apresentadas as tecnologias adjacentes que permitem o seu funcionamento. Pretende também que se perceba o impacto que esta tecnologia pode ter no futuro do desenvolvimento de aplicações Web e na forma em como comunicamos.

Em conjunto com o estudo, é também apresentada uma aplicação desenvolvida como prova de conceito onde são apresentadas e demostradas algumas das capacidades da tecnologia. Com o desenvolvimento desta aplicação pretende-se que fique mais explicito o funcionamento da tecnologia. A aplicação apresentada servirá como protótipo para outras a desenvolver a nível empresarial.

ABSTRACT

The fast and strong evolution of the Web makes it more and more the target for the development of applications, due to its extensibility, simplicity, compatibility and ease to access. It's only natural then to understand that, while there is a constant effort to develop new applications in the Web, there is also an effort to standardize, simplify and give new and better development tools to the Web, more capable and simpler. This has been seen with the evolution of HTML, JavaScript and even CSS. These technologies and their evolution have an enormous impact in the way people use the Web. To better understand this, just think that, not that many years ago the Web consisted of simple pages, with no style, no dynamic content and with very limited capabilities. Nowadays we can see HTML5 pages that have dynamic content, media content, canvas, graphs, visual effects, 3D modulation, etc. All these capabilities where introduced with new Web technologies that came along, became widely used and end up becoming standards defined by competent organizations.

However, it's predictable that this evolution doesn't stop there, so new web technologies are constantly appearing, with new capabilities. WebRTC is one of those technologies that it's still in the development phase, being that at this moment IETF and W3C are gathering efforts in order to create a standard, after its development was started by Google. The objective behind this technology is to allow real time direct communication between peers, without the need to send data over a middleware server and without the need of plugins. With this technology it's possible for any device, providing it has a WebRTC capable browser, to be capable of communicating in a direct way with another peer in order to communicate data, whether it is audio, video or even files.

WebRTC's popularity has grown exponentially in the last three years, and to prove it there is and ever growing number of solutions implemented and greater browser compatibility to this technology, considering that it is not a standard yet.

With this essay it's intended to present in a clear way the WebRTC Technology, to explain the way it works, its capabilities and functionalities. Additionally to the WebRTC Technology, some other technologies also needed to be introduced because of their relation with WebRTC and because they make WebRTC possible. It's also intended that it's understood the impact that this technology may have in future Web applications development and in the way we communicate.

Additionally to the essay, it's presented a Web application developed as proof of concept where some capabilities are presented and where is intended that the way WebRTC works becomes clearer. In a near future the application will be use as a prototype to develop WebRTC in an enterprise environment.

PALAVRAS CHAVE

WebRTC
Comunicação em tempo Real
Aplicações Web
HTML5
JavaScript
PeerConnection
DataChannel
ICE
STUN
TURN

KEY WORDS

WebRTC
Real Time Communication
Web Applications
HTML5
JavaScript
PeerConnection
DataChannel
ICE
STUN
TURN

AGRADECIMENTOS

Gostaria de deixar um agradecimento especial em primeiro lugar a toda a minha família, na qual incluo a minha namorada Joana Coelho, pelo apoio, colaboração e ajuda fornecida não só durante a execução desta dissertação mas ao longo de todo o mestrado e todo o meu percurso académico.

Agradeço também o apoio das empresas, Martifer e Pessoas&Processos, nas quais trabalhei durante o mestrado, pela ajuda e disponibilidade demostrada sempre que foi necessário. Em especial gostaria de deixar um grande agradecimento aos meus colegas em ambas empresas, João Sousa, Tiago Rebelo e Tiago Silva, por todo o apoio e pela ajuda durante a execução dos testes da aplicação.

Por último, mas com uma enorme importância, um agradecimento a todos os meus professores e assistentes na Licenciatura em Engenharia Informática e neste mestrado, com especial aos meus orientadores Rui Almeida e Steven Abrantes, que mostraram um apoio incondicional, e que participaram num grande esforço no desenvolvimento deste trabalho.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

ABREVIATURAS E SIGLAS

AJAX - Asynchronous JavaScript and XML

API - Application Programming Interface (Interface de Programação de Aplicações)

CSS - Cascading Style Sheet

DTLS - Datagram Transport Layer Security

DTMF - Dual-Tone Multi Frequency (Combinação de tons como sinais, usado nos telefones tradicionais)

G.711 - codec de audio

G.722 - codec de audio

HTML - HyperText Markup Language

HTML5 - Versão 5 do HTML

HTTP - HyperText Transfer Protocol

ICE - Interactive Connectivity Establishment

ICE Candidate (Candidato ICE) - Candidatos que permitem a ligação ao parceiro WebRTC

iLBC - codec de audio

IP - Internet Protocol

iSAC - codec de audio

Jquery – Biblioteca JavaScript

JSEP - Javascript Session Establishment Protocol

JSON - JavaScript Object Notation

NAT - Network Address Translator

Opus - codec de audio

ORTC - Object Real-time Communications

P2P - peer-to-peer (ponto-a-ponto)

RTCP - Real-Time Transport Control Protocol

RTP - Real-Time Transport Protocol

SCTP - Stream Control Transmission Protocol

SDP - Session Description Protocol

SIP - Session Initiation Protocol

SRTP - Secure Real-time Transport Protocol

SSL - Secure Sockets Layer

STUN - Session Traversal Utilities for NAT

tags - elementos identificadores no HTML

TCP - Transmission Control Protocol

TLS - Transport Layer Security

TURN - Traversal Using Relays around NAT

UDP - User Datagram Protocol
URL - Uniform Resource Locator (de forma simples, link)
VoIP - Voice over IP
VP8 - codec de vídeo
WebRTC - Web Real-Time Communication
XHR - XMLHttpRequest
XMPP - Extensible Messaging and Presence Protocol

Introdução

Há mais de cem anos já a humanidade sonhava com uma rede global de conhecimento. Em 1910 já existia uma instituição, de seu nome Mundaneum[1], cujo objetivo era agregar todo o conhecimento existente no mundo e classifica-lo a partir de um sistema denominado de "Classificação Decimal Universal". Este sonho foi evoluindo, e aliado a ele foram surgindo novos conceitos de classificação e organização de conhecimento, foram aparecendo novas formas de apresentação de informação como o Memex (1945)[2]. O Memex era um sistema que ambicionava armazenar e comprimir informação e conhecimento de uma forma mecanizada que permitisse flexibilidade e consulta rápida da informação. Considera-se que este sistema influenciou o desenvolvimento dos primeiros sistemas de *Hypertext*, termo que surge cerca de 1965 com Ted Nelson[3].

Estes e muitos outros estudos levaram ao aparecimento, em 1969, da primeira rede de comunicação de pacotes de dados, a ARPANET[4], uma rede desenhada para investigação, educação e para organizações governamentais que com o tempo tornou-se naquilo que hoje conhecemos como Internet. Após o aparecimento da Internet, foram surgindo inúmeras tecnologias como GML (Generalized Markup Language), protocolos de transmissão na internet, novas linguagens, novas formas de apresentação e navegadores[5].

Para este estudo é importante realçar o aparecimento de tecnologias como HTTP (HyperText Transfer Protocol), surgido em 1990[6]; HTML cuja primeira especificação data de 1993[7]; a linguagem JavaScript apresentada em 1995[8]; a evolução do HTML para HTML5 cuja primeira especificação data de 2008[7, p. 5]; navegador Chrome (da Google), lançado a 2 de Setembro de 2008[9]. Todas estas tecnologias assim como algumas outras que irão ser exploradas ao longo da dissertação, permitem que em Maio de 2009 surja o WebRTC[10].

A tecnologia WebRTC surge no seguimento de um estudo feito pela equipa responsável pelo navegador Chrome. Este estudo concentrava-se em tentar encontrar funcionalidades presentes em aplicações Desktop que não estavam disponíveis em aplicações Web. A equipa concluiu que a maioria das diferenças encontradas já estavam a tentar ser solucionadas noutros projetos, mas no entanto, não havia solução para comunicação em tempo real na Web[11]. É precisamente uma solução para este problema que a tecnologia WebRTC nos traz.
Motivado pela constante evolução da Web, pela crescente procura de soluções Web para aplicações, pelas capacidades da tecnologia e pelo gosto pessoal, pretende-se com esta dissertação fazer um estudo aprofundado acerca da tecnologia WebRTC assim como o desenvolvimento de uma aplicação, como prova de conceito, que faça uso da mesma.

1.1 Motivação & Objetivos

Conforme indicado, existem cada vez menos diferenças em capacidades entre as aplicações Web e aplicações Desktop tradicionais. E isto deve-se ao facto de se estar a investir cada vez mais no desenvolvimento de aplicações Web, no entanto, é interssante perceber o porquê de isto vir a acontecer.
Comecemos pelo princípio; há poucos anos atrás, o conceito de aplicações Web era inexistente. Existiam algumas páginas Web mas não podiam ser classificadas de aplicações dado que a sua única funcionalidade era o fornecimento de informações, sendo que nem esses dados eram fornecidos de forma dinâmica. Com a evolução das linguagens de programação da Web e com a introdução da possibilidade de implementar lógica do lado do servidor, começaram a surgir as primeiras aplicações Web.
Com o aparecimento de aplicações Web começa-se então a perceber as suas vantagens que isto pode trazer, vantagens essas relacionadas com:

- Manutenção – torna-se mais facil lançar actualizações uma vez que a aplicação apenas está no servidor[12]
- Mobilidade - permitiu executar aplicações em qualquer dispositivo capacitado de um navegador compatível em qualquer parte do mundo sem ser necessária qualquer instalação, com a evolução também se tornou possível a utilização de tais aplicações em dispositivos móveis[12]
- Compatibilidade – De forma automática as aplicações Web podem ser utilizadas nos mais diversos dispositivos com os mais diversos sistemas operativos, ficando a responsabilidade da compatibilidade com o hardware e sistema operativo do lado dos criadores dos navegadores[12]

- Actualização dos dados – Dadas as caraterísticas da aplicação, funcionam tipicamente com dados actualizados e ligados entre servidor e os vários utilizadores, o que se torna numa grande vantagem para os utilizadores[12]
- Acesso – As aplicações Web têm mais adoção pelos utilizadores dado que permite o acesso de forma mais simples e não requer instalação.[12]

Devido a tudo isto, o uso e desenvolvimento das aplicações Web cresce cada vez mais, e por isso existe o interesse de colmatar as falhas que estas aplicações têm em relação às aplicações tradicionais, desenvolvidas para funcionar no ambiente do sistema operativo. As principais falhas das aplicações Web em relação às aplicações nativas estão relacionadas com menor performance, menor privacidade, a necessidade de conexão à internet e a menor compatibilidade e menor interação com o hardware do dispositivo. [13]
Com o intuito de colmatar essas falhas, estão constantemente a surgir novas tecnologias, quer seja nas redes, nos navegadores ou no desenvolvimento de aplicações. Algumas falhas têm vindo a ser colmatadas, como por exemplo a necessidade de conexão à internet. Com a introdução do HTML5 veio o armazenamento local de dados e de bases de dados, o que possibilita o desenvolvimento de aplicações Web que funcionem sem ligação à internet.[14, p. 5]
É também neste contexto que surge o WebRTC. Esta nova tecnologia pretende quebrar algumas barreiras no actual estado do desenvolvimento de aplicações Web, e para isso, traz consigo a possibilidade de, sem a existência de qualquer extensão, permitir a comunicação directa e em tempo real entre parceiros. Para conseguir isto, fica à responsabilidade do navegador utilizado a implementação da tecnologia *open source* WebRTC. Também para esta tecnologia é importante todo o desenvolvimento de outros mecanismos adjacentes no âmbito da Web, entre eles, a introdução da possibilidade de obter audio e vídeo do utilizador directamente a partir do navegador. É esta uma das características mais utilizadas em aplicações que fazem uso da tecnologia WebRTC pois permite que seja feita uma vídeo chamada directamente entre utilizadores terminais da aplicação.[15]
Assim, pretende-se com esta dissertação explorar mais a fundo esta nova tecnologia de modo perceber o que pode alterar na forma como utilizamos e desenvolvemos aplicações Web. Interessa também elucidar as vantagens e funcionalidades que esta tecnologia apresenta e a facilidade na implementação e na utilização da mesma. Pretende-se também definir correctamente o que está já definido acerca desta tecnologia, o que é deixado a cargo dos desenvolvedores dos navegadores e o que é deixado a cargo dos desenvolvedores de aplicações WebRTC. Um outro objectivo é o de esclarecer o porque da necessidade da ligação ao servidor para estabelecimento de sessão, e de que forma é posteriormente possível a comunicação directa entre os parceiros. Na Figura 1 podemos verificar o funcionamento básico do WebRTC, estando explícito que apesar de a comunicação ser toda feita entre os parceiros, é necessária a ligação ao servidor para o estabelecimento de sessão.

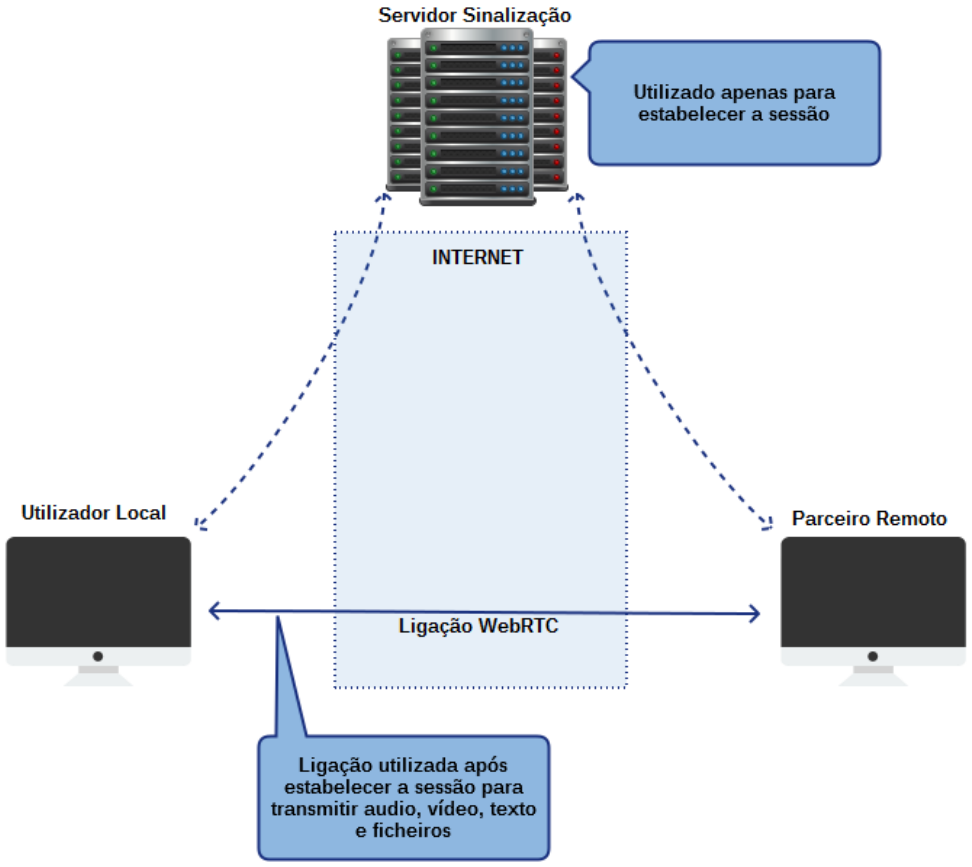


Figura 1 - Estrutura base funcionamento WebRTC

Irá também ser esclarecida a problemática existente na comunicação WebRTC através de NAT e Firewall, assim como a solução para a resolução destes problemas.
Adicionalmente será apresentada uma aplicação WebRTC feita como prova de conceito para demostrar a comunicação WebRTC na qual é possível a comunicação por vídeo chamada, por conversa escrita e é também possível a partilha de ficheiros. Em paralelo com esta aplicação foi desenvolvida também uma aplicação mais simples, apenas com vídeo chamada que pretende demostrar a capacidade de comunicação entre múltiplos parceiros através de WebRTC.

1.2 Plano de tarefas

As várias tarefas necessárias na execução deste documento e das aplicações que o acompanham tiveram de seguir um planeamento. Segue o plano com as tarefas executadas, acompanhadas das datas de execução.

Tarefa	Importância	Finalizado em
Elaboração da proposta de tese de mestrado	Alta	Dezembro 8,2013
Análise de diversas dissertações existentes acerca do tema	Média	Janeiro 3, 2014
Estudo das ferramentas e *frameworks* utilizadas pelo WebRTC	Média	Fevereiro 15, 2014

Estudo de diversos temas relevantes relacionados com WebRTC	Baixa	Março 1, 2014
Esboço da estrutura da dissertação	Alta	Abril 1, 2014
Estudo de utilizações práticas da tecnologia	Média	Abril 15, 2014
Escolha do tipo de aplicação a ser desenvolvido	Média	Abril 15, 2014
Início do desenvolvimento da aplicação Web	Alta	Abril 30, 2014
Finalização da estruturação da aplicação	Média	Maio 9, 2014
Conclusão do estudo das tecnologias	Alta	Maio 12, 2014
Início da primeira fase de testes com utilizadores da aplicação Web	Média	Junho 6, 2014
Conclusão das correções da primeira fase de testes	Média	Junho 20, 2014
Início dos testes de desenvolvimento da utilização do WebRTC	Média	Junho 27, 2014
Início da segunda fase de testes com utilizadores da aplicação Web	Média	Julho 7, 2014
Conclusão das correções da segunda fase de testes	Média	Julho 18, 2014
Início da terceira fase de testes com utilizadores da aplicação Web	Média	Agosto 8, 2014
Início do desenvolvimento da aplicação multi-parceiro	Média	Agosto 8, 2014
Conclusão das correções da terceira fase de testes	Média	Agosto 15, 2014
Testes finais com utilizadores	Alta	Agosto 18, 2014
Conclusão do desenvolvimento e testes da aplicação multi-parceiro	Média	Agosto 28, 2014
Correções finais da aplicação	Alta	Setembro 1, 2014
Início das alterações finais no documento da dissertação	Alta	Setembro 12, 2014
Conclusão do desenvolvimento da aplicação Web	Alta	Setembro 22, 2014
Fim das alterações finais no documento da dissertação	Alta	Outubro 13, 2014

Tabela 1 - Plano de Tarefas

1.3 Metodologias de investigação

As razões desta investigação são fundamentalmente criar um novo interesse numa tecnologia, desenvolver um maior conhecimento sobre o tema e criar conhecimento aos leitores. O resultado esperado desta dissertação é um estudo aprofundado desta nova tecnologia, das suas capacidades e limitações, fazer uma análise crítica e apresentar uma aplicação Web que servirá de apoio à investigação.

No decorrer desta investigação foram utilizados diversos processos, adequados ao tema, que permitiram consolidar, clarificar e criar novos conhecimentos. De forma a serem alcançados os objetivos propostos de uma forma clara e sustentada, foi necessária uma investigação fundamentada e bem estruturada, durante a qual foram tidos em conta 6 pontos essenciais:

Propósito: Clarificar, explicar e experimentar a tecnologia WebRTC, as suas capacidades, limitações, a necessidade da mesma, as suas mais-valias e o que esta tecnologia significa para o futuro da Web. Foi também desenvolvida uma aplicação Web de forma a aplicar e aprofundar o conhecimento teórico.

Produtos: Espera-se com esta dissertação fornecer documentação detalhada e enquadrada acerca do que se trata esta tecnologia, como ela encaixa com as tecnologias atuais, o propósito dela, o que isto poderá permitir, conhecer profundamente a forma como funciona e pretende-se também obter uma aplicação funcional que faça uso desta tecnologia.
Processo: Esta investigação baseou-se no estudo de documentação oficial, especificações, assim como de artigos científicos já existentes sobre as tecnologias e temas adjacentes a ela. Foi feito um levantamento documental que permitiu investigar a tecnologia a fundo e produzir os produtos desejados. Foi ainda feito um estudo acerca dos diversos tipos de aplicação para escolher, de forma sustentada, a aplicação a ser desenvolvida. Foi também será desenvolvida uma aplicação Web.
Participantes: Participaram no processo de investigação diversos agentes. Tais como, professores, colegas investigadores, orientadores da instituição, utilizadores de teste, colegas de trabalho, etc.
Paradigma: A investigação seguiu um paradigma de investigação experimental e objetivista. Isto é justificado pela natureza do problema.
Apresentação: Os resultados obtidos durante a investigação estão apresentados nesta dissertação, assim como também são representados por uma aplicação Web que faz uso da tecnologia em estudo.
Com isto, entende-se que o desenvolvimento deste estudo baseou-se em pesquisa. Nomeadamente pesquisa que permitisse de forma clara compreender tudo o que seria necessário para o desenvolvimento de aplicações WebRTC. Para isso foi necessário analisar artigos, especificações e tutoriais, de forma a perceber o que seria necessário ao funcionamento e ao desenvolvimento de aplicações. Esta pesquisa procurou basear-se nos mecanismos básicos, posteriormente evoluindo para o estudo das várias tecnologias necessárias para o funcionamento de WebRTC. Durante o desenvolvimento deste trabalho, foram também seguidos alguns tutoriais que permitiram a utilização de tecnologias como Node.JS, assim como a utilização de funções cruciais para o desenvolvimento da aplicação, como por exemplo que permitissem a obtenção de áudio e vídeo por parte do utilizador. Os passos necessários para o desenvolvimento resultaram de uma análise às necessidades da aplicação, assim como da pesquisa acerca do funcionamento de algumas soluções já existentes.
A base para a pesquisa das várias tecnologias foi a análise das suas especificações técnicas, normalmente fornecidas pela IETF (Internet Engineering Task Force) e disponíveis na Web.

1.4 Estado da Arte

Perante a realização deste estudo, torna-se importante ficar a conhecer o estado atual do desenvolvimento da tecnologia, o que traz de novo e como se insere em relação às tecnologias atualmente existentes. Esse estudo será feito neste sub-capítulo.

1.4.1 Estado atual

Antes do aparecimento da tecnologia WebRTC, sempre que se desejava efetuar uma chamada de vídeo através de um navegador era obrigatoriamente necessária a instalação de uma extensão Flash ou Java que permitisse tal comunicação, sendo que ainda muitas vezes seria posteriormente necessário utilizar uma aplicação à parte, fora do navegador.[16]

O que WebRTC vem permitir é uma utilização mais simples, rápida e natural (no sentido de apenas usar uma aplicação web, sem instalações necessárias) para o utilizador e permitir aos desenvolvedores de aplicações, a possibilidade de facilmente implementar uma solução que permita a comunicação direta em tempo real que permita, não só vídeo chamada, como envio de ficheiros e texto. Com o uso desta tecnologia, será possível reduzir drasticamente, de milhares de horas, para apenas alguns dias, o tempo de desenvolvimento de tal aplicação.[16]

Tudo isto fica claro se analisarmos o objetivo proposto pelo WebRTC: permitir o desenvolvimento de aplicações de comunicação em tempo real, no navegador, através de APIs JavaScript simples e HTML5.[15]

Perante a evolução que esta tecnologia está a ter, permanece uma questão: Como se compara com aplicações como o Skype? Na verdade, tal comparação não deve ser feita, já que o Skype se trata de uma aplicação, enquanto que WebRTC se trata de uma tecnologia com um conjunto de capacidades.[17] No entanto, esta questão é obvia, uma vez que aplicações que façam uso de WebRTC oferecem funcionalidades semelhantes a aplicações como Skype.[16] Sobre isto, importa referir que, neste momento, o Skype é considerado o serviço de referência globalmente e está disponível em todos ou quase todos os dispositivos,[17] no entanto, a tecnologia WebRTC ainda está a aparecer, inclusivamente ainda está em desenvolvimento, sendo que apresenta fortes vantagens em termos de simplicidade (utilização e desenvolvimento), facilidade de acesso e a não necessidade de qualquer tipo de instalação.[17] Isto permite que qualquer aplicação WebRTC seja acedida imediatamente, sem qualquer necessidade de configuração ou instalação, a partir de qualquer navegador compatível.

De forma a compreender melhor as possibilidades e as vantagens na utilização da tecnologia WebRTC, iremos de seguida analisar algumas soluções já disponíveis que fazem uso desta tecnologia.

1.4.2 Soluções WebRTC disponíveis

Existem já neste momento várias soluções WebRTC disponíveis, algumas *open source*, algumas gratuitas, outras com taxas para uso empresarial, com funcionamento semelhante. Algumas destas soluções permitem o desenvolvimento de aplicações WebRTC ao passo que outras apenas se apresentam como soluções prontas a serem usadas, suportadas por servidores próprios ou por servidores do utilizador, sendo disponibilizada a aplicação como um produto.
Iremos então analisar algumas destas soluções disponíveis entre as quais, a mais popular, o Google Hangouts.

1.4.2.1 simpleWebRTC

Provavelmente a solução mais conhecida e utilizada, à exceção do Google Hangouts, na situação atual da evolução do WebRTC. Permite conhecer em primeira mão, de forma simples, todas as capacidades da tecnologia, permite também o desenvolvimento rápido e fácil de aplicações WebRTC. É utilizada por inúmeras outras soluções, inclusive a solução de conversação em grupo apresentada na íntegra neste documento. [18]
Adicionalmente, para permitir o desenvolvimento de aplicações mais complexas e específicas, esta solução apresenta também alguns módulos que possibilitam isso como, entre outros:

- *Signalmaster* – Servidor de sinalização; [18]
- Webrtc.js – Para além de gerir múltiplas *PeerConnection*, permite de forma transparente a compatibilidade entre navegadores capacitados, permite também alta personalização; [18]
- *Webrtcsupport* – Permite verificar as capacidades do navegador, assim como fazer a construção correta de elementos necessários para conexões WebRTC, em diferentes navegadores; [18]
- *getScreenMedia* – Possibilita a partilha de ecrã; [18]
- *hark* – Deteta quem está a falar numa conversação.[18]

1.4.2.2 Talky

Esta solução é construída a partir da simpleWebRTC. Ainda assim, torna-se importante ser mencionada uma vez que o objetivo desta solução é de ajudar a testar e a desenvolver o WebRTC. Para isso, esta solução foca-se em manter-se atualizada e obter o feedback dos seus utilizadores, mantendo assim estatísticas de utilização e performance. [19][20]
Permite conversas em grupo, partilha de ecrã, salas de conversação com autenticação e uma utilização simples e rápida. Fica apenas a nota que, após testes, se confirmou que a partilha de ecrã só é possível com a adição de uma extensão ao navegador.[19][20]

1.4.2.3 EasyRTC

Apresenta-se como uma das soluções mais preparadas para uso empresarial. Trata-se de uma solução pronta a ser implementada e classifica-se como segura tendo como objetivo mascarar a constante evolução do WebRTC, oferecendo uma API própria. Para além da opção empresarial que é paga, oferece também uma versão gratuita que permite a criação de aplicações WebRTC na sua totalidade. A versão empresarial é um produto pronto a ser utilizado que pode ser personalizado de acordo com a empresa em causa.[21]

Esta solução foi já classificada como a melhor ferramenta WebRTC na primeira conferência e exposição sobre WebRTC em 2012, ganhando já outro prémio na edição 2013 da mesma conferência.[22]

1.4.2.4 appear.in

Esta solução apresenta-se pronta a usar, basta aceder à página e inserir o nome da sala de comunicação que se pretende utilizar. Estas salas servem como canais de comunicação isto é, a comunicação dos parceiros é partilhada pelos outros parceiros da sala, mas não transmitida para utilizadores noutras salas. Estas salas tratam-se de canais de comunicação diferenciados entre si, ou seja, a comunicação de um canal só é partilhada nesse canal. Permite também partilhar ou vedar o acesso à sala por utilizadores indesejados. Permite gratuitamente a conversação entre oito participantes e permite também a conversação por texto.[23]

Foi construída e é mantida por uma pequena equipa de apenas nove elementos, surge do seio de uma operadora, na qual já estava a ser desenvolvido o produto desde 2012.[24]

1.4.2.5 vLine, Bistri e OnSIP

A vLine trata-se de mais uma solução pronta a utilizar tal como appear.in, no entanto esta oferece possibilidade de desenvolvimento de aplicações baseado numa API própria. Esta API encontra-se totalmente documentada e o desenvolvimento de uma aplicação a partir dela pode ser acompanhado por tutoriais oferecidos na própria página.[25]

Tal como esta solução, existem outras semelhantes, como é o caso da Bistri, que é mais completa, no entanto, algumas das características que a tornam mais completa, são pagas.[26]

Uma outra solução muito semelhante à Bistri é OnSIP, também com componente paga, sendo importante realçar o facto de permitir efetuar chamadas para o endereço de um outro utilizador.[23][27]

1.4.2.6 Google Hangouts

O Hangouts, do Google, é certamente a solução mais conhecida, divulgada e testada cujo funcionamento faz uso de WebRTC.

Esta solução apresenta inúmeras capacidades incluindo:

- Conversação com múltiplos utilizadores, com um máximo de dez utilizadores;

- Compatível entre navegadores de computadores e aplicações próprias disponíveis para a maioria dos smartphones;
- Partilha de mensagens, partilha de ecrã e partilha de ficheiros;
- Centralização do parceiro de conversação a falar no momento.[28][29]

A partir de Julho de 2014, deixou de ser necessário a instalação de uma extensão no navegador para a utilização do Hangouts, no caso de o navegador ser o Chrome, para o caso do Firefox, a Google disponibiliza ainda a extensão necessária.[30][31][32]
Atualmente esta solução apresenta-se como uma das mais utilizadas e mais completas soluções de vídeo chamada no mundo, sendo que grande parte da sua popularidade reside na ligação com a Google e Android.[33][34]

1.5 Organização do documento

Este capítulo, o Capítulo 1, trata-se de uma apresentação à dissertação, nele é explicado como surge o tema, o estudo que será feito e qual o resultado que se pretende obter com a realização deste trabalho. É feita uma análise do estado da arte, é apresentado o plano de tarefas do desenvolvimento do projeto e é feita uma análise às metodologias de investigação utilizadas na realização do trabalho.
De seguida, no Capítulo 2, são apresentadas as diversas tecnologias estudadas e utilizadas neste trabalho. Este estudo é feito com o intuito de perceber o funcionamento da tecnologia WebRTC, relaciona-la com as várias tecnologias existentes e compreender o que é necessário para a implementação de WebRTC.
Posteriormente é apresentada a aplicação desenvolvida como prova de conceito. Nessa apresentação declara-se o que se pretende obter com este desenvolvimento, explicam-se as escolhas feitas e demostram-se os passos desenvolvidos que permitiram o resultado final. Sobre esse resultado final é apresentado e demostrado o funcionamento. Adicionalmente, é explicado o desenvolvimento de uma aplicação multi-parceiro demostrativa das capacidades WebRTC.
Por fim, fecha-se com uma conclusão onde é feita uma análise ao trabalho desenvolvido, e também a futuros desenvolvimentos feitos com base na aplicação desenvolvida.
No final do documento existem ainda alguns anexos que servem de apoio ao documento.

1.6 Sumário

A constante evolução da Web levou-nos ao desenvolvimento de novas tecnologias Web, permitindo cada vez mais funcionalidades e de forma geral, cada vez de uma forma mais

simples. Esta mesma evolução levou ao aparecimento da tecnologia WebRTC que se pretende explicar com este documento. Assim, espera-se com este documento ilustrar a importância que esta tecnologia pode vir a ter, e espera-se também mostrar que esta tecnologia se trata de algo que altera a forma como utilizamos e como desenvolvemos aplicações Web. É também importante o estudo do funcionamento da tecnologia, assim como das tecnologias que permitem a utilização e implementação de WebRTC.

Este estudo é motivado pela forte evolução que tem vindo a existir nos últimos anos no ambiente Web, tanto para utilizadores como para desenvolvedores, e pelo interesse existente numa tecnologia com características como WebRTC. Os maiores pontos de interesse do WebRTC são: a não necessidade de qualquer instalação (nem de aplicações nem de programas), a comunicação direta entre terminais, a facilidade de implementação e também o facto de se tratar de um projecto de tecnologia aberta (open source).

Para a realização de forma organizada deste trabalho foi necessário elaborar um plano de trabalhos, definindo as tarefas a realizar e os prazos, esse plano é também apresentado neste capítulo, assim como uma pequena análise das Metodologias de Investigação aqui utilizadas.

De forma a enquadrar melhor esta tecnologia, o que permite, como se perfila em relação às soluções existentes e também com o objetivo de apresentar soluções que façam uso desta mesma tecnologia, é apresentada uma pequena análise acerca de tudo isto. Nesta mesma análise, a solução é comparada de forma geral com o existente, e são apresentadas soluções completas que fazem uso da tecnologia WebRTC.

2 Tecnologias

Neste capítulo serão analisadas algumas das tecnologias cruciais para o funcionamento de WebRTC. Inicialmente são apresentadas as mais básicas tecnologias que permitem o desenvolvimento de aplicações Web, como o HTML5 e o JavaScript, depois é feita uma curta introdução acerca dos codecs e sua utilização em WebRTC, assim como a importância que assumem na especificação e estandardização do WebRTC. Uma vez introduzido o tema WebRTC, torna-se importante analisar esta tecnologia, desde a apresentação das suas APIs, como discussão de temas como segurança e compatibilidade.
Seguidamente, e relacionado com as dificuldades existentes na utilização de WebRTC, analisam-se os problemas de comunicação através de NAT, assim como as opções que permitem a solução desses mesmos problemas. Aprofundando a análise da comunicação WebRTC, é feito também um estudo acerca dos mecanismos de sinalização, por forma a permitir um maior conhecimento na escolha do mecanismo a utilizar, e são também apresentados protocolos de DataChannel que tratam da comunicação em WebRTC.
Por último, será analisada a solução Node.JS, assim como alguns dos seus módulos, uma vez que esta solução é necessária para a implementação da solução.

2.1 HTML5

HTML é uma linguagem de marcação utilizada para a produção de páginas Web. Possui um sistema de anotação, através de *tags,* que permite distinguir diversos elementos através da sua sintaxe. É com recurso a esta linguagem que se definem todos os elementos, mais ou menos complexos, de uma página Web.[14, p. 5]

Posto isto, será mais relevante focar-nos nos elementos do HTML que estão diretamente relacionados com o WebRTC. Para isso será necessário introduzir o HTML5.
Trata-se então da quinta versão da linguagem HTML. Esta nova versão vem adaptar o HTML às novas exigências da Web com novas capacidades, maior simplicidade e melhor interação com os mais recentes tipos de multimédia.
No contexto da tecnologia WebRTC, é importante realçar as novas *tags* de <vídeo> e <audio>, dado que são comummente utilizadas nas aplicações WebRTC, assim como métodos, eventos e conceitos como getUserMedia (para requisitar permissão para aceder e para aceder a vídeo e/ou áudio do utilizador), MediaStream (fluxo de dados multimédia), addtrack (adiciona o MediaStreamTrack ao MediaStream), onaddtrack (evento ao adicionar MediaStreamTrack), entre muitos outros. Estas novidades permitem que as aplicações desenvolvidas sejam compatíveis com mais dispositivos e navegadores, uma vez que está tudo standardizado e não são necessárias extensões, ao contrário do que existia até agora.[35][36][37]
Para além das tags mencionadas, é muitas vezes também encontrada a tag <canvas> nas aplicações WebRTC, utilizada para alterar o vídeo apresentado. Esta tag também é uma novidade trazida pelo HTML5.[38]

2.2 JavaScript

O JavaScript é uma linguagem de programação orientada a objectos utilizada normalmente na Web. Permite ser utilizada do lado do cliente ou do lado do servidor. Contém funções que permitem interação com o utilizador, algum controlo sobre o navegador, comunicação assíncrona e alteração visual de toda a informação apresentada.[39]
A partir desta tecnologia foram criadas diversas implementações com o mais variado tipo de utilização, desde *frameworks* com o objetivo de alterar o visual de uma aplicação Web até protocolos de comunicação.[40]

2.3 Codecs

Algo que será crucial para o sucesso na adoção do WebRTC será a interoperabilidade de voz e vídeo entre aplicações WebRTC, sendo para isso necessária a especificação dos codecs a serem utilizados. Estes codecs permitirão também adaptar os dados de multimédia enviados de forma a permitir uma qualidade de serviço adequada.
De forma a garantir a tal interoperabilidade, assim como capacitar a adaptação dos dados à ligação, terão de ser definidos alguns requisitos para os codecs compatíveis com WebRTC. Esses requisitos são:

- Audio [41][42, p. 264]
  - PCMA/PCMU – Pulse Code Modulation [43]
  - Telephone Event – define um conjunto de eventos de telefonia necessários como tons, sinais, etc.. [44]
  - Opus – Codec de áudio e voz desenhado para um largo espectro de aplicações áudio-interativas [45]
- Video [41][42, p. 264]
  - Terá suportar pelo menos 10 frames por segundo e como recomendado, deve suportar 30 frames por segundo
  - Opcionalmente poderá oferecer suporte para universos de cores adicionais
  - Terá de suportar uma resolução mínima de 320x240
  - Deve suportar as seguintes resoluções: 1280x720, 720x480, 1024x768, 800x600, 640x480, 640x360 e 320x240

Devido à especificação WebRTC apenas definir os requisitos dos codecs, existem vários possíveis, pelo que a implementação dos diferentes codecs varia entre navegadores Web. No entanto, estão a ser estudadas propostas para definir um *codec* vídeo obrigatório no WebRTC.[41][42, p. 264]
Por exemplo, no caso do Firefox, os codecs utilizados para áudio são Opus e G.711, e para vídeo o VP8.[46] O Chrome apesar de usar também VP8 para vídeo, apresenta maior compatibilidade para áudio, oferecendo suporte para os seguintes codecs áudio: iSAC, iLBC, G.711, G.722 e DTMF.[47][48]

2.4 WebRTC

WebRTC (Web Real Time Communication), tal como o nome indica, é uma tecnologia Web que permite comunicações em tempo real. Este é um projeto *open source*, gratuito e em vias de ser normalizado. Tem como objetivo a comunicação direta e em tempo real entre navegadores, sem recurso a extensões ou qualquer outro programa externo. [49]
Para estender as suas capacidades, permite ainda o acesso a componentes internos como a câmara e o microfone com o intuito de possibilitar transmissão deste tipo de dados de forma direta e em tempo real. [49]

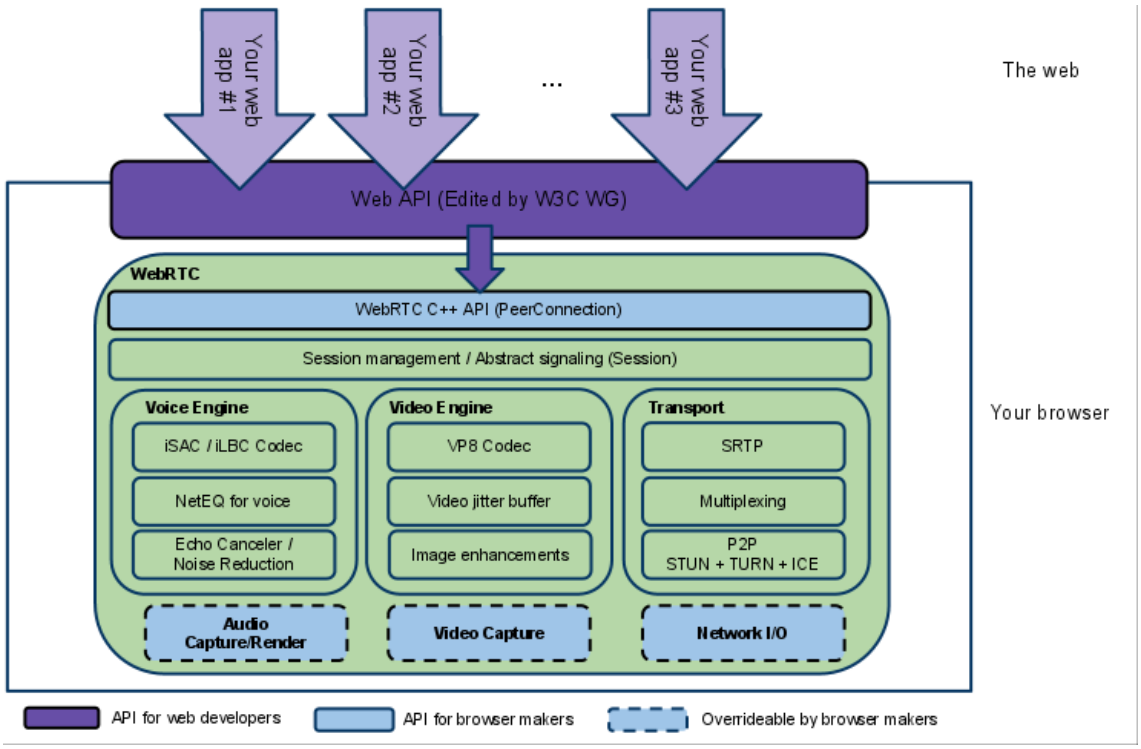


Figura 2 - Arquitetura WebRTC[49]

Esta tecnologia é constituída por duas APIs distintas, conforme representado na Figura 2. A WebRTC C++ API para desenvolvimento de navegadores e plataformas capacitadas de WebRTC e a WebRTC Web API utilizada por pessoas que pretendam usufruir desta tecnologia no desenvolvimento de aplicações Web, estas duas APIs são apresentadas de seguida.[49]

2.4.1 WebRTC C++ API

Dado tratar-se de uma tecnologia *open source*, o código que permite implementar capacidades WebRTC a um navegador ou outra qualquer aplicação encontra-se disponível para qualquer pessoa. Este código, o WebRTC C++ API está disponível para utilização, estudo e até para deteção de erros e melhorias, todo o código se encontra disponível. [49]
Esta API contém ainda mecanismos de Sessão/Transporte, *framework* para voz, equalização de voz e *framework* para vídeo.[49]

2.4.2 WebRTC API

Para quem pretenda desenvolver aplicações Web, com a tecnologia WebRTC deverá fazer uso desta API. Tipicamente as aplicações são desenvolvidas em JavaScript, do lado do cliente, por forma a minimizar a comunicação com o servidor. Essa comunicação com o servidor deve apenas ser necessária para estabelecer o contacto inicial com um ou mais parceiros em WebRTC. Analisemos então alguns conceitos importantes desta API.[50][51][19]

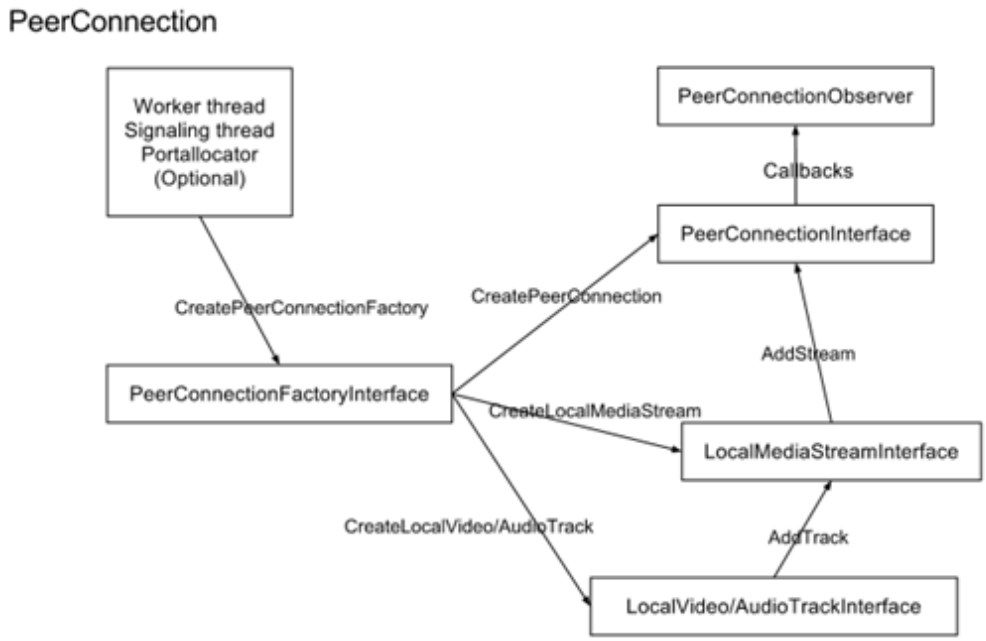


Figura 3- Esquema do funcionamento da *PeerConnection*[51]

- *PeerConnection* API (Ver) [52]
  - Permite comunicação direta entre dois terminais, sendo apenas necessária haver negociação de sinalização. Para tal negociação é implementado no servidor um canal de sinalização tipicamente fazendo uso de *WebSockets*. *PeerConnection* utiliza ICE, STUN e TURN, de modo a ser possível comunicação sobre NAT e Firewall. As tecnologias ICE, STUN e TURN são apresentadas posteriormente neste documento. Pode ser verificado o seu funcionamento através da Figura 3
- *MediaStream* API [51][53]
  - Conforme representado na Figura 4, *MediaStream* é um encapsulador de dados vídeo e áudio, podendo também ser outro tipo de dados.
  - Este objeto, *MediaStream*, pode ser utilizado para reproduzir, gravar ou enviar o seu conteúdo para outro parceiro.
  - Possibilita processamento de áudio e vídeo, assim como controlo de largura de banda. Permite também adicionar, remover ou silenciar a multimédia na aplicação.
  - Existem dois tipos de *MediaStream*, a local e a remota.

- A Local *MediaStream* é utilizada para conter os dados obtidos com o método *getUserMedia*(), que serve para obter multimédia do terminal, podendo ainda ser parametrizável.
- A transmissão de *MediaStream* pode ocorrer através de SRTP, RTP, RTCP, fazendo ainda uso de DTLS.

Figura 4- Esquema do conteúdo de uma Stream[51]

❖ ORTC API
- Surge como uma alternativa ao *PeerConnection* API, atualmente o protótipo apresenta-se como extensão para o Chrome e para Internet Explorer.[19]

❖ *getUserMedia*
- Método para obter dados de áudio e de vídeo a partir do microfone e camara do utilizador. [19]

❖ *Simulcast*
- Permite obter os *MediaStream* em múltiplas resoluções e *framerate* e enviar para o parceiro. [19]

❖ *mediaConstraints*
- Permite especificar resoluções vídeos restritas, sendo isso importante para controlo de largura de banda. [19]
- Permite especificar no momento de pedido de acesso à multimédia e permite também alterar um *MediaStream* existente. [19]

❖ TURN support
- Suporta TURN, o que é necessário para continuar as comunicações sobre NAT e Firewall quando falha o STUN. [19]

❖ *WebAudio* Integration
- Esta integração permite o processamento de áudio nativo. Para as aplicações WebRTC é importante que isto exista e que seja possível a sua utilização nos *MediaStream*. [19]

- *dataChannels*
  - Este suporte pode ser:
    - Fiável – significa que a entrega de dados na ordem correta é garantida.
    - Não fiável – Significa que a ordem de entrega de dados pode não ser respeitada e que pode ocorrer falha de entrega de dados. [19]
- *Screen Sharing*
  - Permite requisitar acesso *MediaStream* ao ecrã do utilizador, ou seja, permite partilha de ecrã. [19]
  - Em Chrome é necessário uma extensão, com exceção à Aplicação Hangouts que tem permissão de acesso. [19]
- *Stream re-broadcasting*
  - Permite comunicação direta entre múltiplos utilizadores através da possibilidade de colocar um objeto *MediaStream* numa outra *PeerConnection*. [19]
- *Multiple Streams*
  - Permite transporte de múltiplos *MediaStream* em uma só *PeerConnection*. [19]
- *Solid interoperability*
  - É importante que exista total e sólida interoperabilidade para ser possível comunicação entre navegadores diferentes de forma simples com WebRTC. [19]
- *Echo cancellation*
  - Possibilita o cancelamento de eco no áudio para evitar problemas com feedback. [19]

2.4.3 Segurança

Uma grande preocupação quanto à utilização desta nova tecnologia prende-se com a segurança necessária, uma vez que existe preocupação que, por exemplo, uma videochamada não seja escutada ou gravada ou que por exemplo, não seja recebida uma mensagem enviada por um destinatário que não um parceiro da conversação. [54]

A segurança em WebRTC é garantida por métodos standard de encriptação, métodos com provas dadas quanto à proteção dos utilizadores quanto a intrusões indevidas à sua informação privada. WebRTC tem características nativas embutidas que respondem às questões de segurança impostas. [54]

Começando pelo básico, a arquitetura e forma de funcionamento do WebRTC implica especificamente a não necessidade de extensões. Não havendo necessidade de instalação de extensões, a segurança nesse ponto fica a cargo do navegador em utilização e é automaticamente

eliminada uma ameaça uma vez que tais extensões são muitas vezes acompanhados de *Malware*. Adicionalmente, o acesso aos dispositivos do utilizador tem sempre de ser permitida pelo próprio utilizador, sendo que sempre que tal acesso está a decorrer, tal é indicado pelo navegador, tipicamente com um símbolo na aba, e muitas vezes também pelo *hardware*, sendo que tradicionalmente o funcionamento das webcams é indicado por uma luz ao lado da câmara. [54]
De forma à WebRTC transmitir dados, esses dados devem primeiro ser encapsulados e encriptados utilizado o protocolo DTLS, que ele próprio é um standard e é obrigatoriamente suportado por todos os navegadores com capacidades WebRTC. Este protocolo é desenhado de forma a evitar que a informação transmitida seja escutada ou alterada de alguma forma. Este protocolo foi modelado a partir do TLS, um protocolo que oferece encriptação total com métodos de encriptação assimétricos, confidencialidade de dados e autenticação de mensagens. Isto permite que os dados transmitidos por WebRTC sejam transportados seguramente por uma qualquer conexão SSL (Secure Sockets Layer). [54]
WebRTC oferece encriptação durante toda a ligação entre parceiros com praticamente qualquer tipo de servidor. Quando são utilizados servidores intermediários, estes só irão tratar a camada UDP do pacote WebRTC, quer isto dizer que, não podem nem sabem alterar ou interpretar a camada de dados aplicacionais, os dados WebRTC. Assim, os servidores intermediários não poderão descodificar os dados enviados entre os utilizadores.
A própria camada de sinalização poderá ser encriptada, dependendo do mecanismo de sinalização utilizado. No caso dos *WebSockets* por exemplo, a transmissão poderá ser feita sobre Secure *WebSockets*, que também faz uso de TLS para encriptar a conexão *WebSocket*. Podem também ser utilizados mecanismos de autenticação para garantir autenticidade e autorização, permitindo também inclusive fazer controlo de utilizadores e acessos.[54]

2.4.4 Compatibilidade

Para cumprir com os objetivos desejados do WebRTC, será crucial que exista inter-compatibilidade entre dispositivos diferentes e navegadores diferentes, no entanto, isto não depende inteiramente do projeto WebRTC em si, mas principalmente dos fabricantes dos navegadores e dos dispositivos. Para além de serem capazes de trabalhar com WebRTC, as plataformas necessitam também de suportar HTML5 e *JavaScript*.

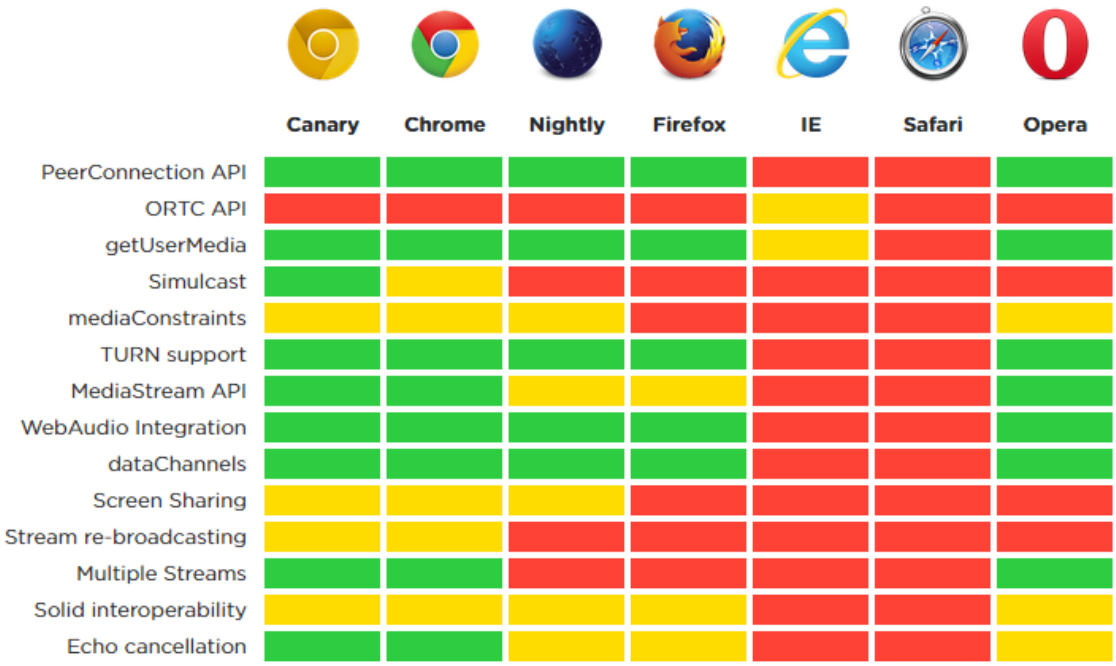


Figura 5 - Estado atual da compatibilidade dos navegadores com a tecnologia WebRTC[19]

Podemos verificar na Figura 5, a compatibilidade das funções apresentadas, nos navegadores mais utilizados com tecnologia WebRTC. Note-se que se apresentam as várias funções do WebRTC e a sua compatibilidade com cada navegador, sendo que verde representa total compatibilidade, amarelo compatibilidade parcial e vermelho representa incompatibilidade.
Para além da compatibilidade nas funções, existem já análises importantes quanto ao feedback dos utilizadores sobre qualidade do serviço. Os dados apresentados na Figura 6 dizem respeito ao feedback dos utilizadores da aplicação Talky. [19] Deste estudo podem-se tirar duas conclusões:

- A utilização em Chrome ou Firefox da tecnologia produz resultados muito semelhantes, sendo que a diferença provavelmente está relacionada com a subjetividade de cada utilizador.
- A utilização de WebRTC em Chrome ou Firefox satisfaz, sendo que apresenta resultados muito bons em áudio e vídeo no entanto, não cumpre o esperado em grande parte dos casos a alta qualidade das vídeo chamadas.

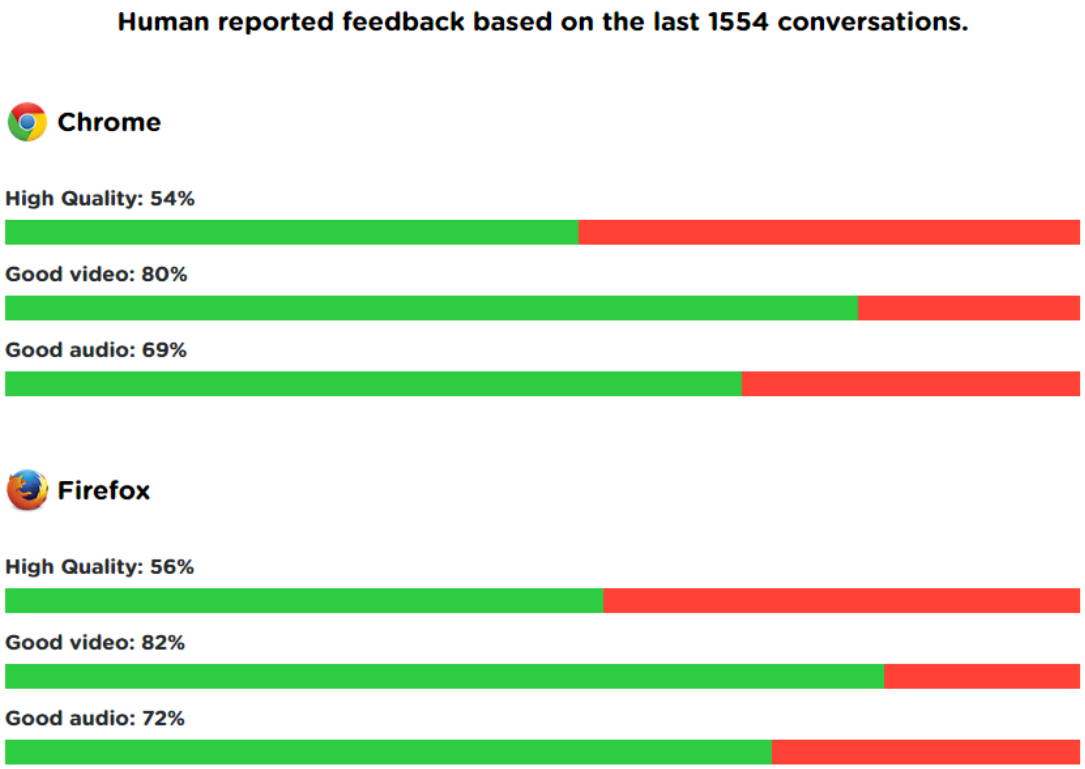


Figura 6 - Feedback dos utilizadores acerca da qualidade de comunicação em WebRTC[19]

2.5 Comunicação através de NAT

Na sua forma mais básica, NAT é um método no qual os IPs são mapeados de uma rede para a outra, de forma transparente para os utilizadores finais. Este método permite que nos dados de um datagrama IP sejam alteradas informações de endereços de rede, mascarando-os permitindo assim que exista comunicação entre redes distintas.[55]

A necessidade do NAT é evidente em casos onde existe comunicação entre duas redes internas distintas, isto é, duas redes internas que estão ligadas através de uma ligação à internet. De forma a perceber melhor o conceito, imaginemos o seguinte cenário:

Na Rede interna A existe um computador PC A com o IP 10.1.1.2/8;

Na Rede interna B existe um computador PC B com o IP 10.1.1.2/8;

As redes não têm qualquer ligação uma à outra, a não ser que ambas têm acesso à internet.

Imaginemos então que não existe NAT, o pacote sai do PC A, sai da rede interna e imediatamente é perdido pois no domínio público não existem IPs privados, logo ninguém tem caminho definido para aquele IP. É então necessária a existência de NAT na fronteira entre as redes internas e a internet, conforme especificado na Figura 7:

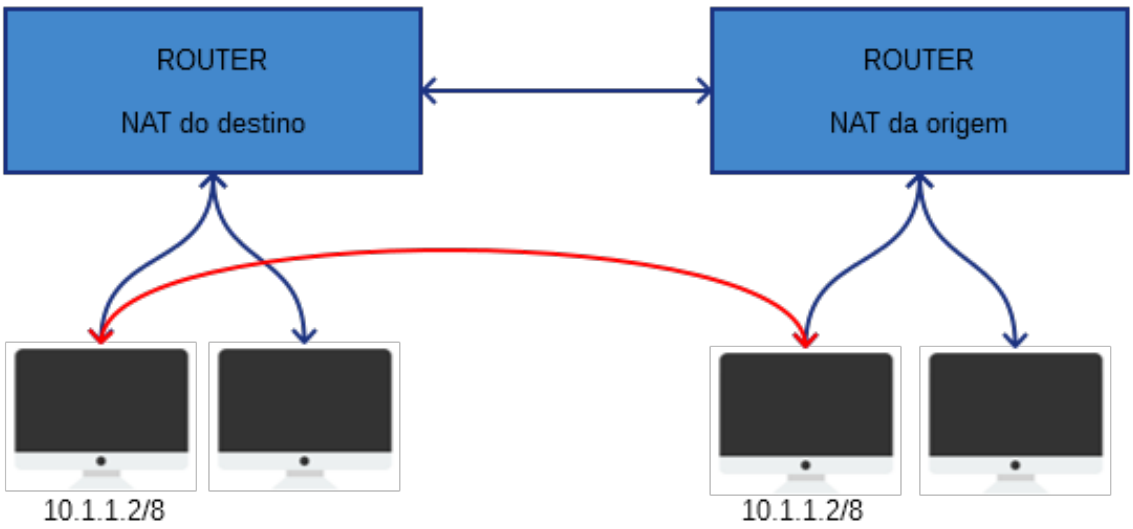


Figura 7 - Comunicação através de NAT

Com NAT, o que acontece é que quando os pacotes do PC A saem da rede interna, o NAT altera as informações de IP, com um mapeamento próprio, ficando no cabeçalho o endereço de IP público da rede, sendo assim possível a comutação dos pacotes no domínio público até a rede de destino, onde o NAT do destino recebe o pacote, e analisa-o para determinar qual o IP interno de destino, voltando a alterar as informações de rede e encaminhando-o para o PC B.[56][57]
Se para o caso de comunicação tradicional cliente/servidor, NAT funciona bem, este método causa muitos problemas em comunicações P2P. As aplicações P2P normalmente funcionam intercambiando uma combinação IP:porto, sendo esses portos os que iram ser utilizados para comunicar ou receber comunicações. Esta forma de funcionamento entra em conflito com NAT e Firewall, uma vez que nesses casos, para além de que se existir Firewall, provavelmente o porto estará bloqueado para comunicações, existindo NATs no percurso da ligação o IP:porto indicado nem sequer estará ao alcance da rede. Por outras palavras, a comunicação é bloqueada ou perdida.[58][59]
Dada a necessidade de comunicação P2P em WebRTC, torna-se fácil perceber que WebRTC tem problemas com NAT e até Firewall (devido bloqueio de portos de rede), no entanto existem os protocolos ICE, STUN e TURN para contornar estes problemas, iremos de seguida falar sobre eles.

2.5.1 ICE - Interactive Connectivity Establishment

ICE é um protocolo para NAT transversal, que permite que sejam estabelecidas sessões de multimédia baseadas em UDP, criadas num modelo de pedido/resposta. [60][61]
Este protocolo fornece capacidade de fazer transmissão de qualquer tipo de protocolo orientado à sessão, entre NAT e Firewall, sendo que foi criado para funcionar com SDP. ICE faz uso de STUN e TURN que veremos mais à frente, é extremamente robusto, o que significa que funciona

mesmo sobre tipologias complexas e oferece também alguma otimização uma vez que só fará uso de TURN (intermediários) caso ocorra alguma falha.[60][61]
No caso do WebRTC, ICE possibilita que haja comunicação entre utilizadores mesmo que entre eles exista NAT, uma vez que fornece a cada um dos parceiros um caminho candidato para estabelecer conexão com o outro parceiro. Este protocolo contém ainda um método de segurança que faz com que só sejam enviados dados após a troca de informação ICE entre os utilizadores. [60]

2.5.2 STUN - Session Traversal Utilities for NAT

O protocolo STUN funciona como uma ferramenta auxiliar quando é necessário fazer a travessia de NAT e pode ser utilizado por um terminal para descobrir a combinação IP:porto alocado por NAT, como ilustrado na Figura 8. Pode também ser utilizado para verificar conectividade entre terminais ou para manter o mapeamento NAT atribuído. Este protocolo não deve ser confundido com ICE pois este protocolo por si mesmo não consegue fazer a travessia de NAT, mas sim uma ferramenta auxiliar para conseguir fazer essa travessia. STUN permite também a retransmissão de pacotes de dados entre dois terminais. [62]

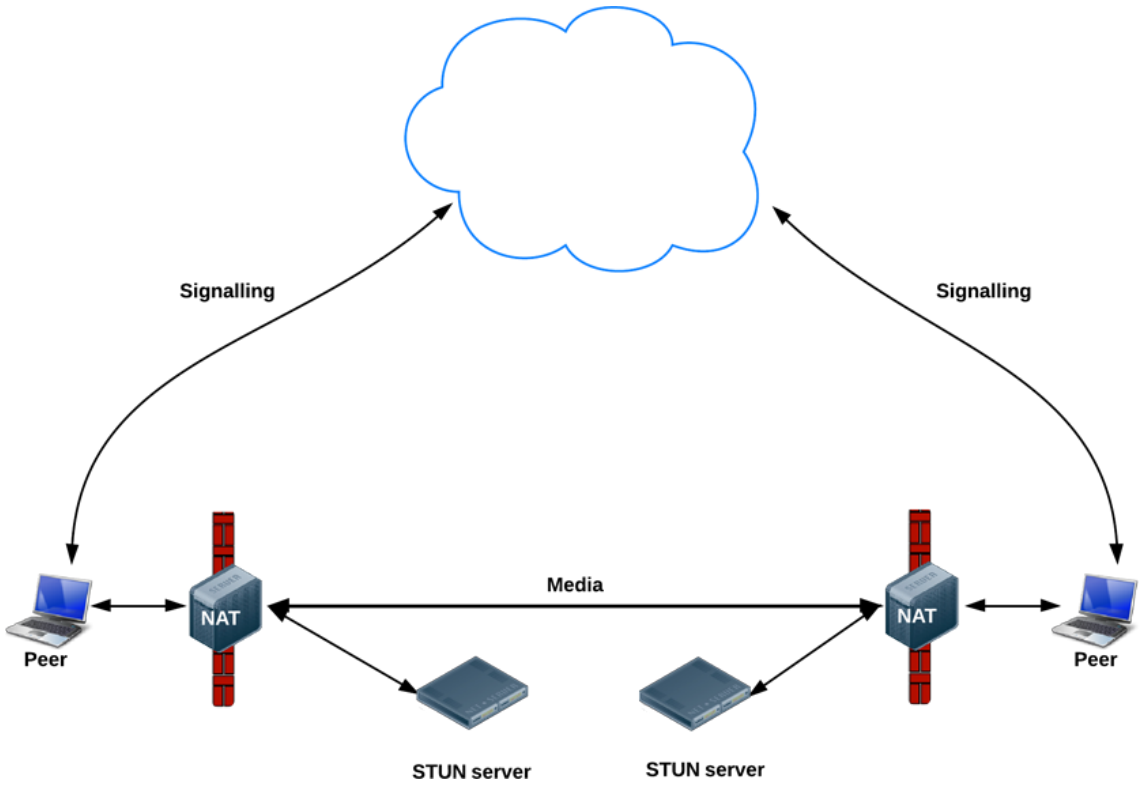


Figura 8 - Funcionamento STUN[55]

O funcionamento é simples, um cliente STUN envia um pedido ao servidor STUN de forma a descobrir qual a combinação IP:porto que lhe foi atribuída por NAT. Este pedido é atendido pelo

servidor que envia a informação. Com esta informação, o cliente já consegue saber qual o seu IP:porto público, informando assim o outro terminal da ligação P2P.[62]
Os servidores STUN podem ser configurados, no entanto geralmente, assim como na aplicação desenvolvida, são utilizados servidores que estão na internet disponibilizados pela Google.[63]

2.5.3 TURN - Traversal Using Relays around NAT

Considerando o que foi visto nos capítulos ICE e STUN, é necessário considerar que podem ainda existir falhas em casos esporádicos. Quando essas falhas ocorrem torna-se impossível determinar um caminho direto entre os dois parceiros P2P, sendo então necessário recorrer a um ponto intermédio. Este ponto intermédio está normalmente situado algures na internet e o seu trabalho será retransmitir pacotes entre os dois parceiros P2P situados atrás de NAT. Este método de funcionamento está definido no protocolo TURN, esquematizado na Figura 9.[64]
Quer isto dizer que quando TURN é necessário, a transmissão de um parceiro P2P passa por um servidor TURN intermediário que retransmite a comunicação ao outro parceiro P2P. Esta retransmissão faz com que no servidor exista uma maior carga tanto em termos de rede como em termos de processamento. [65]
Conforme referido anteriormente, é o protocolo ICE que determina quando existe a necessidade de fazer a utilização de TURN.[65]

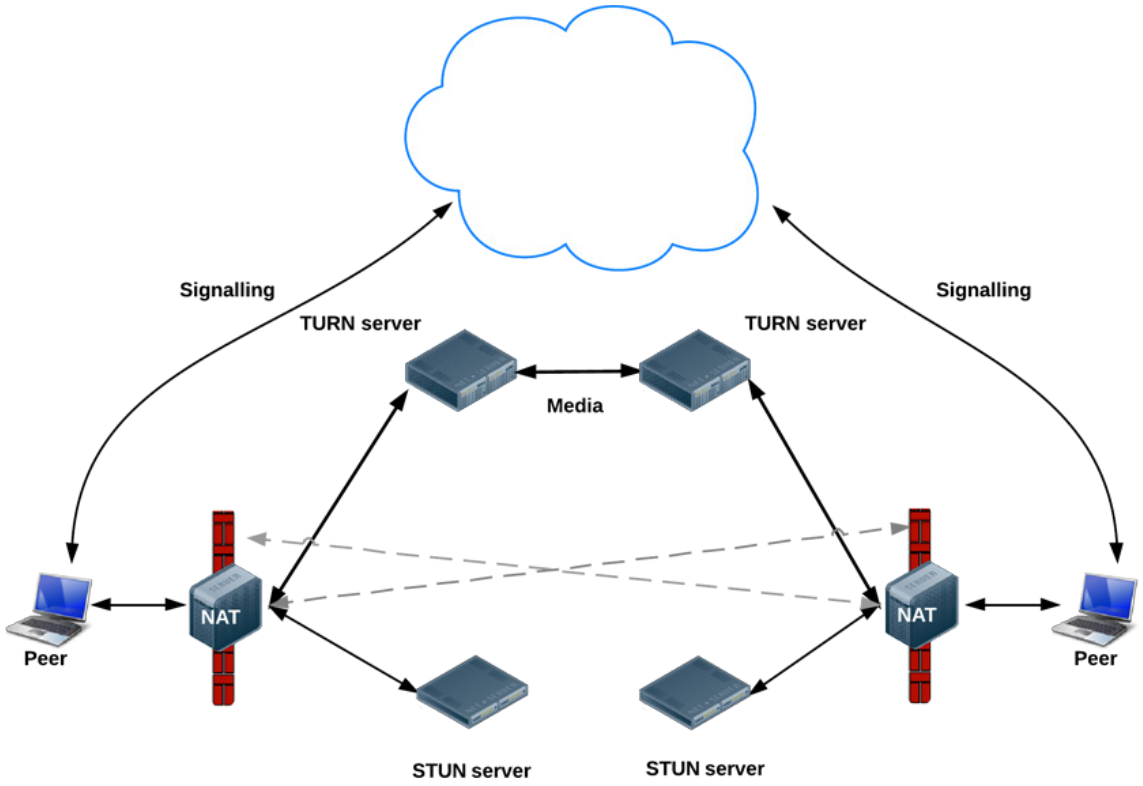


Figura 9 - Funcionamento TURN + STUN[55]

2.6 Sinalização

Apesar do conceito do WebRTC implicar comunicação direta entre clientes, é necessária a negociação de parâmetros de sessão, descritos via SDP, de forma a estabelecer uma ligação entre os clientes. Quer isto dizer que, para além de ser necessária a existência de servidores para alojar a aplicação, é necessário que este servidor possibilite a negociação da sessão, tradicionalmente denominada de sinalização.[66] Podemos verificar o papel da sinalização (*Signalling*) na comunicação WebRTC na Figura 10.

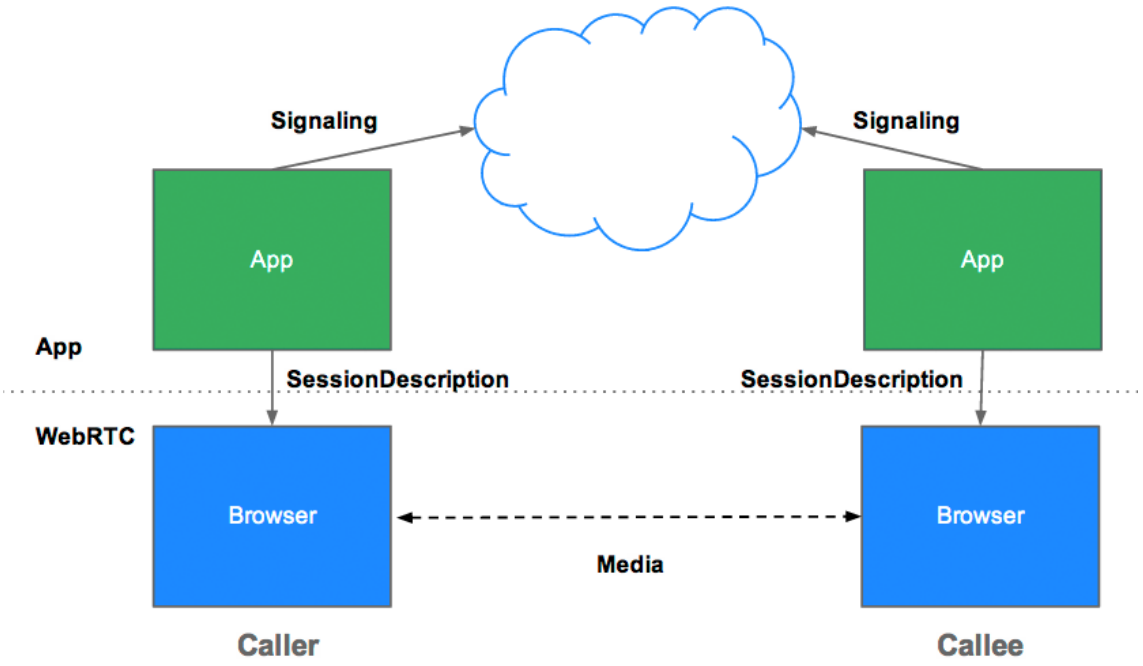


Figura 10- Sinalização em comunicação em WebRTC[67]

Conforme podemos verificar na Figura 10, a aplicação no servidor, necessita de fazer uma conexão tradicional com o outro parceiro, de forma a negociar os parâmetros de sessão. Esta comunicação é a Sinalização (Signalling).
A especificação do WebRTC deixa a cargo do programador a escolha do protocolo de sinalização a usar na sua aplicação. Segue a análise de alguns mecanismos de sinalização que podem ser utilizados para o estabelecimento de comunicação WebRTC. [49]

2.6.1 XMPP over WebSockets/Jingle

O protocolo Jingle define um protocolo XMPP (*Extensible Messaging and Presence Protocol*) para iniciar e gerir sessões multimédia ponto-a-ponto entre duas entidades XMPP (ver Figura 11) permitindo interoperabilidade com padrões atuais na internet. Esta interoperabilidade é permitida devido ao suporte do protocolo a diversos tipos de aplicação (vídeo, voz e dados) e a diversos métodos de transporte. Neste protocolo as mensagens trocadas entre pontos não são enviadas em

frames mas em *streams* XMPP. Este protocolo permite ainda negociar os parâmetros da ligação entre os dois pontos e com esta informação criar um túnel de média onde há o intercâmbio de *streams* entre dois utilizadores.[68][69]

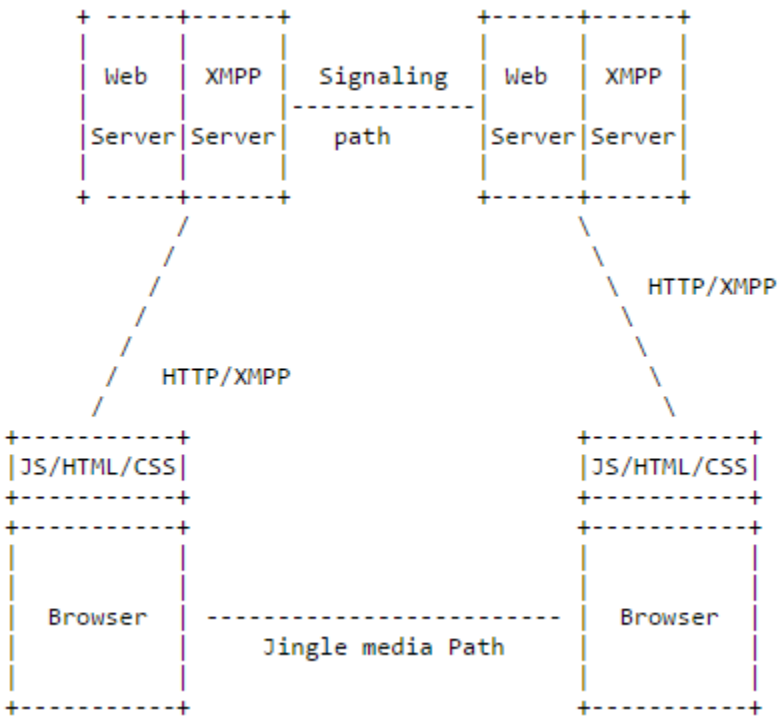


Figura 11 - Esquema funcionamento XMPP[64]

2.6.2 XHR(XMLHttpRequest) / Comet

Este método é diferente dos outros analisados até ao momento. Trata-se de um modelo aplicacional no qual um pedido HTTP é prolongado e assim permite ao servidor enviar informação para o cliente sem que este faça um pedido específico.[70] Ou seja, mantém aberta a conexão feita no pedido da página, para ser possível a transação de dados.

Este modelo vai contra o modelo clássico das aplicações web prolongando o tempo de resposta a pedidos, sendo que para isso faz uso de capacidades próprias dos navegadores. [70]

Para além de que este método obriga que o programador defina um método de sinalização, tem ainda um problema de escalabilidade uma vez que exige uma maior carga ao servidor, o que será problemático quando houverem muitos utilizadores.[70][71]

2.6.3 JSEP - JavaScript Session Establishment Protocol

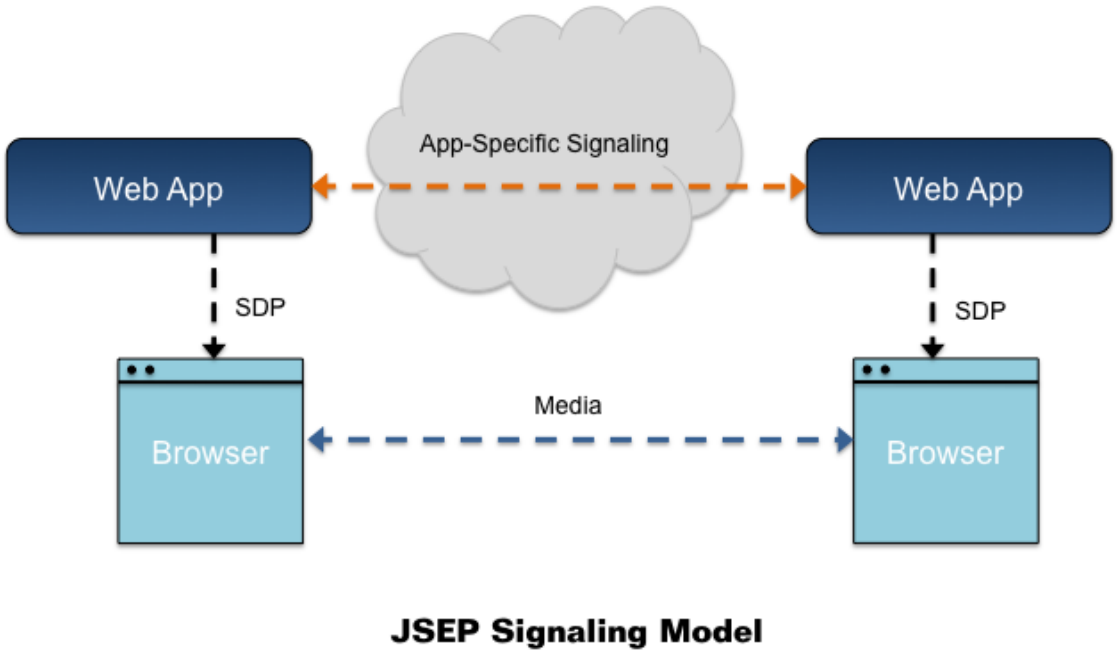


Figura 12 - Modelo de sinalização JSEP[72]

A ideia geral por trás da configuração das comunicações WebRTC, foi de especificar e controlar completamente a comunicação ao nível da multimédia mas, por outro lado, deixar a sinalização das comunicações a cargo das aplicações. Com base neste pensamento foram trabalhados alguns mecanismos que por um ou outro motivo não tiveram sucesso. Para resolver esta questão surge o JSEP (*JavaScript Session Establishment Protocol*). Este protocolo é responsável pela forma como os clientes recolhem informação acerca dos *codecs* utilizados e do tipo de multimédia suportado.[73]

De forma geral, o JSEP permite a criação de mensagens de oferta (*offers*) e de atendimento (*answers*) e define de que forma estas mensagens podem ser aplicadas (Ver Figura 12). Apesar disto, é importante ressalvar que este protocolo não define de que forma esta informação é trocada, quer isto dizer que fica a cargo do programador definir como a informação é enviada e recebida do servidor Web.[74]

2.6.4 WebSockets

O funcionamento clássico de aplicações Web é feito através de pedidos feitos pelo utilizador através do navegador e respostas enviadas pelo servidor. Quer isto dizer que para haver comunicação de dados, esta teria de ser feito através do pedido feito pelo utilizador e/ou da resposta do servidor. Com a evolução da Web, foi necessário alterar este paradigma, o que

acabou por acontecer com o aparecimento do AJAX (Asynchronous Javascript and XML) que permitiu tornar as aplicações Web mais dinâmicas sendo responsivas.[75]
No entanto, esta solução não conseguia dar resposta a todas as necessidades uma vez que, com o uso do AJAX (Asynchronous JavaScript and XML), a comunicação apenas é feita quando um utilizador aciona um evento ou através de temporizadores, o que não permite que exista uma transmissão bidirecional constante de dados. Uma solução existente para esta problemática era a sondagem longa, que criava uma ligação permanente até ao servidor, o que obrigava o servidor a ter várias ligações TCP para cada cliente.[76]
De forma a dar resposta às necessidades e evitar os problemas da sondagem longa, foi desenhado o protocolo *WebSockets*, baseado em *Sockets* e que faz o uso de HTTP. [76]
O protocolo de *WebSockets* é definido de forma simples na sua especificação "Para permitir aplicações Web com comunicação bidirecional com processos do servidor". Este protocolo possibilita então a existência de uma ligação permanente entre cliente e servidor, sendo assim possível que quer um quer outro possam iniciar o envio de dados a qualquer momento. Assim, o servidor e o cliente criam uma ligação permanente, na qual o cliente faz apenas um pedido e o servidor envia, através de vários pacotes de dados, toda a informação necessária, sem a necessidade de o cliente fazer qualquer outro pedido.[77]

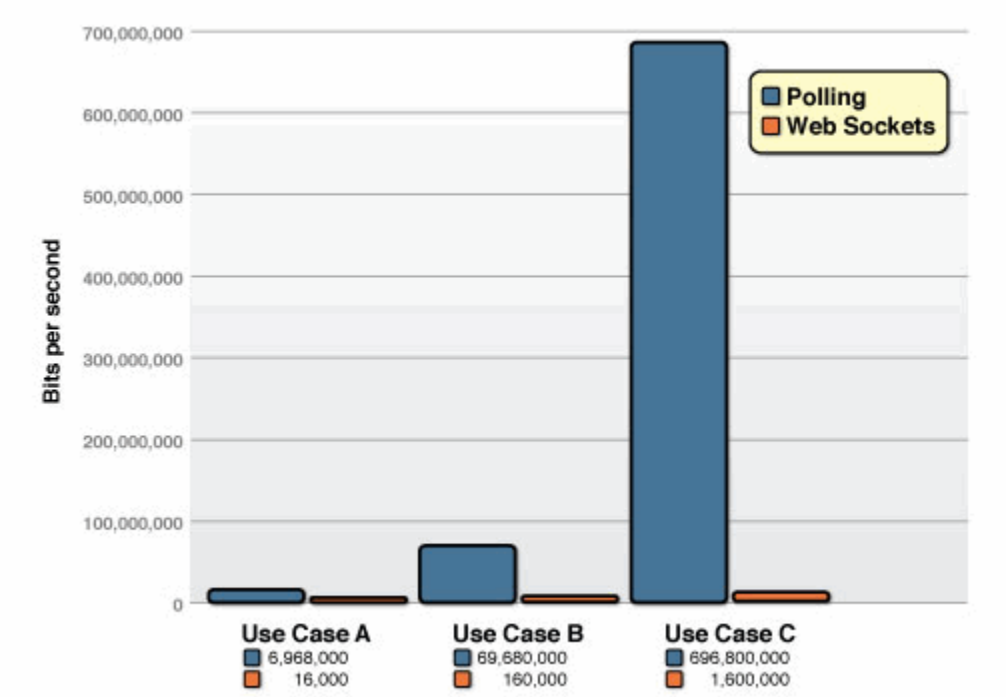


Figura 13 - Comparação entre comunicação desnecessária entre método polling e WebSocket[78]

Conforme indicado na Figura 13, a grande vantagem da utilização de *WebSockets* é a menor carga imposta na rede, tornando a transmissão de dados mais rápida, quando comparado com

aplicações que fazem uso de *pooling* (como aplicações que seguem o modelo Comet, apresentado anteriormente no sub-capítulo XHR(XMLHttpRequest) / Comet).[78][79]
Sendo que o objetivo base do WebRTC é a comunicação em tempo real na web, e considerando a comparação feita dos vários mecanismos de sinalização, torna-se óbvia a utilização que terá de ser feita destes *WebSockets*. Na prova de conceito apresentada, a utilização de *WebSockets* é feita com Node.JS, mais especificamente com o módulo Socket.IO, cujo funcionamento é explicado mais detalhadamente no subcapítulo a ele dedicado.[77, p. 64]

2.6.5 SIP (Session Initiation Protocol) over WebSockets

Antes de detalhar SIP *over WebSockets*, será importante introduzir o conceito do SIP. O SIP é um protocolo de controlo ao nível aplicacional que permite criar, modificar ou terminar sessões com um ou mais participantes. Estas sessões habitualmente tratam-se de chamadas *VoIP*, conferências multimédia ou difusão multimédia.[80]
De uma maneira simples, SIP *over Websockets* descreve a utilização de *WebSockets* para transporte de mensagens SIP entre cliente e servidor. Esta implementação está subdividida em duas definições, sendo que ambas comunicam por *WebSocket* SIP *subprotocol*.[81] [82]
SIP *WebSocket Client* – Que permite abrir ligações de saída para o servidor; [81]
SIP *WebSocket Server* – Que permite ao servidor aguardar e tratar ligações de entrada iniciadas pelo cliente. [81]
Resumidamente, no SIP *over WebSockets* é feito o transporte de mensagens SIP através de *WebSockets*.

2.7 Protocolos data channel

Como é óbvio, em WebRTC o foco da comunicação está no multimédia, no entanto é fácil perceber que pode também interessar e muito, a transmissão de dados não-multimédia. De forma simples, os dados multimédia são transmitidos através de SRTP (Secure Real-time Transport Protocol), enquanto os outros tipos de dados são transmitidos com SCTP encapsulado em DTLS.[83] Podemos verificar na Figura 14 onde se situa SCTP nos protocolos Data Channel.
De forma a ser possível a transmissão de dados não-multimédia em WebRTC, surgiram os canais de dados (Data Channels). Um Data Channel é uma conexão de grande performance e com poucos atrasos, trata-se de uma comunicação direta entre dois terminais (por comunicação direta, entenda-se: sem utilização do servidor da aplicação como retransmissor). Veremos neste capítulo os protocolos utilizados para este tipo de transmissões, evidenciados também na figura abaixo.[84][85]

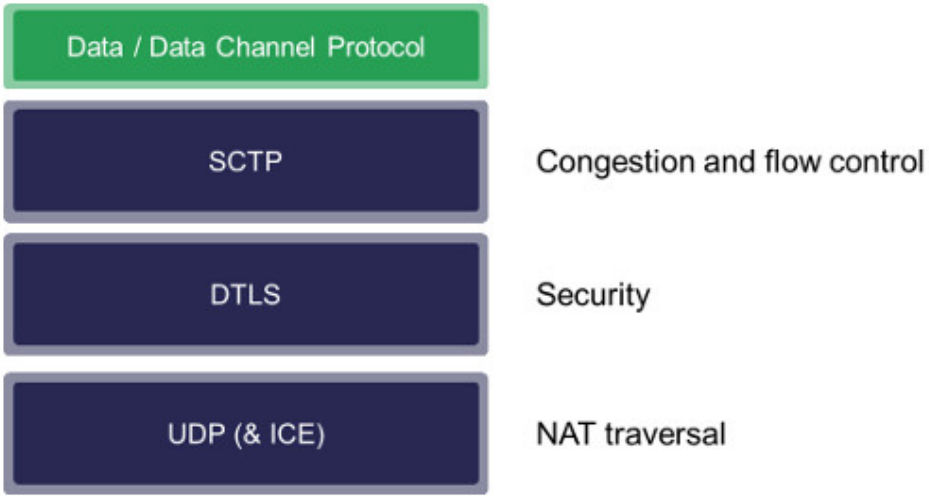


Figura 14 - Protocolos Data Channel com SCTP, DTLS e UDP[85]

2.7.1 SCTP - Stream Control Transmission Protocol

SCTP (Stream Control Transmission Protocol) é um protocolo de transporte fiável, a ser utilizado em redes não fiáveis, que não garantem entrega de pacotes. Este protocolo garante entrega livre de erros, a não entrega de dados duplicados, entrega sequencial de mensagens em múltiplos fluxos de dados, tolerância a falhas da rede e fragmentação de dados de acordo com o máximo definido. Oferece também alguma segurança uma vez que evita congestão de dados e ataques de flooding.[86][87]

Flooding trata-se de bombardear um switch com pacotes com MAC de destino distintos, forçando o switch a esgotar memória e começar a enviar os pacotes em broadcast, podendo assim os pacotes serem acedidos por outros que não o destinatário.[88]

Este protocolo tem um papel semelhante ao da UDP e do TCP. Na Figura 15 podemos verificar mais algumas características, assim como as diferenças quando comparado com UDP e TCP. [85]

	TCP	UDP	SCTP
Reliability	Reliable	Unreliable	Configurable
Delivery	Ordered	Unordered	Configurable
Transmission	Stream	Messages	Messages
Flow control	Yes	No	Yes
Congestion control	Yes	No	Yes

Figura 15 - Comparação TCP/UDP/SCTP[85]

2.7.2 UDP - User Datagram Protocol

UDP é um protocolo de comunicação definido para possibilitar a comutação de pacotes em modo de datagramas que fornece também um procedimento para que as mensagens enviadas contenham o mínimo possível de dados protocolares, para haver uma comunicação mais rápida.
Neste protocolo a entrega ordenada de dados, a retransmissão de comunicações falhadas e o controlo de congestão de rede não são garantidos só por si, deixando isso para o nível aplicacional. Apesar destas desvantagens, tudo isto faz com que seja possível uma entrega rápida dos pacotes de dados, o que é essencial para comunicações em tempo real.[89]

2.7.3 DTLS - Datagram Transport Layer Security

Devido ao aparecimento de protocolos ao nível aplicacional que utilizam transporte UDP, e sendo necessária a introdução de segurança neste tipo de comunicação, o protocolo DTLS surge como uma forma de implementar o protocolo TLS (o mais utilizado para implementar segurança em tráfego de redes) sobre os datagramas utilizados no transporte UDP. [90]
Assim, o DTLS é um protocolo utilizado para implementar segurança de tráfego na rede, sendo o seu objetivo permitir comunicação entre cliente e servidor, protegendo os dados comunicados de ser visualizados, alterados ou falsificados. Este protocolo é especificado deliberadamente com o objetivo de ser o mais similar possível ao TLS, de forma a minimizar a criação de um novo mecanismo de segurança, e maximizar a reutilização de código e infraestruturas necessárias.[90]

2.7.4 Secure real time protocol - Datagram Transport Layer Security

O RTP é um protocolo de comunicação que fornece transmissão de dados ponto-a-ponto, adequando a transmissão em tempo real dos dados. Este protocolo é habitualmente utilizado para comunicar dados áudio e vídeo, apesar de inicialmente ter sido especificado para conferências multimédia, é habitualmente utilizado para comunicação em tempo real de multimédia em transmissão de dados ponto-a-ponto. RTP não garante a entrega de pacotes nem qualidade de serviço, mas para colmatar isto, é capaz de fazer reconstrução temporal, detetar perdas e identificar conteúdo. Fornece também confidencialidade e autenticação de mensagens.[91][92]
Para completar o protocolo RTP, surge o SRTP que implementa mecanismos de segurança adicionais. Em termos de segurança o SRTP fornece encriptação e autenticação de mensagens, sendo que em termos de performance é capaz de conseguir transmitir uma grande quantidade de dados em diversas tipologias de rede.[93][94]
O uso de DTLS como extensão para estabelecer chaves para SRTP conjuga a performance e benefícios de encriptação do SRTP com a flexibilidade e conveniência do DTLS. Torna-se também necessário como mecanismo externo, uma vez que SRTP não permite a gestão de chaves nativas. [93]
Os pontos chaves desta combinação são: [93]

- Dados aplicacionais são protegidos com SRTP;
- É com DTLS que se estabelecem as chaves a utilizar, os algoritmos e parâmetros de SRTP;
- É utilizada uma extensão DTLS para negociar com algoritmos SRTP.

É esta a solução utilizada em WebRTC para encriptação de dados RTP e gestão de chaves (através do DTLS). De realçar que esta solução está especificada para casos com exatamente dois utilizadores.[95]

2.7.5 SDP - Session Description Protocol

SDP (Session Description Protocol) tal como o nome indica este protocolo descreve os parâmetros de inicialização de uma sessão criada para transmissão de dados multimédia. Este protocolo não faz transmissão de dados por si mesmo, é apenas usado para a negociação feita entre terminais aquando do início de uma sessão. Esta descrição torna-se essencial para toda a transmissão feita posteriormente, uma vez que terá de se ter em conta as características dos dados enviados.[96]
Tipicamente o SDP contém informação sobre o nome e objetivo da sessão, o tempo que a sessão tem como ativa, a multimédia transmitida na sessão e informação necessária sobre a rede, desde endereços IP até largura de banda. Um pacote SDP está estruturado de forma simples com uma descrição textual estruturada onde estão também contidos pares chave/valor como parâmetros.[97]
Este protocolo torna-se então útil para o WebRTC, na medida em que SDP pode ser utilizado para descrever a sessão, possibilitando assim a negociação de sessão. Esta negociação é baseada no mecanismo de oferta/resposta, sendo que na oferta é transmitida para o outro utilizador uma descrição sobre a multimédia a ser transmitida, os *codecs* suportados e descrição dos protocolos suportados. No caso da resposta, é enviada a mesma informação que na oferta, mas com dados relativos ao outro utilizador.
Em WebRTC esta informação está presente no objeto "*RTCSessionDescription*".[98]

2.8 NODE JS

Dadas as necessidades de implementação de um servidor que, para além da disponibilização das páginas Web permita também comunicação através de WebSocket, foi necessário recorrer à utilização de Node.JS, assim como aos seus módulos node-static e socket.io. O servidor da aplicação será um Node.JS, esse servidor é necessário para a disponibilização da aplicação (disponibilização dos ficheiros, recorrendo a node-static) e para construção de mecanismos de

comunicação através de WebSockets, que permitam a sinalização entre parceiros. Iremos então agora analisar essas tecnologias.
A necessidade de desenvolver aplicações de rede rápidas, fáceis e escaláveis levou à criação do Node.Js. Esta framework, que ainda não está num estado maduro, utiliza um modelo de input/output assíncrono, com ações originadas por eventos, e é este modelo que permite que seja leve e eficiente, sendo ideal para aplicações com bastante transmissão de dados e nas quais se pretende comunicação em tempo real, em sistemas distribuídos. O modelo do Node.Js é semelhante e inspirado por sistemas como "Ruby's Event Machine"[99], e "Python's Twisted"[100], sendo que no caso do Node.Js o modelo dirigido por eventos é levado mais longe.[101]
O Node.Js classifica-se assim como um sistema que permite criar, principalmente, aplicações do lado do servidor, em ambiente JavaScript. Fica ainda a nota que o Node.Js executa o código JavaScript utilizando o motor criado pela Google V8.[102]
A instalação e implementação desta tecnologia é ainda de bastante difícil configuração, no entanto, as vantagens da sua utilização justificam este trabalho extra.
Esta tecnologia integra ainda um gestor de módulos (Node Package Manager), e é este gestor que permite ao utilizador a instalação de módulos como os mencionados nos seguintes sub-capítulos, Socket.io e node-static.[103][104][105]

2.8.1 SOCKET.IO

O Socket.IO é uma biblioteca JavaScript que permite o desenvolvimento de aplicações web com capacidades de comunicação cliente/servidor bidirecional e em tempo real. Esta biblioteca está dividida em duas partes, uma API a ser utilizada do lado do servidor e outra API para o lado do cliente. Apesar de não ser considerado por alguns autores, trata-se de uma forma simples de *Websockets*.[106]
O Socket.IO é utilizado no desenvolvimento da prova de conceito apresentada, permitindo a comunicação bidirecional em tempo real, assim como a criação e utilização de salas para diferenciar por canais as transmissões de dados.[107]

2.8.2 NODE-STATIC

Este módulo trata-se de um simples servidor de ficheiros. Entre as suas características principais, relevo para o facto de seguir os *standards* definidos para a disponibilização de ficheiros por HTTP. Este módulo possui também um mecanismo de *caching*, que tem como objetivo acelerar o processo de disponibilização de ficheiros.[108]
É outro módulo utilizado na prova de conceito apresentada. É utilizado apenas para criar o servidor responsável pela disponibilização (por parte do servidor) dos ficheiros que compõem a aplicação.[109]

2.9 Sumário

São várias as tecnologias que possibilitam a existência da tecnologia WebRTC. Desde a mais elementar para o desenvolvimento Web, até à existência de mecanismos que permitam soluções de problemas inerentes à tipologia das redes. Neste capítulo são analisados de forma geral, todas as tecnologias necessárias à criação de uma aplicação WebRTC, de forma a tornarem-se mais percetível as suas características e limitações. Adicionalmente, é feita uma análise a alguns dos mecanismos de sinalização e também à tecnologia Node.JS, utilizada neste trabalho.

3 Prova Conceito

Segue-se a apresentação da aplicação funcional principal apresentada. Esta aplicação vem no seguimento do estudo feito e surge como um protótipo para uma solução que irá ser utilizada a nível empresarial, principalmente para comunicação com clientes e com consultores localizados nos clientes.

Este desenvolvimento surge como resultado de uma pesquisa aprofundada sobre a tecnologia e pretende demostrar a facilidade da utilização desta tecnologia, ao mesmo tempo que permite que seja verificado o seu funcionamento e se demostra a disponibilidade e facilidade de acesso que WebRTC permite.

De forma geral, o que esta aplicação permite é que um utilizador consiga fácil e rapidamente a comunicação através de áudio, vídeo, texto e transmissão de ficheiros. Para isto, basta ao utilizador aceder à página, selecionar a sala de comunicação onde pretende ingressar e permitir o acesso aos seus dispositivos. Não é necessária instalação de qualquer extensão ou software para permitir isto, basta a utilização de um navegador compatível, em qualquer plataforma.

3.1 Objetivo

Pretende-se com esta aplicação, demostrar as capacidades de uma aplicação básica que funcione com WebRTC. Assim, espera-se poder clarificar algumas dúvidas quanto ao funcionamento, fiabilidade, capacidade e vantagens da utilização desta nova tecnologia.

O desenvolvimento desta aplicação também pretende evidenciar as vantagens que esta tecnologia traz não só para utilizadores mas também para programadores, que mesmo sem recorrer a

soluções existentes, são capazes de desenvolver aplicações WebRTC sem que isso signifique que tenham de ter conhecimentos avançados sobre redes, *hardware* utilizado ou conhecimento específico do navegador.
Adicionalmente, e com o decorrer do projeto, considerou-se oportuno que fosse apresentada uma solução WebRTC que permitisse conversação em grupo e com total compatibilidade entre navegadores. Por forma a demostrar a simplicidade da utilização de soluções existentes no desenvolvimento de novas soluções, foi utilizado simpleWebRTC no desenvolvimento da aplicação. O facto de ser desenvolvida a partir de uma solução já existente, permite perceber que, dada a simplicidade, a reutilização de soluções existentes no desenvolvimento de aplicações mais complexas, pode ser um fator muito importante na proliferação desta nova tecnologia.
Durante o desenvolvimento da aplicação, foi decidido que serviria como protótipo para posterior desenvolvimento de aplicações a nível empresarial.

3.2 Seleção da implementação

Para o desenvolvimento da aplicação, foi necessário tomar algumas decisões de forma a definir o que seria feito, as capacidades da aplicação, entre outros. Apesar dessa tomada de decisões, posteriormente, com a evolução da aplicação e com as primeiras utilizações, foram detetados novos requisitos.
Devido à especificação estândar do WebRTC ainda não estar finalizada, existem ainda diversas diferenças nas implementações dos navegadores, daí, e conforme o que era pretendido da aplicação, foi necessário definir à partida qual seria o navegador sobre o qual iria incidir o foco de compatibilidade. Isto é, foi necessário definir que navegador teria obrigatoriamente de ser compatível com todas as funcionalidades a aplicação. Devido às origens da tecnologia e ao esforço evidente na compatibilidade das funcionalidades WebRTC, o navegador escolhido foi o Chrome do Google.
Uma das mais importantes decisões técnicas quanto ao desenvolvimento da aplicação seria a escolha da metodologia a utilizar para permitir a sinalização necessária. Devido aos argumentos já aqui apresentados em sua defesa, e sendo considerada pessoalmente como a melhor opção, foi escolhido o método de sinalização através de *websockets*, implementado através de Node.Js e socket.io.
Quanto às funcionalidades, foi decidido que seria imprescindível a existência de comunicação entre terminais com o mesmo navegador, na mesma rede local. Esta restrição à rede local está diretamente relacionado com os problemas de comunicação de WebRTC através de NAT e Firewall. Essa comunicação teria então de permitir a transmissão vídeo e áudio entre terminais, adicionalmente seria também interessante conversação em texto.
De modo a existir alguma organização, e multiplicar as utilizações da aplicação, foi implementado um sistema de salas de conversação, tradicional dos *WebSockets* e do WebRTC.

Este sistema permite que existam salas de conversação identificadas por um nome único, onde é feita tal comunicação. Na aplicação principal a sala é considerada cheia quando existem já dois parceiros, não sendo possível a entrada de mais parceiros. No caso da aplicação multi-parceiro, apesar de a partir dos oito utilizadores começar a serem notórios problemas de performance do navegador (excessiva carga de CPU), não foi definido limite no número de utilizadores, uma vez que a funcionalidade do simpleWebRTC poderá a ter melhorias neste sentido e porque a aplicação multi-parceiro faz sempre uso da versão mais recente do simpleWebRTC.
Posteriormente, e por sugestão de diversos utilizadores na fase de testes, foi implementada funcionalidade para permitir partilha de ficheiros. A forma de comunicação dos ficheiros é em tudo semelhante à comunicação da conversação em texto. Apenas é necessária a diferenciação no caso dos ficheiros uma vez que, devido ao seu tamanho, é necessária a sua segmentação durante o envio, sendo que esses segmentos são agrupados no destino. Esta funcionalidade será descrita em maior detalhe posteriormente.
Conforme já foi mencionado, seria também interessante a possibilidade de existirem conversações multipartido. Como tal, foi decidido que seria feita uma aplicação à parte, que possibilitasse esse tipo de conversações. Para o caso da aplicação multi-parceiro desenvolvida posteriormente, foi escolhida a solução existente simpleWebRTC, por se tratar uma das mais populares, mais testadas e também pela simplicidade que a caracteriza.

3.3 Desenvolvimento

Após a definição do que seria a aplicação e de modo a serem cumpridos os objetivos, foi necessário dar início ao desenvolvimento da aplicação que resultaria do estudo feito ao longo deste trabalho. Para conseguir o funcionamento da aplicação definida foi necessário seguir vários passos de forma a garantir o funcionamento e as capacidades desejadas na aplicação. Segue-se uma explicação dos passos dados, assim como a justificação dos mesmos.
Dado o objetivo principal da aplicação ser uma aplicação de vídeo chamada, seria obrigatório começar o desenvolvimento pela obtenção de multimédia a ser transmitida posteriormente entre os parceiros. Para isso, foi criada uma página Web simples, na qual era apresentado um elemento vídeo, sendo esse vídeo obtido a partir do hardware do terminal, devidamente autorizado pelo utilizador. Assim, foi invocado, através de JavaScript, o método getUserMedia, com as *constraints* a requisitar vídeo e áudio, sendo que posteriormente, esses dados media obtidos foram transformados num objeto URL e inseridos como *source* (fonte) no elemento vídeo da página. Na Figura 16 podemos verificar o funcionamento desta página, assim como podemos também verificar o código no anexo Código Fase 1 da Aplicação.

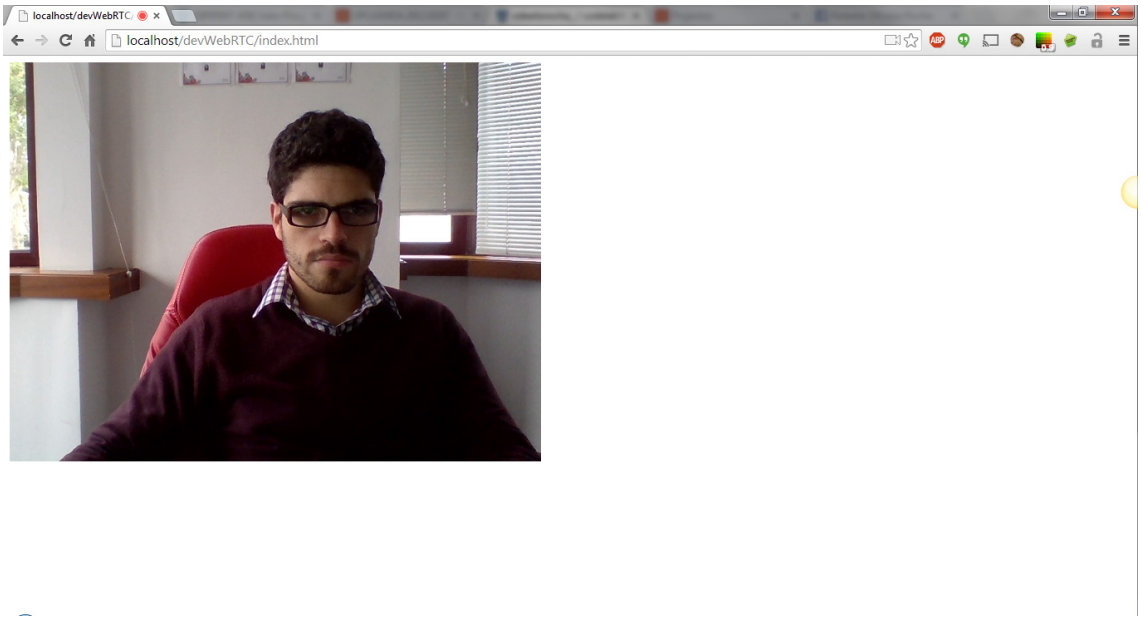


Figura 16 - Obtenção de vídeo do utilizador

Fica a nota que, em todos os passos seguintes é feita a utilização de um ficheiro denominado "adapter.js", mantido pela Google com o intuito de abstrair o utilizador da tecnologia das diferenças entre implementação dos vários navegadores e alterações à especificação [110].
Posteriormente, e dado que já foram obtidos os dados multimédia, foi interessante introduzir o WebRTC, sendo que para isso foi introduzida a utilização da PeerConnection. O Objetivo neste passo seria fazer a comunicação através da PeerConnection, dentro da mesma página. Ou seja, não fazer qualquer transmissão na rede, mas apenas utilizar a PeerConnection como meio de transmissão do vídeo obtido entre elementos da página, sendo que para isto seriam já utilizadas as *SessionDescription* assim como os candidatos *ICE*. Desta forma, nesta página são apresentados dois elementos de vídeo, um que apresenta diretamente o que foi obtido com o método getUserMedia, e outro elemento, à direita, que apresenta os dados da stream remota. Isto é conseguido seguindo uma série de passos iniciados obviamente, tal como na aplicação anterior, pela obtenção da multimédia do utilizador (getUserMedia) onde é também definida a função a ser executada em caso de sucesso e a função executada em caso de erro de obtenção. Assim sendo, ao ser permitido o acesso aos dispositivos, é iniciada a função de sucesso que irá iniciar a chamada, para isso irá colocar o vídeo obtido no local do vídeo local, iniciar a PeerConnection local e associar a ela a função a lançar no evento de receção de candidato ICE e posteriormente inicia também a PeerConnection remota (dado ser tudo feito na mesma página), associando também uma função para o evento de receção de candidato ICE. Imediatamente após isto, é adicionada a stream local à PeerConnection local e sobre essa PeerConnection é criada uma oferta e ao criar essa oferta é incluída a descrição local (em formato SDP) na PeerConnection local como descrição local e na PeerConnection remota como descrição remota. De seguida é repetido o processo mas para o caso da PeerConnection remota, sendo que apenas difere no facto

de ser criada uma resposta e não uma oferta. Adicionalmente, entretanto, foi associada à PeerConnection remota, um evento para tratar as streams recebidas, sendo que esse tratamento passa por apresentar o vídeo no local definido para o vídeo remoto. Conforme já foi indicado, foram definidos métodos para gerir a receção de candidatos ICE, sendo que nesses métodos o que é feito é adicionar na PeerConnection o candidato de ligação ao parceiro. Falta apenas referir que, ao sair da aplicação, é desligada a chamada fechando ambas as PeerConnection. Para perceber melhor o funcionamento desta fase da aplicação apresentada na Figura 17, poderá consultar o anexo Código Fase 2 da Aplicação.

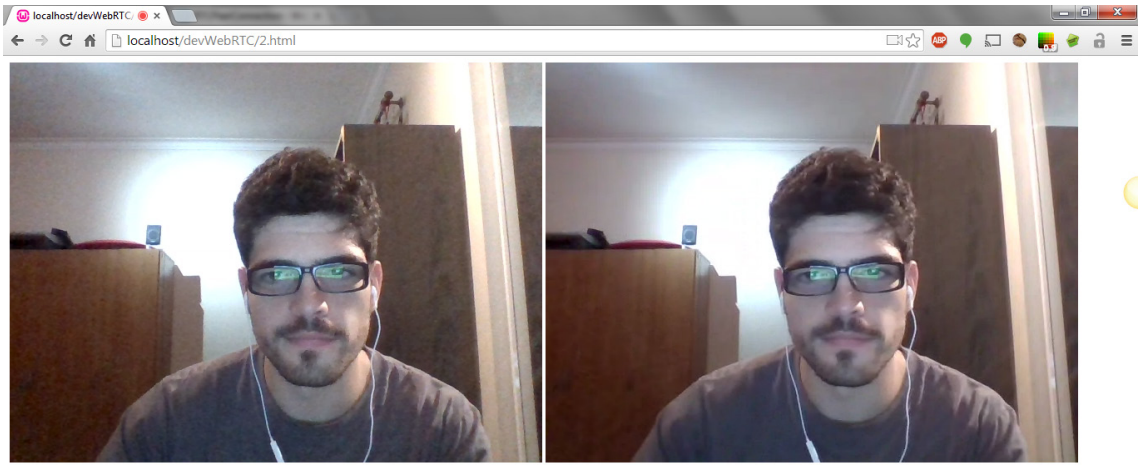


Figura 17 - Transmissão via PeerConnection

Numa terceira fase, e ainda com comunicação restrita à mesma página do navegador, foi introduzida a possibilidade de conversação por texto ao utilizador, sendo para isso necessária a utilização de DataChannel. Nesta fase, adicionaram-se à página duas caixas de texto, uma para envio e outra para receção de texto, adicionou-se também um botão para permitir o envio desse texto. Em termos de programação, foi alterada a função de iniciar chamada de forma a criar um DataChannel na PeerConnection local, e foi associada à PeerConnection remota a função para gerir a receção do DataChannel. Nessa função foi definido que as mensagens recebidas serão colocadas na caixa de texto de receção de texto. Ao botão de envio de texto, foi associada uma função que envia através do DataChannel da PeerConnection local os dados presentes na caixa de texto de envio. Foi também alterada a função de desligar de modo a fechar os DataChannel antes de fechar as PeerConnection. Este passo pode ser verificado na *Figura 18*, assim como tecnicamente no anexo Código Fase 3 da Aplicação.

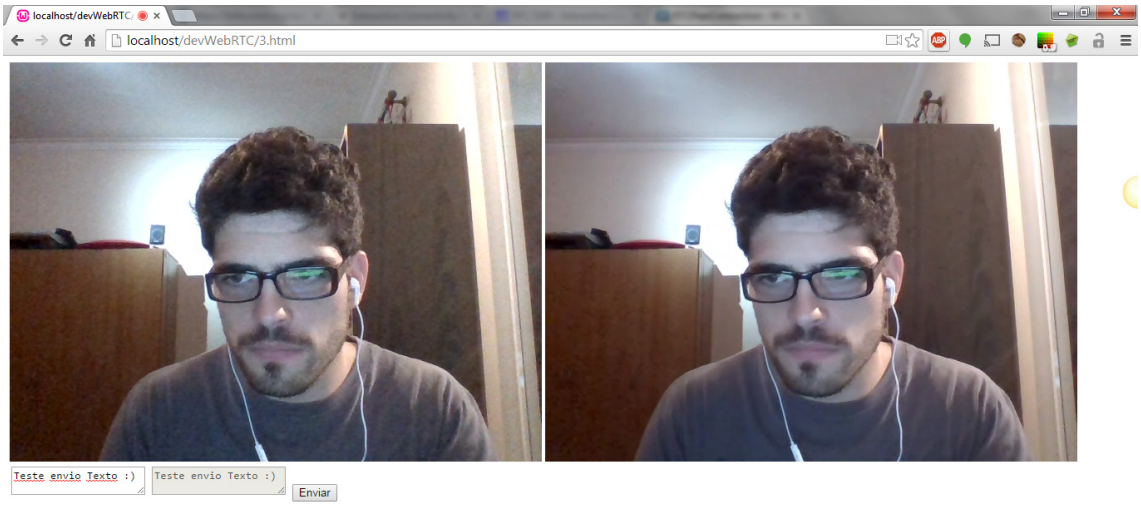


Figura 18 - Envio de texto

Posteriormente, seria necessário que todo este funcionamento fosse implementado de forma a permitir a comunicação através de rede, ao invés de apenas comunicar dentro da mesma página. Segue então a explicação das alterações necessárias para esta fase.

Todo este processo, o envio e receção de candidatos ICE, as ofertas e suas respostas, fazem parte do processo de sinalização (Signalling), necessário à comunicação WebRTC. Como já foi dito anteriormente, é necessária a utilização de servidores, não só para alojar a aplicação, mas também para permitir este processo de sinalização. Basicamente o que é feito nesta fase é uma troca de metadados através do servidor da aplicação entre os parceiros, de forma a definir toda a configuração da comunicação. Para isto, para além de todos os métodos já implementados é necessária a existência de um servidor de sinalização que permita a troca de mensagens entre os parceiros. E se até agora bastava um servidor Web comum para a apresentação da página, é nesta fase que se torna necessária a existência de um servidor com Node.JS, de modo a criar o servidor de sinalização, que permitirá a comunicação através de WebSockets. O funcionamento deste servidor terá dois focos, funcionar como intermediário na comunicação passando as mensagens recebidas de um parceiro para o outro, e fazendo a gestão de salas de comunicação. No desenvolvimento desta aplicação, e devido à necessidade de utilizar Node.JS para a comunicação através de WebSockets, foi configurado e utilizado um servidor local.

As salas de comunicação nos WebSockets existem apenas para se fazer uma separação da comunicação feita através do servidor. Isto é, o objetivo das salas de comunicação é que a comunicação seja feita apenas entre parceiros presentes na mesma sala, passando apenas as mensagens recebidas de um parceiro para os parceiros da mesma sala. Adicionalmente, é utilizado este mecanismo de gestão de salas para limitar o acesso à sala a dois utilizadores como máximo. Como podemos verificar no anexo Código Servidor da Aplicação, o funcionamento do servidor é muito simples, e como já dito é feita a transmissão de mensagens recebidas no WebSocket para os parceiros da mesma sala, e é feito o controlo de número de utilizadores a apenas dois, sendo que caso exista um utilizador que pretenda entrar numa sala já com dois

utilizadores, será informado que a sala está cheia (esta parte deve ser gerida pela aplicação e não pelo servidor).
Do lado da aplicação, de forma a ser possível a comunicação, foram efetuadas modificações de forma a ser pedido ao utilizador o nome da sala que deseja entrar e, ao estar esse nome definido, ser feita a conexão ao WebSocket do servidor. Para organizar melhor a aplicação em termos de código, foi criado um ficheiro JavaScript à parte, referenciado na página, que contém todas as funções necessárias à comunicação já explicadas anteriormente, assim como algumas funções novas, explicadas a seguir.
Para alem da iniciação da conexão ao WebSocket do servidor, foram definidas todas as ações de resposta a eventos (mensagens) do WebSocket, tal como:

- WebSocket informa que o utilizador é criador da sala.
- WebSocket informa que a sala está cheia:
    - O utilizador é informado pela aplicação que a sala está cheia.
- WebSocket informa que o parceiro entrou na sala:
    - Define-se a comunicação como pronta.
- WebSocket informa que entrou outro parceiro:
    - Define-se a comunicação como pronta.

Foram também definidos os servidores STUN e TURN a ser utilizados, assim como as constraints da PeerConnection. No caso dos servidores STUN, foi definido que seriam utilizados servidores STUN disponibilizados pela Google para o efeito, presentes na internet [63]. Enquanto que no caso dos servidores TURN foram obtidos alguns servidores a partir do serviço *computeengineondemand*, presente na *appspot* [111]. Este serviço, entre outras coisas, fornecesse endereços de servidores TURN funcionais e que podem ser utilizados[112]. No entanto, a utilização do serviço *computeengineondemand*, não garante sucesso uma vez que o navegador pode impedir a navegação à pagina do serviço, sendo que para isso foram também definidos (estaticamente), por prevenção, alguns endereços de servidores TURN obtidos anteriormente do mesmo serviço. Estes servidores, tanto STUN como TURN, são então adicionados como servidores Ice à PeerConnection, de forma a poderem ser utilizados quando necessário.
Posteriormente, e no decorrer dos testes e da utilização da aplicação, foram feitas algumas melhorias em termos de apresentação e utilização, focando-se na organização dos elementos na página, melhor fornecimento de informações ao utilizador, e funcionalidades de envio e receção de ficheiros. O envio/receção de ficheiros será explicado em maior detalhe no sub-capítulo Funcionamento técnico.

3.4 Funcionamento

Neste sub-capítulo será explicada a forma de como um utilizador pode utilizar a aplicação, mas também a forma como esta funciona a nível técnico, assim como as suas funcionalidades. É importante referir que não será apresentada a solução na íntegra, uma vez que esta aplicação irá ser utilizada como base para desenvolvimento de novas aplicações a nível empresarial.

3.4.1 Funcionamento pelo utilizador

Ao abrir a aplicação, o utilizador é de imediato questionado sobre qual a sala que deseja entrar, ou seja, em que sala pretende iniciar a comunicação como demostrado na Figura 19.

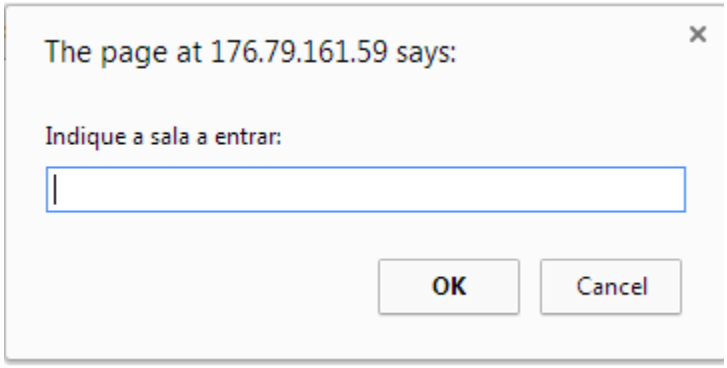


Figura 19 - Questão sobre qual sala entrar

Esta janela pode no entanto não aparecer caso a sala já esteja indicada no URL como parâmetro.

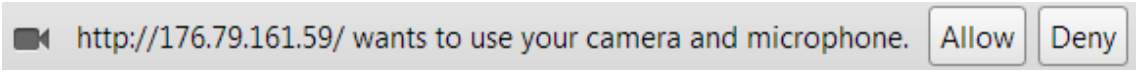


Figura 20 - Pedido de acesso aos dispositivos

Após a entrada na sala, e sendo obrigatório que seja permitido o acesso aos dispositivos, (pedido demostrado na Figura 20) pode acontecer uma de três coisas:
1 - A sala não contém parceiros, então é criada e surge o ecrã a indicar que não se encontra em conexão com ninguém, permitindo a mudança de sala. Esta mensagem (Figura 21) irá desaparecer assim que um parceiro se conectar na mesma sala consigo. Nesse caso será apresentada a conversação, conforme apresentada posteriormente.

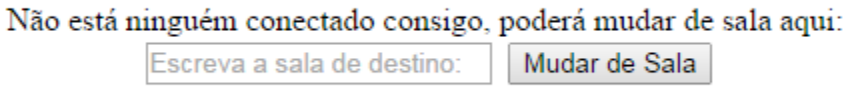


Figura 21 - Aviso de inexistência de conexão

2 - A sala contém um parceiro e por isso é iniciada imediatamente a conversação, conforme será apresentada posteriormente na Figura 24.

3 - A sala contém dois parceiros conectados e por isso está cheia, repete-se o caso 1 em que é apresentada a mensagem que o utilizado não se encontra conectado com ninguém.
Ao ser iniciada a conversação, é apresentada uma página onde é possível a comunicação via áudio, onde são apresentadas as camaras, do parceiro do lado esquerdo e própria do lado direito. Sobre a visualização do parceiro é possível parar ou continuar a visualização assim como tirar o som ou voltar a pôr. No topo da página é apresentada a possibilidade de alterar a sala e na parte inferior surge a secção de conversa por texto. Nessa secção é apresentada uma caixa, onde é mostrada a conversa atual, sendo que o que foi dito pelo parceiro é visível imediatamente por baixo da sua imagem assim como acontece com o texto enviado e a visualização local. Para permitir enviar mensagens surge uma caixa de texto onde pode ser escrito o texto acompanhado de um botão de enviar. Também é visível um ícone de ajuda a indicar que o envio de ficheiros pode ser feito através de *drag-and-drop*, quer isto dizer que para enviar um ficheiro basta arrasta-lo para a caixa de conversação (na aplicação surge a mensagem indicada na Figura 22).

Figura 22 - Instruções para envio de ficheiros

A receção dos ficheiros tem de ser sempre validada pelo utilizador que recebe o ficheiro, como demostrado na Figura 23.

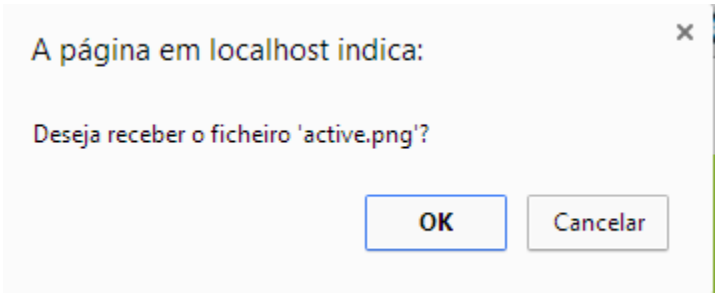


Figura 23 - Confirmação da receção do ficheiro

Para finalizar a chamada, basta sair da página, sendo o parceiro sempre informado que a chamada foi desligada.

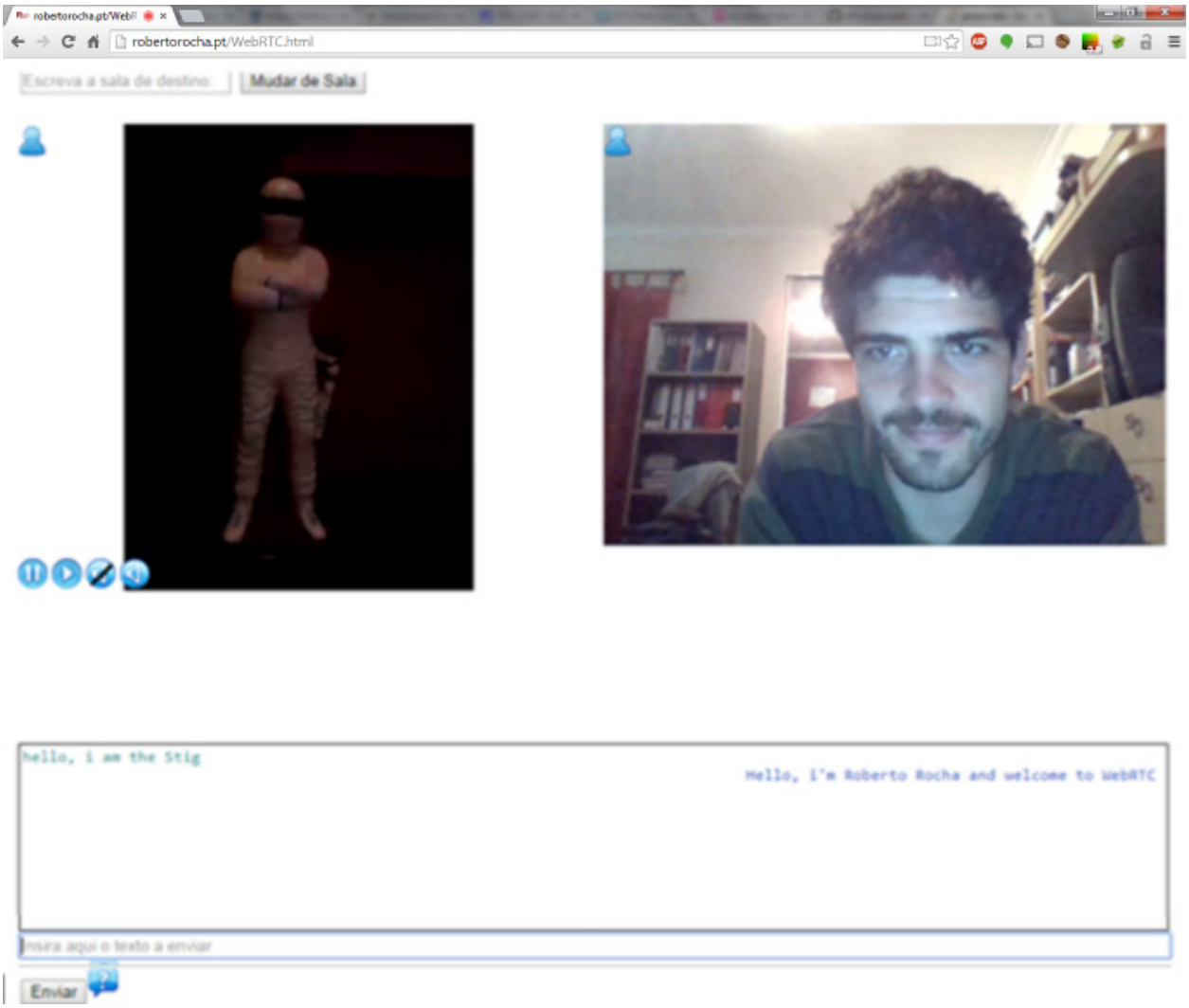


Figura 24 - Demostração de chamada em funcionamento

3.4.2 Funcionamento técnico

De modo a perceber um pouco melhor o funcionamento da aplicação, da comunicação WebRTC, de todos os mecanismos necessários para a comunicação e dos passos a dar, vamos analisar o funcionamento da aplicação, a nível técnico. Ou seja, iremos seguir o que acontece, em termos técnicos, durante a utilização da aplicação.

Assim que um utilizador entra na aplicação, é iniciado todo o processo que irá permitir a comunicação. Inicialmente, e após o utilizador indicar qual a sala que deseja entrar, o *WebSocket* do servidor (socket.io em Node.Js) recebe o evento acerca do pedido do cliente para criar ou juntar-se a uma sala. Isto é, recebe mensagem a indicar que existe um utilizador a pedir para entrar na sala X. Posteriormente o servidor verifica, quantos clientes existem nessa sala, podendo acontecer (função na Figura 25):

- A sala está vazia, e o utilizador é inserido nessa sala. Imediatamente essa inserção é comunicada ao utilizador;
- A sala contém apenas um parceiro e o utilizador junta-se a essa sala, sendo essa informação transmitida de imediato ao utilizador;
- A sala contém mais que um parceiro e por isso está cheia. É comunicado ao utilizador que a sala que pretende aceder está cheia.

```
socket.on('criar ou juntar', function (sala) {
    keepsala = sala;
    var numClientes = io.sockets.clients(sala).length;

    log('Sala ' + sala + ' tem ' + numClientes + ' cliente(s)');
    log('Pedido para criar ou entrar em sala', sala);

    if (numClientes == 0){ //se não tem ninguém, cria sala
        socket.join(sala);
        socket.emit('criada', sala);
    } else if (numClientes == 1) { //se existe um parceiro na sala, entra na sala criada
        io.sockets.in(sala).emit('entrar', sala);
        socket.join(sala);
        socket.emit('entrou', sala);
    } else { // dado que no máximo só podem haver dois parceiros por sala, se existirem dois parceiros na sala, a sala está cheia
        socket.emit('cheia', sala);
    }
    socket.emit('emit(): cliente ' + socket.id + ' entrou na sala sala ' + sala);
    socket.broadcast.emit('broadcast(): cliente ' + socket.id + ' entrou na sala ' + sala);

});
```

Figura 25 - Ação do servidor *WebSocket* ao receber mensagem 'criar ou juntar'

Imediatamente após este passo, é pedido acesso à câmara e ao microfone do utilizador, sendo essa permissão obrigatória para poder existir comunicação na aplicação. Este pedido é feito com as *constraints* definidas com áudio e vídeo a verdadeiro (Figura 26), ou seja, para obter áudio e vídeo do utilizador. Na função getUserMedia é também especificada a função a executar em caso de sucesso e a função a executar em caso de erro (Figura 26).

```
var constraints = {video: true, audio :true}; //constrains da conexão WebRTC

getUserMedia(constraints, handleUserMedia, handleUserMediaError); // GET USER MEDIA!!!
```

Figura 26 - *Contraints* WebRTC e getUserMedia

Em caso de sucesso o que ocorre é o seguinte (função na Figura 27): a *stream* obtida localmente é inserida como *source* (fonte) no elemento vídeo criado com o propósito de apresentar o vídeo local (função attachMediaStream) e posteriormente é emitida via *socket* a mensagem para o parceiro do tipo "*got user media*", para assinalar que foi obtido o *stream* local.

```
function handleUserMedia(stream) {
  localStream = stream;
  attachMediaStream(VideoLocal, stream);
  console.log('A mostrar a camara local.');
  sendMessage('got user media');
  if (isInitiator) {
    maybeStart();
  }
}
```

Figura 27- Ação no sucesso de getUserMedia

O parceiro ao receber esta mensagem pela primeira vez encarrega-se de invocar a função *maybeStart*, também invocada entretanto pelo primeiro utilizador. Nessa função é criada a *PeerConnection*, onde é imediatamente adicionada a *stream* local. Neste momento é também criado o *DataChannel* pelo criador da sala (o parceiro que entrou quando a sala estava vazia), sendo que se não for o criador da sala, o parceiro irá receber a configuração desse *DataChannel*.
Ao ser criada a *PeerConnection*, são também definidos os eventos para gerir a adição e remoção de *streams* e ao ser criado o *DataChannel* são definidos os eventos a ser executados na abertura e no fecho dos mesmos.
Adicionalmente e caso seja o criador da sala, na mesma função (maybeStart) é feita a chamada para o parceiro. Ao fazer a chamada o que é realmente feito é a criação de uma oferta e o seu envio para o parceiro através da *PeerConnection*.
Ao ser recebida a oferta, é adicionada à *PeerConnection* a descrição de sessão remota necessária e é criada a resposta também na *PeerConnection*.
Quando o outro parceiro recebe a resposta, define também a descrição de sessão remota na sua *PeerConnection*.
Ao ser criada a oferta ou a resposta, é definida também, na *PeerConnection*, a descrição de sessão local, entretanto enviada para ser definida como descrição remota pelo outro parceiro.
Neste momento é iniciada a comunicação WebRTC, sendo que quando um parceiro recebe a *stream* remota irá defini-la como *source* (fonte) do elemento vídeo criado com o propósito de apresentar o vídeo remoto.
O envio de texto ou ficheiros na conversa é feito através do método *send* do *dataChannel* conforme ilustrado na Figura 28. O envio e a receção são feitos de forma distinta caso seja ficheiro ou texto a ser enviado.

```
function sendData() { // enviar texto no chat
  var data = EnvioTexto.value;
  EnviarTexto.value += EnvioTexto.value + '\n';
  TextoRecebido.value += '\n';
  EnvioTexto.value = "";
  sendChannel.send(data);
  TextoRecebido.scrollTop = TextoRecebido.scrollHeight;
  EnviarTexto.scrollTop = EnviarTexto.scrollHeight;
}
```

Figura 28 - Função de envio de texto

Para o caso da conversação em texto o funcionamento é simples: quando o utilizador faz "Enviar" é obtido o texto e é enviado através do *DataChannel* já configurado anteriormente através do método *send*. No outro terminal é acionado um evento, que por sua vez chama uma função onde é extraída a mensagem e o texto recebido é colocado no local da conversa do parceiro remoto, conforme podemos ver na função da Figura 29.

```
else{//receber texto
      trace('Received message: ' + event.data);
      TextoRecebido.value += event.data + '\n';
      EnviarTexto.value += '\n';
      TextoRecebido.scrollTop = TextoRecebido.scrollHeight;
      EnviarTexto.scrollTop = EnviarTexto.scrollHeight;
```

Figura 29 - Receção do texto

No caso do envio de ficheiros (Ver função na Figura 30), o funcionamento é em tudo igual, diferindo a preparação. Assim, para enviar ficheiros e após o utilizador ter arrastado o ficheiro até a janela de conversação, é lido o ficheiro através de um *FileReader* (objeto *jquery* para leitura de ficheiros). Após o ficheiro ser lido é verificado o seu tamanho e, dada a limitação do *DataChannel*[113], caso o seu tamanho exceda 500 bytes (a limitação real varia entre 900 bytes e 1300 bytes, mas a utilização de valores mais altos que 500 bytes provou ser instável), o ficheiro é segmentado e enviado em partes para o parceiro. O ficheiro é enviado como objeto transformado em string (através de *Stringify*, método do *JSON*). Nesse objeto são ainda adicionados dados a indicar se é o primeiro envio, se é o ultimo envio e o nome do ficheiro.

```
var readFile = function(e, resultado) {
    if (e) resultado = e.target.result;
    var data = {};
    if (resultado.length > 500) {
        data.message = resultado.slice(0, 500);
        data.filename = filename;
        if (e){
            data.firsttime = true;
            EnviarTexto.value += "A enviar o ficheiro '" + filename + "'.\n";
            TextoRecebido.value += '\n';
            TextoRecebido.scrollTop = TextoRecebido.scrollHeight;
            EnviarTexto.scrollTop = EnviarTexto.scrollHeight;
        }
        else
        {
            data.firsttime = false;
        }
    } else {
        data.message = resultado;
        data.last = true;
        data.filename = filename;
        data.firsttime = false;
    }
    sendChannel.send(JSON.stringify(data));
    var remainingDataURL = resultado.slice(data.message.length);
    if (remainingDataURL.length) setTimeout(function () {
        readFile(null, remainingDataURL); // enviar nova parte
    }, 500)
}
```

Figura 30 - Função de envio de ficheiro

A receção do ficheiro é tratada pela mesma função que trata a receção de mensagens de texto, por isso torna-se necessário identificar se os dados recebidos têm ou não a estrutura definida no envio. Caso não tenha, é feito o tratamento já descrito de receção de texto. Caso tenha essa estrutura, é perguntado ao utilizador se deseja guardar o ficheiro que está a ser enviado (o ficheiro é identificado pelo nome) e caso seja aceite, o ficheiro irá ser recebido e posteriormente guardado no disco. Dada a possibilidade de o ficheiro ser recebido em partes, a função irá agrupar os dados recebidos até receber o atributo a indicar que se trata da última parte do ficheiro. Nesse momento o ficheiro é guardado no disco. (Ver código *JavaScript* na Figura 31)

```
if(verificaEstrutura(event.data))// Recebe ficheiros
{
    var eventdata = JSON.parse(event.data);
    if(eventdata.firsttime)
    {
        var r = confirm("Deseja receber o ficheiro '" + eventdata.filename + "'?");
        if (r == true) {
            receivefile = true;
            TextoRecebido.value += "A receber o ficheiro '" + eventdata.filename + "'.\n";
            EnviarTexto.value += '\n';
            TextoRecebido.scrollTop = TextoRecebido.scrollHeight;
            EnviarTexto.scrollTop = EnviarTexto.scrollHeight;
        } else {
            receivefile = false;
            TextoRecebido.value += "Cancelado o ficheiro '" + eventdata.filename + "'.\n";
            rejectedfilename = eventdata.filename;
            EnviarTexto.value += '\n';
            TextoRecebido.scrollTop = TextoRecebido.scrollHeight;
            EnviarTexto.scrollTop = EnviarTexto.scrollHeight;
        }
    }
    if(receivefile && eventdata.filename != rejectedfilename){//fim da receção do ficheiro e guardar o arquivo no disco
        arrayToStoreChunks.push(eventdata.message);
        if (eventdata.last) {
            guardaEmDisco(arrayToStoreChunks.join(''), eventdata.filename);
            arrayToStoreChunks = []; // resetting array
            TextoRecebido.value += eventdata.filename + ' recebido.' + '\n';
            TextoRecebido.scrollTop = TextoRecebido.scrollHeight;
            EnviarTexto.scrollTop = EnviarTexto.scrollHeight;
        }
    }
}
```

Figura 31 - Função de receção de ficheiros

Ao fechar a janela ou ao mudar de página, a chamada será desligada, o que significa que a *PeerConnection* é fechada. Quando isto ocorre, o outro parceiro é notificado da saída do parceiro remoto.

3.5 Análise ao tráfego

Para provar e demostrar o funcionamento do WebRTC foram feitos algum testes em ambiente de rede local à aplicação. Iremos então analisar o cenário e as capturas de rede obtidas durante a execução dos testes.

Para a execução dos testes, foi montado seguinte cenário de rede em rede local:

Dispositivo	Endereço IP
Servidor (Servidor da aplicação)	192.168.1.68
PC 1	192.168.1.70
PC 2	192.168.1.64

Tabela 2 - Cenário de análise de tráfego

Entrada do PC1 na aplicação, escolhe a sala 1 e aceita o acesso aos dispositivos.

São trocadas mensagens:

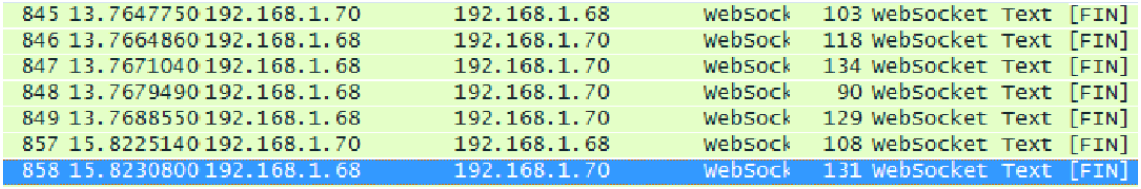


Figura 32- Comunicação na entrada do PC1

Origem	Destino	Mensagem
PC1	**Servidor**	Criar ou jantar na sala 1
Servidor	**PC1**	A sala 1 tem 0 clientes
Servidor	**PC1**	Pedido para criar ou jantar na sala 1
Servidor	**PC1**	Criada Sala 1
Servidor	**PC1**	Cliente PC1 entrou na sala 1
PC1	**Servidor**	gotUserMedia
Servidor	**PC1**	Recebido gotUserMedia

Tabela 3 - Mensagens trocadas na entrada do PC1

PC2 entra na aplicação, na sala 1 e permite o acesso aos dispositivos:

São trocadas as seguintes mensagens:

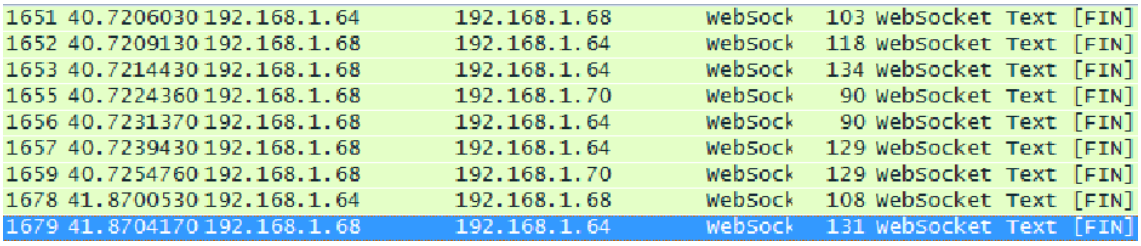
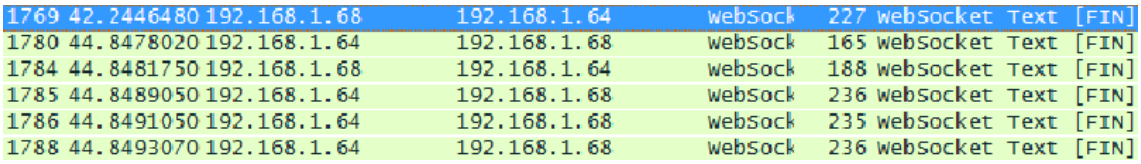


Figura 33 - Comunicação na entrada do PC2

Origem	Destino	Mensagem
PC2	**Servidor**	Criar ou juntar sala 1
Servidor	**PC2**	Sala 1 tem 1 cliente
Servidor	**PC2**	Pedido para criar ou entrar na sala 1
Servidor	**PC1**	Entrar na sala 1
Servidor	**PC2**	Entrou na sala 1
Servidor	**PC2**	Cliente PC2 entrou na sala 1
Servidor	**PC1**	Cliente PC2 entrou na sala 1
PC2	**Servidor**	gotUserMedia
Servidor	**PC2**	Recebido gotUserMedia

Tabela 4 - Mensagens trocadas na entrada do PC1

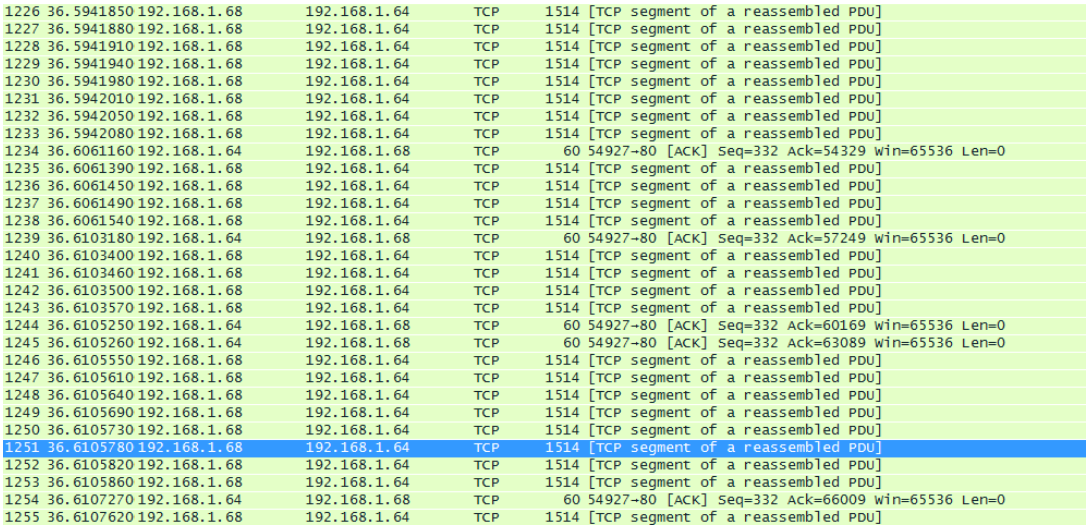
Comunicação entre PC1 e PC2 após ser iniciada a conversação:



1769	42.2446480	192.168.1.68	192.168.1.64	WebSock	227	WebSocket Text [FIN]
1780	44.8478020	192.168.1.64	192.168.1.68	WebSock	165	WebSocket Text [FIN]
1784	44.8481750	192.168.1.68	192.168.1.64	WebSock	188	WebSocket Text [FIN]
1785	44.8489050	192.168.1.64	192.168.1.68	WebSock	236	WebSocket Text [FIN]
1786	44.8491050	192.168.1.64	192.168.1.68	WebSock	235	WebSocket Text [FIN]
1788	44.8493070	192.168.1.64	192.168.1.68	WebSock	236	WebSocket Text [FIN]

Figura 34 - Comunicação *WebSocket* entre PC1 e PC2

Inicialmente são trocadas mensagens por *WebSockets* com informação acerca dos dados a enviar (áudio, vídeo e dados) assim como informação acerca dos candidatos e mensagens SDP (VER ANEXO). Depois são trocados dados (comunicação) através de TCP.



1226	36.5941850	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1227	36.5941880	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1228	36.5941910	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1229	36.5941940	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1230	36.5941980	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1231	36.5942010	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1232	36.5942050	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1233	36.5942080	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1234	36.6061160	192.168.1.64	192.168.1.68	TCP	60	54927→80 [ACK] Seq=332 Ack=54329 Win=65536 Len=0
1235	36.6061390	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1236	36.6061450	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1237	36.6061490	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1238	36.6061540	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1239	36.6103180	192.168.1.64	192.168.1.68	TCP	60	54927→80 [ACK] Seq=332 Ack=57249 Win=65536 Len=0
1240	36.6103400	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1241	36.6103460	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1242	36.6103500	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1243	36.6103570	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1244	36.6105250	192.168.1.64	192.168.1.68	TCP	60	54927→80 [ACK] Seq=332 Ack=60169 Win=65536 Len=0
1245	36.6105260	192.168.1.64	192.168.1.68	TCP	60	54927→80 [ACK] Seq=332 Ack=63089 Win=65536 Len=0
1246	36.6105550	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1247	36.6105610	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1248	36.6105640	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1249	36.6105690	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1250	36.6105730	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1251	36.6105780	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1252	36.6105820	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1253	36.6105860	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]
1254	36.6107270	192.168.1.64	192.168.1.68	TCP	60	54927→80 [ACK] Seq=332 Ack=66009 Win=65536 Len=0
1255	36.6107620	192.168.1.68	192.168.1.64	TCP	1514	[TCP segment of a reassembled PDU]

Figura 35 - Comunicação direta WebRTC entre PC1 e PC2

3.6 Testes e erros detetados

Segue-se uma pequena apresentação de alguns erros detetados durante a fase de testes da aplicação e a sua correção. Apresentam-se separadas as três fases de testes, as quais foram sempre acompanhadas por uma fase de correções. A definição das fases está especificada no sub-capítulo Plano de tarefas.

3.6.1.1 Fase 1

Problema detetado	Tester	Data	Solução	Estado
É necessário atualizar para mudar de sala	Joana Coelho	09/06/2014	Implementado campo para mudar posteriormente	Corrigido na fase 1
Não consigo tirar som	João Sousa	10/06/2014	Implementado botão de tirar som à	Corrigido na fase 1

			transmissão do parceiro	
Não se percebe quem disse o quê no chat	Tiago Rebelo	11/06/2014	Organizada a informação lado a lado	Corrigido na fase 2
Não funciona fora da rede local	Tiago Rebelo	11/06/2014	Implementado STUN e TURN	Corrigido na fase 2
Layout da página está todo desalinhado	Joana Coelho	09/06/2014	Alinhado layout	Corrigido na fase 3

Tabela 5 - Erros registados na primeira fase de testes

NOTA: Alguns erros apesar de corrigidos nesta fase, só foram finalmente considerados corrigidos em fases posteriores, conforme indicado.

3.6.1.2 Fase 2

Problema detetado	Tester	Data	Solução	Estado
É possível envio de ficheiros?	Tiago Silva	15/07/2014	Adicionada a opção de envio de ficheiros	Corrigido na fase 2
Texto do colega está em baixo da minha cara e vice-versa	Tiago Rebelo	07/07/2014	Reorganizada a página	Corrigido na fase 2
Mantém-se sem funcionar fora da rede local	Tiago Rebelo	07/07/2014	ALLOW-ORIGIN está a dar erro pois falta o header definido do lado do serviço utilizado, não tem ficheiro de configuração, por isso foram adicionados alguns endereços de STUN estáticos	Corrigido na fase 2
Estou na sala e não sei se está mais alguém	Joana Coelho	09/07/2014	Sala apenas "aberta" quando existe parceiro para chamada	Corrigido na fase 2

Como sei que o meu colega abandonou a sala??	Joana Coelho	09/07/2014	Adicionado aviso quando o parceiro remoto desliga a chamada	Corrigido na fase 2
Reposicionar e redimensionar vídeos	João Sousa	11/07/2014	Layout corrigido	Corrigido na fase 3

Tabela 6 - Erros registados na segunda fase de testes

NOTA: Alguns erros apesar de corrigidos nesta fase, só foram finalmente considerados corrigidos em fases posteriores, conforme indicado.

3.6.1.3 Fase 3

Problema detetado	Tester	Data	Solução	Estado
Não é intuitivo o posicionamento do botão upload	Tiago Silva	09/08/2014	Implementado drag-drop para envio de ficheiros	Corrigido na fase 3
Quando cancelo receção de um ficheiro e a seguir aceito outro, recebo bem o segundo, mas recebo também o primeiro com erro	João Sousa	10/08/2014	É agora controlado o ficheiro na verificação da receção de dados	Corrigido na fase 3
É impossível utilizar no telemóvel	Joana Coelho	09/08/2014	Correção de Layout	Corrigido na fase 3
Como sei que uma sala está cheia	Tiago Rebelo	12/07/2014	Adicionada notificação e aviso na página de que a sala está cheia	Corrigido na fase 3

Tabela 7 - Erros registados na terceira fase de testes

Os erros detetados na 3ª fase foram detetados e corrigidos, sendo que não se considerou útil a execução de mais testes à aplicação.

3.7 Aplicação multi-parceiro

Com o objetivo de permitir a comunicação entre múltiplos parceiros, foi adicionalmente desenvolvida uma aplicação demostrativa da capacidade de comunicação WebRTC entre múltiplos parceiros. Esta aplicação faz uso de uma solução já aqui apresentada, o simpleWebRTC, tratando-se de uma implementação simples daquilo que pode ser a comunicação entre diversos parceiros.
Esta aplicação, devido a ter sido desenvolvida apenas como uma demostração, apenas em âmbito académico e por fazer uso da solução simpleWebRTC, irá ser apresentada na íntegra. O facto de utilizar simpleWebRTC simplifica imensamente a aplicação, utilizando recursos fornecidos pela própria solução simpleWebRTC.
Pretende-se também com esta demostração evidenciar a simplicidade com a qual pode ser utilizada comunicação WebRTC a partir de uma solução disponível já existente. Esta simplicidade oferecida por vários serviços poderá ser uma das chaves para a proliferação desta tecnologia, sendo que se torna um grande impulsionador para a utilização de WebRTC.

```
var webrtc = new SimpleWebRTC({
    localVideoEl: 'VideoLocal',
    remoteVideosEl: 'VideosParceiros',
    autoRequestMedia: true});
webrtc.on('readyToCall', function () {
    room = prompt('Que sala que pretende aceder?');
    webrtc.joinRoom(room);});
```

Figura 36 - Código necessário para aplicação multi-parceiro com utilização de simpleWebRTC

Disto isto, basta então referenciar a mais recente biblioteca javascript disponibilizada pela simpleWebRTC, localizada em http://simplewebrtc.com/latest.js[18]. E posteriormente incluir o seguinte código javascript (Figura 36) após execução da página:
Explicado de forma simples, 'VideoLocal' e 'VideosParceiros' são *divs* necessárias no HTML da página, sendo que o vídeo local obtido a partir dos dispositivos locais é apresentado em 'VideoLocal' e os vários vídeos dos parceiros remotos são apresentados na *div* 'VideosParceiros'.
O parâmetro autoRequestMedia como verdadeiro apenas serve para ser automaticamente invocado o método getUserMedia, para pedir ao utilizador permissão de acesso aos seus dispositivos.
A função definida no evento 'readyToCall', irá ser lançada após o utilizador permitir o acesso aos seus dispositivos e irá obter a sala que o utilizador pretenda utilizar e irá fazer com que este se junte à sala.
Nesta aplicação, no momento em que entre um novo parceiro na sala, será adicionado um novo elemento vídeo na div 'VideosParceiros' e este poderá de imediato a comunicação WebRTC com todos os parceiros na sala. Podemos ver a aplicação em funcionamento na Figura 37.

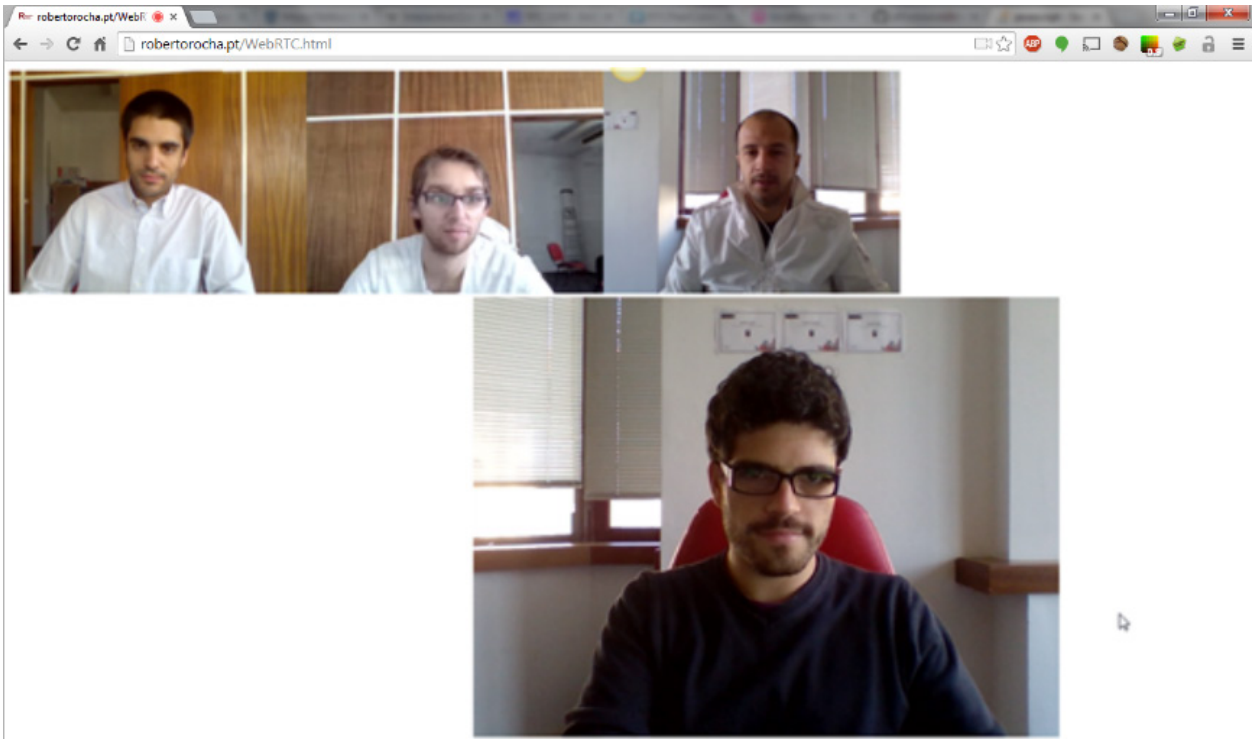


Figura 37 - Demostração da aplicação multi-parceiro

3.8 Sumário

Dado que este trabalho seria acompanhado de uma aplicação como prova de conceito, é importante apresentar essa aplicação em pormenor, definindo de forma clara o que se pretende com o seu desenvolvimento, o porquê de ter sido desenvolvida esta aplicação e como funciona de forma geral a aplicação. Em conjunto com tudo isto apresenta-se uma discussão das escolhas que tiveram de ser feitas no desenvolvimento da aplicação. Considerando todas as limitações existentes apresenta-se também a evolução do desenvolvimento da aplicação, detalhando cada um dos passos necessários. Posteriormente, é explicado o funcionamento da aplicação, tanto a forma como um utilizador faz uso da aplicação, como a forma como tecnicamente as operações são executadas durante a utilização da aplicação.

É ainda feita uma análise à transmissão dos dados da aplicação e são apresentados de forma simples os casos que surgiram nas fases de testes.

Por último, é feita uma apresentação de uma aplicação multi parceiro, desenvolvida com fins demostrativos da utilização da tecnologia na comunicação com múltiplos parceiros em simultâneo.

4 Conclusão

Este projeto foi desenvolvido no âmbito do Mestrado em Sistemas e Tecnologias de Informação para as Organizações, com o objetivo de apresentar a tecnologia WebRTC cuja importância é ainda desconhecida, mas que é promissora devido às suas características e conceito. Esta tecnologia permite comunicação direta em tempo real ponto-a-ponto. Comunicação essa que se torna importante numa altura em que cada vez mais é utilizada a internet para a efetuar chamadas, videochamadas e até para envio de mensagens. Outra característica determinante para a proliferação da utilização da tecnologia é o fato de esta permitir que tais chamadas sejam feitas a partir de um navegador sem a necessidade de instalação de qualquer *software* adicional, o que nos dias de hoje também é favorável.

Pretendia-se então com esta dissertação conhecer e compreender tudo o que estava por trás desta tecnologia, o seu funcionamento, as capacidades desta tecnologia assim como tudo o que a ela estava adjacente e cuja utilização era necessária. Para isto, várias tecnologias foram estudadas em detalhe e utilizadas, tendo sido aqui apresentadas de forma resumida.

Pretendia-se também que fosse conseguida uma implementação da tecnologia, como prova de conceito que fosse capaz de funcionar sob WebRTC e que permitisse observar a tecnologia em funcionamento, demostrando assim as suas capacidades.

Com tudo isto em mente, foi desenvolvido um estudo aprofundado sobre a tecnologia WebRTC e a comunicação ponto a ponto, assim como sobre várias tecnologias Web e protocolos que permitem a implementação de soluções WebRTC sob qualquer cenário de rede. Foram estudadas as dificuldades de standardização da tecnologia, assim como sua implementação e compatibilidades sobre os vários navegadores existentes. Foi também feito um importante enquadramento desta tecnologia na evolução da Web, assim como nas necessidades atuais.

Houve a necessidade de entender a pormenor todo o funcionamento do WebRTC, como é feita a sinalização, como são criadas as *PeerConnection* e estabelecidos os *DataChannel*, qual o funcionamento de toda a API do WebRTC. Para além disto, foi necessário compreender a evolução existente da Web que permite a implementação desta tecnologia, tal como os elementos de áudio e vídeo, assim como a possibilidade de interação com os dispositivos do utilizador, após devida autorização.

Após este estudo fica claro que, apesar da comunicação WebRTC dispensar a existência de servidor, este ainda é necessário para alojamento da aplicação e para permitir a sinalização (negociação de parâmetros de comunicação).

Foi necessário também perceber todo o funcionamento de rede, os problemas de comunicação sobre NAT e Firewall, assim como os métodos para contornar esses problemas. Foi feito o estudo de várias tecnologias de sinalização e transporte, assim como de todos os mecanismos que possibilitam o WebRTC como ICE, STUN e TURN.

Verificou-se uma grande facilidade na utilização e implementação da tecnologia WebRTC, dado que permite uma utilização livre de instalações e porque permite de forma relativamente simples a sua utilização no desenvolvimento de aplicações. O fato de existirem já muitas soluções disponíveis de forma aberta permite ainda que sejam utilizadas essas soluções no desenvolvimento de aplicações, simplificando ainda mais a sua implementação. A maior dificuldade na utilização da tecnologia prende-se com o fato de esta tecnologia ainda estar em desenvolvimento o que faz com que sofra alterações constantes.

Sobre a aplicação desenvolvida, foi necessário perceber como podia ser feita uma implementação da tecnologia e o que era necessário para tal, o que levou ao estudo de algumas tecnologias como Node.JS e a sua complexa implementação. Foi necessário fazer um estudo do que seria pretendido e exequível, considerando as capacidades atuais e o que já teria sido feito. Esta análise foi seguida de uma análise técnica aos mecanismos de sinalização e as suas implementações, assim como também foi necessário analisar como deveria estar organizada a aplicação.

Após a aplicação estar finalmente em funcionamento, existiram as obrigatórias fases de testes, o que levou à organização de utilizadores para os testes e definição de fases de testes e correções. Algumas dessas correções levaram à implementação de novas funcionalidades, como o envio de ficheiros, e originaram novas ideias, como a possibilidade de existir comunicação entre múltiplos parceiros. Considerando este *feedback* obtido dos utilizadores, foi posteriormente desenvolvida uma aplicação multi-parceiro.

Para o desenvolvimento da aplicação multi-parceiro decidiu-se utilizar uma das soluções disponíveis apresentadas. Esta utilização provou-se vantajosa devido ao tempo de desenvolvimento que foi despendido, o que prova também que a utilização destas aplicações poderá impulsionar a proliferação da tecnologia WebRTC uma vez que permitem uma implementação rápida e fácil da tecnologia.

Considerando tudo isto, conclui-se que o estudo feito permitiu a compreensão sobre o funcionamento da tecnologia, assim como das necessidades da mesma, capacidades de evolução e dificuldades de implementação. Com esta dissertação espera-se também que este conhecimento seja, pelo menos em parte, transportado para o leitor, ficando ele conhecedor da tecnologia em causa, e com um maior conhecimento que possuía anteriormente.
O desenvolvimento da aplicação principal assume-se como um grande resultado deste projeto, uma vez que tal desenvolvimento será utilizado para a produção de novas soluções na área da comunicação a nível empresarial. Pretende-se que sejam feitas novas e mais capazes aplicações baseadas nesta que permitam comunicação entre colaboradores e clientes, com capacidades que permitam dar suporte, efetuar reuniões, fazer videoconferências e que permita o trabalho em conjunto de entidades fisicamente separadas de uma forma rápida, direta e simples.
Com este trabalho espera-se que fique mais claro o funcionamento de WebRTC e as suas capacidades, trata-se de um trabalho de investigação que pretende a proliferação do conhecimento da tecnologia por forma a ser mais utilizada.

4.1 Desenvolvimentos futuros

Terminada a aplicação e servindo ela de protótipo para algo a desenvolver no futuro, é interessante perceber o que pode ser feito com base no já existente. Existem neste momento inúmeras funcionalidades implementadas não só sobre WebRTC, mas também sobre vídeo e áudio obtido a partir dos dispositivos terminais.
No futuro, pretende-se que aplicações desenvolvidas com base neste protótipo tenham algumas funcionalidades extras, como por exemplo:

- Partilha de ecrã – Permitir selecionar, durante a transmissão, a opção de alterar ou conjugar a transmissão entre vídeo local obtido de uma câmara, ou transmissão vídeo de conteúdo presente no ecrã de um utilizador.
- Deteção de voz – Uma funcionalidade interessante é também a deteção de voz, nomeadamente de qual o interlocutor em conversação atualmente. Pretende-se com isto melhorar a organização e facilitar o foco numa conversação em grupo.
- Deteção facial – Existem atualmente algumas bibliotecas JavaScript com capacidades de reconhecimento facial, este reconhecimento pode ter implementações interessantes numa aplicação WebRTC não só a nível de entretenimento (Conjugação com as muito populares expressões emocionais animadas), mas também a nível de acessibilidade e privacidade. Isto é, poderão ser implementadas funcionalidades como:

- Acessibilidade – Permitir utilização da aplicação por pessoas com deficiências motoras, através da introdução de navegação com rastreamento do olhar;
- Privacidade – Permitir que apenas seja apresentado vídeo quando seja detetada uma face humana, por modo a evitar que seja transmitido vídeo indevido. Ou pelo contrário, possibilitar que seja enviado o vídeo com censura das faces humanas.

- Autenticação Segura – No âmbito empresarial torna-se necessária a existência de autenticação Segura na aplicação, esta deve ser assegurada através de servidores e ligações seguras. Associada a esta autenticação será também interessante integrar informação sobre os utilizadores.
- Permitir efetuar chamadas – Seria também interessante a possibilidade de efetuar ligações a um parceiro sem que este tenha de aceder à aplicação e entrar na mesma sala. Esta funcionalidade deve ser integrada com informação sobre os utilizadores
- Permitir mandantes – Dado tratar-se de uma aplicação a usar a nível empresarial, será importante a existência de mandantes. Estes mandantes têm como objetivo permitir a diferenciação da mesma aplicação. Isto é, deve permitir que seja utilizada a mesma aplicação para várias empresas de forma transparente, sendo apresentada toda a imagem (entenda-se: conjunto de imagens, slogans e outras informações) correspondente à empresa em causa.

4.1.1 Utilização em meio empresarial

Conforme foi referido anteriormente, pretende-se que esta aplicação sirva como um ponto de partida no desenvolvimento de novas aplicações de comunicação, no âmbito de uma empresa. Assim, espera-se a curto prazo o início do desenvolvimento de várias aplicações, que utilizem WebRTC, tal como:

- Aplicação de suporte ao cliente
    - Será criada uma aplicação que permita a comunicação através de voz, vídeo e texto, que os clientes comuniquem diretamente com colaboradores responsáveis.
    - Esta aplicação deve permitir que para isso o cliente não necessite de fazer mais do que abrir uma página Web, no entanto deve ser possível toda esta comunicação ao mesmo tempo que é apresentada uma página com imagem (entenda-se: conjunto de imagens, slogans e outras informações)

de uma empresa ou grupo específico. Nesta aplicação será também interessante a implementação de métodos de partilha de ecrã via WebRTC.

- Aplicação para apresentações
    - Será desenvolvida uma aplicação que irá permitir a divulgação via aplicação Web de apresentações e reuniões.
    - Pretende-se que exista um controlo de utilizadores ao nível de controlo de acesso e também ao controlo na comunicação dos utilizadores na apresentação (permitir ou não voz/vídeo; deteção de voz para foco na apresentação). É importantíssimo na aplicação que seja possível a partilha de ecrã e a autenticação de utilizadores.
- Aplicação para comunicação interna
    - Será desenvolvida uma aplicação que permita a comunicação via vídeo, áudio, texto e partilha de ficheiros. Com esta aplicação pretende-se que os funcionários da empresa consigam comunicar entre si, em par ou conjunto de pessoas, para tudo o que possam necessitar. É obrigatório que exista a possibilidade de contactar com uma ou mais pessoas sem definição inicial da sala de comunicação. Para isso deverá ser implementado um sistema que permita a criação de salas automaticamente quando um utilizador contacte outro e, que o parceiro seja de imediato (e automaticamente) encaminhado para uma página Web cuja sala de comunicação está especificada à partida.

REFERÊNCIAS

[1] "Historical Timeline of Computable Knowledge: 1900-1959." [Online]. Available: http://www.wolframalpha.com/docs/timeline/computable-knowledge-history-5.html. [Accessed: 27-Sep-2014].

[2] "Dennis G. Jerz: On the Trail of the Memex. Vannevar Bush, Weblogs and the Google Galaxy." [Online]. Available: http://www.dichtung-digital.org/2003/issue/1/jerz/. [Accessed: 27-Sep-2014].

[3] "What is Hypertext?" [Online]. Available: http://www.w3.org/WhatIs.html. [Accessed: 27-Sep-2014].

[4] "ARPANET Timeline." [Online]. Available: http://www.cs.utexas.edu/users/chris/think/ARPANET/Timeline/. [Accessed: 27-Sep-2014].

[5] "Web and HyperText History." [Online]. Available: http://www.webdirections.org/history/. [Accessed: 27-Sep-2014].

[6] "Hypertext Transfer Protocol -- HTTP/1.1." [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html. [Accessed: 27-Sep-2014].

[7] "HTML/Specifications - Web Education Community Group." [Online]. Available: http://www.w3.org/community/webed/wiki/HTML/Specifications. [Accessed: 18-Dec-2013].

[8] "Press Release," 16-Sep-2007. [Online]. Available: https://web.archive.org/web/20070916144913/http:/wp.netscape.com/newsref/pr/newsrelease67.html. [Accessed: 27-Sep-2014].

[9] "It was when not if... Google Chrome | DoesWhat." .

[10] "Google release of WebRTC source code from Harald Alvestrand on 2011-05-31 (public-webrtc@w3.org from May 2011)." [Online]. Available: http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html. [Accessed: 27-Sep-2014].

[11] "What was the history of WebRTC inside Google before it was released to the public? - Quora." [Online]. Available: http://www.quora.com/What-was-the-history-of-WebRTC-inside-Google-before-it-was-released-to-the-public?ref=fb. [Accessed: 27-Sep-2014].

[12] "Desktop Applications Vs. Web Applications." [Online]. Available: http://www.streetdirectory.com/travel_guide/114448/programming/desktop_applications_vs_web_applications.html. [Accessed: 27-Sep-2014].

[13] "Desktop vs. Web Applications: A Deeper Look and Comparison," *Segue Technologies*. [Online]. Available: http://www.seguetech.com/blog/2013/06/07/desktop-vs-web-applications-deeper-comparison. [Accessed: 08-Oct-2014].

[14] "HTML5 Introduction." [Online]. Available: http://www.w3schools.com/html/html5_intro.asp. [Accessed: 27-Sep-2014].

[15] "WebRTC." [Online]. Available: http://www.webrtc.org/. [Accessed: 08-Oct-2014].

[16] D. B. O. D. 18 and 2013, “What WebRTC is and why you should care,” *PandoDaily*, 18-Dec-2013. .
[17] “Is Skype a WebRTC Killer? - Post - No Jitter.” [Online]. Available: http://www.nojitter.com/post/240158376/is-skype-a-webrtc-killer. [Accessed: 08-Oct-2014].
[18] “SimpleWebRTC.js from &yet.” [Online]. Available: http://simplewebrtc.com/. [Accessed: 27-Sep-2014].
[19] “Is WebRTC ready yet?” [Online]. Available: http://iswebrtcreadyyet.com/. [Accessed: 27-Sep-2014].
[20] “Talky.” [Online]. Available: https://talky.io/. [Accessed: 27-Sep-2014].
[21] “EasyRTC - Open Source WebRTC tools for Enterprise Developers.” [Online]. Available: http://easyrtc.com/. [Accessed: 27-Sep-2014].
[22] “EasyRTC - Features.” [Online]. Available: http://easyrtc.com/features/. [Accessed: 27-Sep-2014].
[23] “Six WebRTC Video Calling Solutions Ready to Use Right Now From Your Browser.” [Online]. Available: http://www.webrtcworld.com/topics/webrtc-world/articles/372507-six-webrtc-video-calling-solutions-ready-use-right.htm. [Accessed: 27-Sep-2014].
[24] “appear.in,” *appear.in*. [Online]. Available: https://appear.in. [Accessed: 27-Sep-2014].
[25] “Developer Tools - vLine.” [Online]. Available: https://vline.com/developer/. [Accessed: 27-Sep-2014].
[26] “Home - Bistri Developers - Open WebRTC Platform - API, SDK -,” *Bistri Developers - Open WebRTC Platform - API, SDK -*. [Online]. Available: http://developers.bistri.com/. [Accessed: 27-Sep-2014].
[27] “OnSIP Network Platform | OnSIP.” [Online]. Available: http://www.onsip.com/webrtc-sip-network/platform/e. [Accessed: 27-Sep-2014].
[28] “Google unveils Hangouts: a unified messaging system for Android, iOS, and Chrome,” *The Verge*. [Online]. Available: http://www.theverge.com/2013/5/15/4332556/google-hangouts-unified-messaging-google-io-2013. [Accessed: 27-Sep-2014].
[29] “Hangouts do Google+ – Hangouts do Google.” [Online]. Available: https://www.google.com/hangouts/. [Accessed: 27-Sep-2014].
[30] K. C. Tofel, “So long plug-ins, Google Hangouts works with WebRTC,” 06-Jul-2014. .
[31] J.-E. Sneddon, “Google+ Hangouts Plugin No Longer Needed in Chrome,” *OMG! Chrome!*. [Online]. Available: http://www.omgchrome.com/google-hangouts-chrome-plugin-free-2/. [Accessed: 27-Sep-2014].
[32] “Hangouts.” [Online]. Available: https://www.google.com/tools/dlpage/hangoutplugin. [Accessed: 27-Sep-2014].
[33] “Five Best Web-Based Video Chat Services,” *Lifehacker*. [Online]. Available: http://lifehacker.com/five-best-web-based-video-chat-services-844494319. [Accessed: 13-Oct-2014].
[34] “The 7 Best Video Chat Apps for Your Smartphone,” *Search Engine Journal*. .

[35] “Capturing Audio & Video in HTML5 - HTML5 Rocks.” [Online]. Available: http://www.html5rocks.com/pt/tutorials/getusermedia/intro/. [Accessed: 27-Sep-2014].
[36] “Media Capture and Streams.” [Online]. Available: http://www.w3.org/TR/mediacapture-streams/. [Accessed: 27-Sep-2014].
[37] “NavigatorUserMedia.getUserMedia() | MDN.” [Online]. Available: https://developer.mozilla.org/en-US/docs/NavigatorUserMedia.getUserMedia. [Accessed: 27-Sep-2014].
[38] “HTML5 and WebRTC.” [Online]. Available: http://html5videoguide.net/presentations/WebDirCode2012/#page8. [Accessed: 27-Sep-2014].
[39] “JavaScript: The Definitive Guide, 6th Edition - O’Reilly Media.” [Online]. Available: http://shop.oreilly.com/product/9780596805531.do. [Accessed: 27-Sep-2014].
[40] “JavaScript | MDN.” [Online]. Available: https://developer.mozilla.org/pt-PT/docs/Web/JavaScript. [Accessed: 27-Sep-2014].
[41] A. Grange and H. Alvestrand, “VP8 as RTCWEB Mandatory to Implement.” [Online]. Available: http://tools.ietf.org/html/draft-alvestrand-rtcweb-vp8-02. [Accessed: 27-Sep-2014].
[42] M. Isomaki, D. Singer, X. Marjou, G. Martin-Cocher, C. Jennings, J. Rosenberg, B. Aboba, BoB, and G. Mandyam, “H.264 as Mandatory to Implement Video Codec for WebRTC.” [Online]. Available: http://tools.ietf.org/html/draft-burman-rtcweb-h264-proposal-02. [Accessed: 27-Sep-2014].
[43] H. S. <schulzrinne@cs.columbia.edu>, “RTP Profile for Audio and Video Conferences with Minimal Control.” [Online]. Available: http://tools.ietf.org/html/rfc3551#section-4.5.14. [Accessed: 08-Oct-2014].
[44] T. Taylor and H. Schulzrinne, “Definition of Events for Modem, Fax, and Text Telephony Signals.” [Online]. Available: http://tools.ietf.org/html/rfc4734. [Accessed: 08-Oct-2014].
[45] T. Terriberry and K. Vos, “Definition of the Opus Audio Codec.” [Online]. Available: https://tools.ietf.org/html/draft-ietf-codec-opus-16. [Accessed: 08-Oct-2014].
[46] “Firefox - WebRTC.” [Online]. Available: http://www.webrtc.org/firefox. [Accessed: 27-Sep-2014].
[47] “Chrome’s WebRTC roadmap,” *Chromium Blog*. .
[48] “Chrome and WebRTC,” *OnSIP*. [Online]. Available: http://www.onsip.com/webrtc-sip-network/webrtc-browser-support/chrome-webrtc. [Accessed: 27-Sep-2014].
[49] “General Overview - WebRTC.” [Online]. Available: http://www.webrtc.org/reference/architecture. [Accessed: 27-Sep-2014].
[50] “webrtc · apis · WPD · WebPlatform.org.” [Online]. Available: http://docs.webplatform.org/wiki/apis/webrtc. [Accessed: 27-Sep-2014].
[51] “WebRTC Native APIs - WebRTC.” [Online]. Available: http://www.webrtc.org/reference/native-apis. [Accessed: 27-Sep-2014].
[52] “API Description - WebRTC.” [Online]. Available: http://www.webrtc.org/reference/api-description. [Accessed: 27-Sep-2014].

[53] “MediaStream API,” *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_API. [Accessed: 08-Oct-2014].
[54] “WebRTC Security,” *OnSIP*. [Online]. Available: http://www.onsip.com/webrtc-sip-network/webrtc-implementation/webrtc-security. [Accessed: 27-Sep-2014].
[55] “WebRTC in the real world: STUN, TURN and signaling - HTML5 Rocks.” [Online]. Available: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/. [Accessed: 27-Sep-2014].
[56] *Network Protocols Handbook*. Javvin Technologies Inc., 2005.
[57] “Introduction to WebRTC architecture,” *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Guide/API/WebRTC/WebRTC_architecture. [Accessed: 27-Sep-2014].
[58] “An Intro to WebRTC’s NAT/Firewall Problem - webrtcHacks.” [Online]. Available: http://webrtchacks.com/an-intro-to-webrtcs-natfirewall-problem/. [Accessed: 27-Sep-2014].
[59] “Nat traversal in WebRTC context,” 09:06:54 UTC.
[60] J. R. <jdrosen@cisco.com>, “Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols.” [Online]. Available: http://tools.ietf.org/html/rfc5245. [Accessed: 27-Sep-2014].
[61] “Interactive Connectivity Establishment | Internet Society.” [Online]. Available: http://www.internetsociety.org/articles/interactive-connectivity-establishment. [Accessed: 27-Sep-2014].
[62] D. Wing, P. Matthews, R. Mahy, and J. Rosenberg, “Session Traversal Utilities for NAT (STUN).” [Online]. Available: http://tools.ietf.org/html/rfc5389. [Accessed: 27-Sep-2014].
[63] Google, “stun_server_list - Google Project Hosting,” *STUN Server List*. [Online]. Available: https://code.google.com/p/natvpn/source/browse/trunk/stun_server_list. [Accessed: 08-Oct-2014].
[64] P. Matthews, R. Mahy, and J. Rosenberg, “Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN).” [Online]. Available: http://tools.ietf.org/html/rfc5766. [Accessed: 27-Sep-2014].
[65] “WebRTC: If it’s P2P, why do I need a server?” [Online]. Available: http://blog.vline.com/post/63765098884/webrtc-if-its-p2p-why-do-i-need-a-server. [Accessed: 27-Sep-2014].
[66] “WebRTC Signaling Concepts ® Muaz Khan.” [Online]. Available: https://www.webrtc-experiment.com/docs/WebRTC-Signaling-Concepts.html. [Accessed: 27-Sep-2014].
[67] “jsep.png (834×520).” [Online]. Available: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/jsep.png. [Accessed: 27-Sep-2014].
[68] admin, “XMPP Technologies: Jingle – The XMPP Standards Foundation.” .

[69] S. Ludwig, J. Beda, P. Saint-Andre, R. McQueen, S. Egan, and J. Hildebrand, “Jingle,” 23-Dec-2009. [Online]. Available: http://xmpp.org/extensions/xep-0166.html. [Accessed: 27-Sep-2014].
[70] “AJAX alliance recognizes mashups,” *InfoWorld*, 24-Sep-2007. [Online]. Available: http://www.infoworld.com/article/2642719/application-development/ajax-alliance-recognizes-mashups.html. [Accessed: 27-Sep-2014].
[71] “5 Different Signaling Protocol Options for WebRTC Services,” *BlogGeek.me*. [Online]. Available: http://bloggeek.me/siganling-protocol-webrtc/. [Accessed: 27-Sep-2014].
[72] “JSEP Signaling Model.” .
[73] “Re: JSEP proposal from Harald Alvestrand on 2012-01-02 (public-webrtc@w3.org from January 2012).” [Online]. Available: http://lists.w3.org/Archives/Public/public-webrtc/2012Jan/0002.html. [Accessed: 27-Sep-2014].
[74] “draft-ietf-rtcweb-jsep-03 - Javascript Session Establishment Protocol.” [Online]. Available: http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03. [Accessed: 27-Sep-2014].
[75] “The WebSocket API.” [Online]. Available: http://dev.w3.org/html5/websockets/. [Accessed: 27-Sep-2014].
[76] M. Ubl, E. K. P. O. 20th, and 2010 Comments: 46 Your browser may not support the functionality in this article, “Apresentando WebSockets: trazendo soquetes para a web - HTML5 Rocks,” *HTML5 Rocks - Um recurso para desenvolvedores HTML5 em web aberta*. [Online]. Available: http://www.html5rocks.com/pt/tutorials/websockets/basics/. [Accessed: 27-Sep-2014].
[77] “RFC 6455 - The WebSocket Protocol.” [Online]. Available: http://tools.ietf.org/html/rfc6455. [Accessed: 27-Sep-2014].
[78] “WebSocket.org | The Benefits of WebSocket.” [Online]. Available: https://www.websocket.org/quantum.html. [Accessed: 27-Sep-2014].
[79] “WebRTC » WebSockets ® Muaz Khan.” [Online]. Available: https://www.webrtc-experiment.com/websocket/. [Accessed: 27-Sep-2014].
[80] “RFC 7118.” [Online]. Available: https://datatracker.ietf.org/doc/rfc7118/. [Accessed: 27-Sep-2014].
[81] T. Quintana, “HTML5 WebRTC and SIP Over WebSockets.” Abril-2013.
[82] Rosenberg, Schulzrinne, Camarillo, Johnston, Peterson, Sparks, Handley, and Schooler, “SIP: Session Initiation Protocol.” Jun-2002.
[83] M. Tuexen, S. Loreto, and R. Jesup, “WebRTC Data Channels.” [Online]. Available: http://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-11. [Accessed: 27-Sep-2014].
[84] “Using the WebRTC Data Channel.” [Online]. Available: http://danristic.com/html5/javascript/webrtc/2013/08/13/using-the-webrtc-data-channel.html. [Accessed: 27-Sep-2014].
[85] “Why was SCTP Selected for WebRTC’s Data Channel?,” *BlogGeek.me*. [Online]. Available: http://bloggeek.me/sctp-data-channel/. [Accessed: 27-Sep-2014].

[86] “RFC 3286 - An Introduction to the Stream Control Transmission Protocol (SCTP).” [Online]. Available: http://tools.ietf.org/html/rfc3286. [Accessed: 27-Sep-2014].
[87] “Protocolo SCTP.” [Online]. Available: http://www.inf.ufrgs.br/~cechin/Net/sctp/sctp.html. [Accessed: 27-Sep-2014].
[88] “What is Flooding? - Definition from Techopedia,” *Techopedia.com*. [Online]. Available: http://www.techopedia.com/definition/16190/flooding-networking. [Accessed: 08-Oct-2014].
[89] “What is UDP (User Datagram Protocol)? - Definition from WhatIs.com.” [Online]. Available: http://searchsoa.techtarget.com/definition/UDP. [Accessed: 27-Sep-2014].
[90] N. Modadugu and E. Rescorla, “Datagram Transport Layer Security.” [Online]. Available: http://tools.ietf.org/html/rfc4347. [Accessed: 27-Sep-2014].
[91] “RTP, Real-time Transport Protocol.” [Online]. Available: http://www.networksorcery.com/enp/protocol/rtp.htm. [Accessed: 27-Sep-2014].
[92] V. Jacobson, R. Frederick, S. Casner, and H. Schulzrinne, “RTP: A Transport Protocol for Real-Time Applications.” [Online]. Available: http://tools.ietf.org/html/rfc3550. [Accessed: 27-Sep-2014].
[93] “SRTP - voip-info.org.” [Online]. Available: http://www.voip-info.org/wiki/view/SRTP. [Accessed: 27-Sep-2014].
[94] “What is SRTP (Secure Real-Time Transport Protocol or Secure RTP)? - Definition from WhatIs.com.” [Online]. Available: http://whatis.techtarget.com/definition/SRTP-Secure-Real-Time-Transport-Protocol-or-Secure-RTP. [Accessed: 27-Sep-2014].
[95] D. McGrew and E. Rescorla, “Datagram Transport Layer Security (DTLS) Extension to Establish Keys for Secure Real-time Transport Protocol (SRTP).” [Online]. Available: http://tools.ietf.org/html/rfc5764. [Accessed: 27-Sep-2014].
[96] M. Handley and V. Jacobson, “SDP: Session Description Protocol,” Mar. 1998.
[97] “Session Description Protocol.” .
[98] S. and C. J. Nandakumar, “SDP for the WebRTC,” 23-Feb-2013. [Online]. Available: http://tools.ietf.org/id/draft-nandakumar-rtcweb-sdp-01.html. [Accessed: 27-Sep-2014].
[99] “eventmachine @ GitHub.” [Online]. Available: http://rubyeventmachine.com/. [Accessed: 08-Oct-2014].
[100] “Twisted.” [Online]. Available: https://twistedmatrix.com/trac. [Accessed: 08-Oct-2014].
[101] “node.js.” [Online]. Available: http://nodejs.org/. [Accessed: 27-Sep-2014].
[102] “Chrome V8 — Google Developers.” [Online]. Available: https://developers.google.com/v8/. [Accessed: 08-Oct-2014].
[103] “6 things you should know about Node.js,” *JavaWorld*, 12-Dec-2013. [Online]. Available: http://www.javaworld.com/article/2079190/scripting-jvm-languages/6-things-you-should-know-about-node-js.html. [Accessed: 27-Sep-2014].
[104] C. A. Cois, “Why You Should Learn Node.js Today.” .

[105] “Why The Hell Would I Use Node.js? A Case-by-Case Tutorial,” *Toptal Engineering Blog*, 13-Aug-2013. [Online]. Available: http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js. [Accessed: 27-Sep-2014].
[106] “socket.io.” [Online]. Available: https://www.npmjs.org/package/socket.io. [Accessed: 27-Sep-2014].
[107] “Socket.IO.” .
[108] “node-static.” [Online]. Available: https://www.npmjs.org/package/node-static. [Accessed: 27-Sep-2014].
[109] “6 Must Have Node.js Modules | Nodejitsu Inc.” [Online]. Available: http://blog.nodejitsu.com/6-must-have-nodejs-modules/. [Accessed: 27-Sep-2014].
[110] Google, “adapter.js - webrtc - Web-based real-time communication - Google Project Hosting,” *adapter.js - webrtc - Web-based real-time communication - Google Project Hosting*. [Online]. Available: https://code.google.com/p/webrtc/source/browse/trunk/samples/js/base/adapter.js?r=4259. [Accessed: 08-Oct-2014].
[111] ComputeEngineOnDemand, “Obtenção de servidores TURN,” *Obtenção de servidores TURN*. [Online]. Available: http://computeengineondemand.appspot.com/turn?username=41784574&key=4080218913. [Accessed: 08-Oct-2014].
[112] “alfreddatakillen/computeengineondemand,” *GitHub*. [Online]. Available: https://github.com/alfreddatakillen/computeengineondemand. [Accessed: 08-Oct-2014].
[113] “WebRTC data channels - Grupos do Google.” [Online]. Available: https://groups.google.com/forum/#!topic/discuss-webrtc/U927CZaCdKU. [Accessed: 08-Oct-2014].

ANEXOS

A. Casos de uso entrada na aplicação

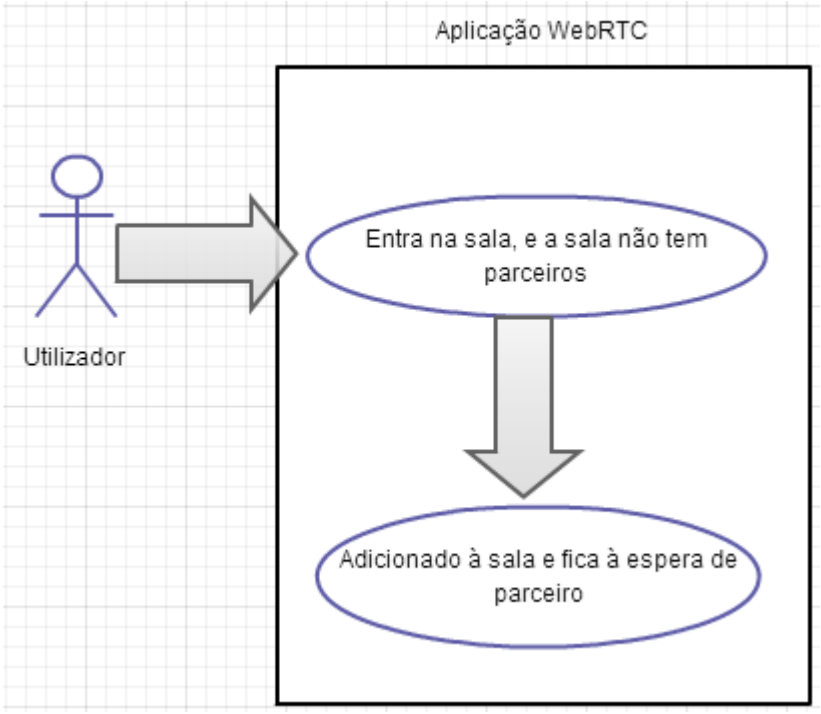


Figura 38 - Utilizador entra em sala vazia

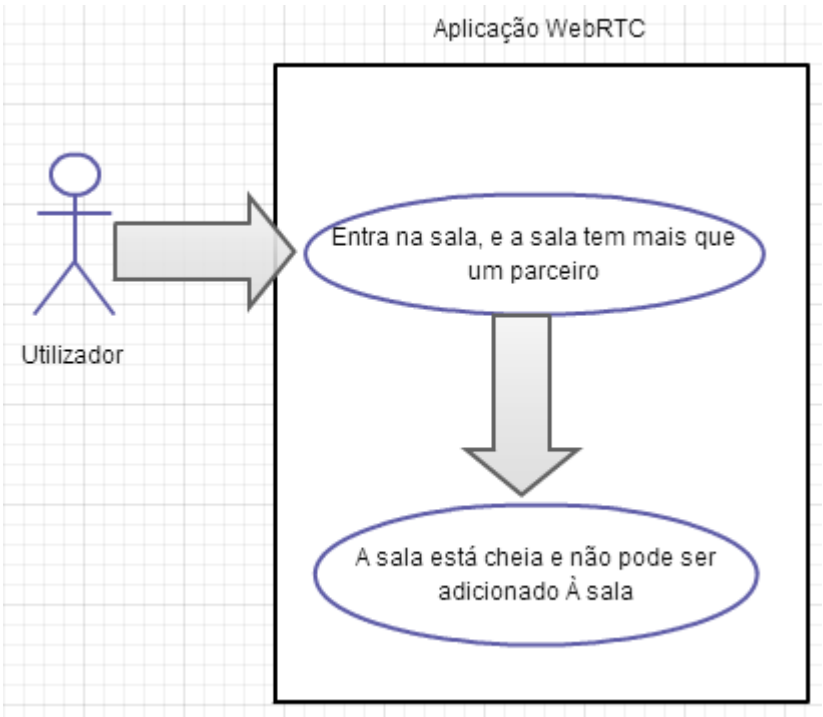


Figura 39 - Utilizador entra em sala cheia

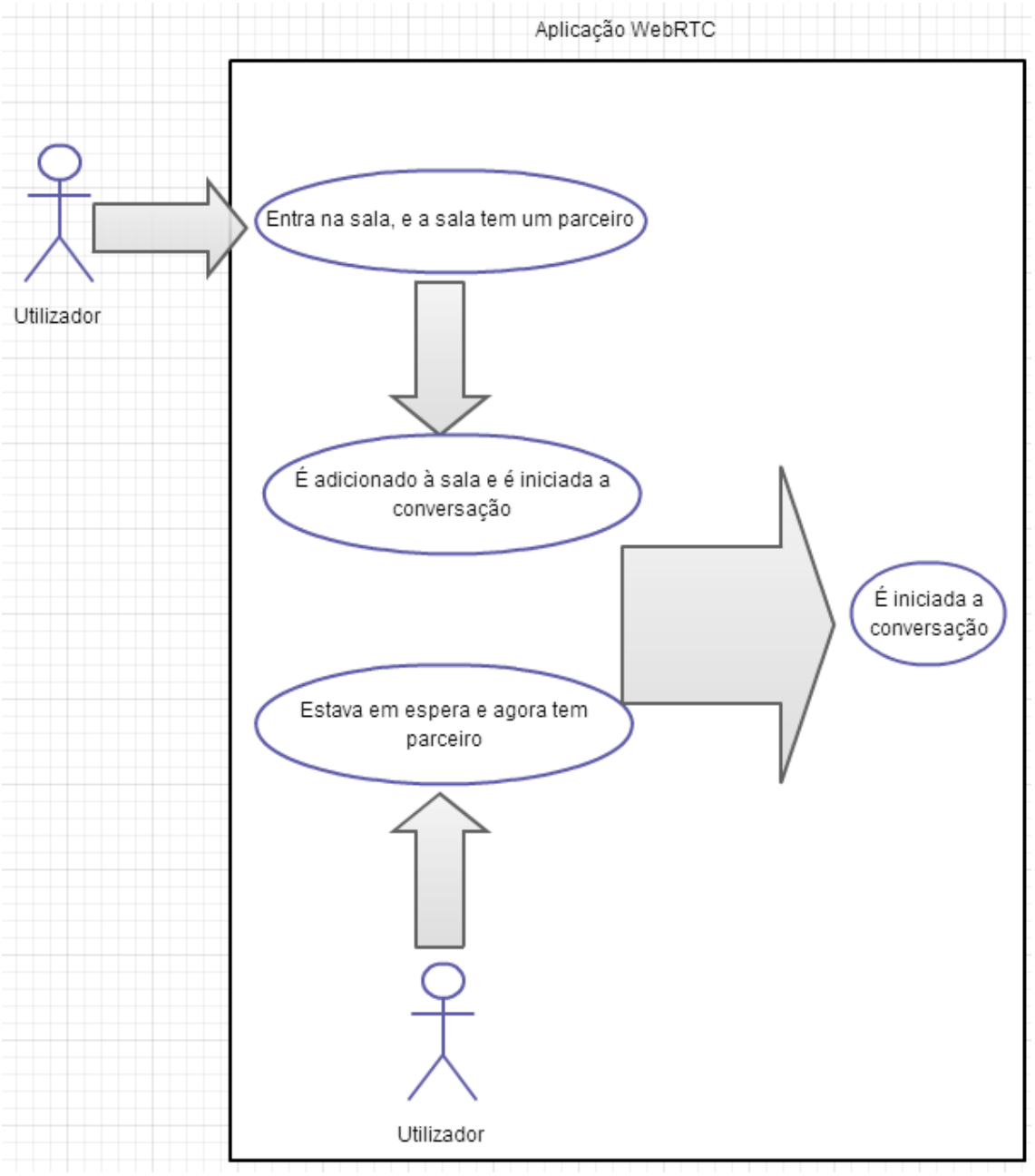


Figura 40 - Utilizador entra em sala com um parceiro

B. Código Fase 1 da Aplicação

```
<DOCTYPE!>
<html>
	<head>
	</head>
	<body>
		<video id="video"/>
		<script>
			navigator.getUserMedia = navigator.getUserMedia ||
			 navigator.webkitGetUserMedia || navigator.mozGetUserMedia;

			var constraints = {audio:true, video: true};

			function successCallback(localMediaStream) {
			 window.stream = localMediaStream;
			 var video = document.querySelector("video");
			 video.src = window.URL.createObjectURL(localMediaStream);
			 video.play();
			}

			function errorCallback(error){
			 console.log("navigator.getUserMedia error: ", error);
			}

			navigator.getUserMedia(constraints, successCallback, errorCallback);
		</script>
	</body>
</html>
```

C. Código Fase 2 da Aplicação

```
<DOCTYPE!>
<html>
	<head>
		<script src='js/lib/adapter.js'></script>
	</head>
	<body>
		 <video id="videoLocal" autoplay muted></video>
		 <video id="videoRemoto" autoplay muted></video>
		<script>
			var localStream, localPeerConnection, remotePeerConnection;

			var videoLocal = document.getElementById("videoLocal");
			var videoRemoto = document.getElementById("videoRemoto");

			function iniciarChamada(stream){
			 var servers = null;
			 videoLocal.src = URL.createObjectURL(stream);
			 localStream = stream;
			 localPeerConnection = new RTCPeerConnection(servers);
			 localPeerConnection.onicecandidate = gotLocalIceCandidate;

			 remotePeerConnection = new RTCPeerConnection(servers);
			 remotePeerConnection.onicecandidate = gotRemoteIceCandidate;
			 remotePeerConnection.onaddstream = trataStreamRemota;

			 localPeerConnection.addStream(localStream);
			 localPeerConnection.createOffer(gotLocalDescription,handleError);
			}

			function iniciar() {
			 getUserMedia({audio:true, video:true}, iniciarChamada,
				function(error) {
				});
			}

			function gotLocalDescription(description){
			 localPeerConnection.setLocalDescription(description);
			 //trace("Offer from localPeerConnection: \n" + description.sdp);
			 remotePeerConnection.setRemoteDescription(description);
			 remotePeerConnection.createAnswer(gotRemoteDescription,handleError);
			}

			function gotRemoteDescription(description){
			 remotePeerConnection.setLocalDescription(description);
```

```
                    localPeerConnection.setRemoteDescription(description);
                }

                function desligar() {
                    localPeerConnection.close();
                    remotePeerConnection.close();
                }

                function trataStreamRemota(event){
                    videoRemoto.src = URL.createObjectURL(event.stream);
                }

                function gotLocalIceCandidate(event){
                    if (event.candidate) {
                        remotePeerConnection.addIceCandidate(new
RTCIceCandidate(event.candidate));
                    }
                }

                function gotRemoteIceCandidate(event){
                    if (event.candidate) {
                        localPeerConnection.addIceCandidate(new
RTCIceCandidate(event.candidate));
                    }
                }

                function handleError(){}

                iniciar();
                window.onbeforeunload=desligar;
            </script>
        </body>
</html>
```

D. Código Fase 3 da Aplicação

```
<DOCTYPE!>
<html>
	<head>
		<script src='js/lib/adapter.js'></script>
	</head>
	<body>
		<video id="videoLocal" autoplay muted></video>
		<video id="videoRemoto" autoplay muted></video>
		<textarea id="caixaTextoEnvio"></textarea>
		<textarea id="caixaTextoRececao" disabled></textarea>
		<button id="botaoEnvio">Enviar</button>
		<script>
			var localStream, localPeerConnection, remotePeerConnection;

			var videoLocal = document.getElementById("videoLocal");
			var videoRemoto = document.getElementById("videoRemoto");
			var sendChannel, receiveChannel;

			var botaoEnvio = document.getElementById("botaoEnvio");
			botaoEnvio.onclick = sendData;


			function iniciarChamada(stream){
			  var servers = null;
			  videoLocal.src = URL.createObjectURL(stream);
			  localStream = stream;
			  localPeerConnection = new RTCPeerConnection(servers);
			   try {
					sendChannel =
localPeerConnection.createDataChannel("sendDataChannel",{reliable: false});
				} catch (e) {
					alert('Erro ao criar o DataChannel');
				}
			  localPeerConnection.onicecandidate = gotLocalIceCandidate;

			  remotePeerConnection = new RTCPeerConnection(servers);
			  remotePeerConnection.onicecandidate = gotRemoteIceCandidate;
			  remotePeerConnection.onaddstream = trataStreamRemota;

			  remotePeerConnection.ondatachannel = gotReceiveChannel;

			  localPeerConnection.addStream(localStream);
			  localPeerConnection.createOffer(gotLocalDescription,handleError);
			}
```

```
                function sendData() {
                  var data = document.getElementById("caixaTextoEnvio").value;
                  sendChannel.send(data);
                }

                function desligar() {
                  sendChannel.close();
                  receiveChannel.close();
                  localPeerConnection.close();
                  remotePeerConnection.close();
                }

                function iniciar() {
                  getUserMedia({audio:true, video:true}, iniciarChamada,
                        function(error) {
                        });
                }

                function gotLocalDescription(description){
                  localPeerConnection.setLocalDescription(description);
                  remotePeerConnection.setRemoteDescription(description);
                  remotePeerConnection.createAnswer(gotRemoteDescription,handleError);
                }

                function gotRemoteDescription(description){
                  remotePeerConnection.setLocalDescription(description);
                  localPeerConnection.setRemoteDescription(description);
                }

                function trataStreamRemota(event){
                  videoRemoto.src = URL.createObjectURL(event.stream);
                }

                function gotLocalIceCandidate(event){
                  if (event.candidate) {
                        remotePeerConnection.addIceCandidate(new
RTCIceCandidate(event.candidate));
                  }
                }

                function gotRemoteIceCandidate(event){
                  if (event.candidate) {
                        localPeerConnection.addIceCandidate(new
RTCIceCandidate(event.candidate));
                  }
                }

                function gotReceiveChannel(event) {
```

```
                receiveChannel = event.channel;
                receiveChannel.onmessage = (function(evt){
                        document.getElementById("caixaTextoRececao").value = evt.data;
                });
            }

            function handleError(){}

            iniciar();
            window.onbeforeunload=desligar;
        </script>
    </body>
</html>
```

F. Código Servidor da Aplicação

```
//servidor tradicional que permita o acesso aos ficheiros, implementação de controlo de acesso
a todas as origens
var static = require('node-static');
var http = require('http');
var file = new(static.Server)();
var app = http.createServer(function (req, res) {
        res.setHeader('Access-Control-Allow-Origin', '*');
  file.serve(req, res);
}).listen(80);

//configuração do websocket
var keepsala;
var io = require('socket.io').listen(app);
io.sockets.on('connection', function (socket){//comunicação nos sockets feita em inglês, para
evitar erros

        function log(){
                var array = [">>> "];
          for (var i = 0; i < arguments.length; i++) {
                array.push(arguments[i]);
          }
            socket.emit('log', array);
        }

        socket.on('message', function (message) {
                log('Recebida mensagem: ', message);
                socket.broadcast.emit('message', message); //broadcast ou in(<sala>)
                //socket.in(keepsala).emit('message', message);
        });

        socket.on('criar ou juntar', function (sala) {
                keepsala = sala;
                var numClientes = io.sockets.clients(sala).length;

                log('Sala ' + sala + ' tem ' + numClientes + ' cliente(s)');
                log('Pedido para criar ou entrar em sala', sala);

                if (numClientes == 0){ //se não tem ninguém, cria sala
                        socket.join(sala);
                        socket.emit('criada', sala);
                } else if (numClientes == 1) { //se existe um parceiro na sala, entra na sala criada
                        io.sockets.in(sala).emit('entrar', sala);
                        socket.join(sala);
                        socket.emit('entrou', sala);
                } else { // dado que no máximo só podem haver dois parceiros por sala, se
existirem dois parceiros na sala, a sala está cheia
                        socket.emit('cheia', sala);
```

```
            }
            socket.emit('emit(): cliente ' + socket.id + ' entrou na sala sala ' + sala);
            socket.broadcast.emit('broadcast(): cliente ' + socket.id + ' entrou na sala ' + sala);

        });

});
```

H. Comunicação entrada PC1

Segue-se o detalhe dos pacotes de rede capturados da comunicação aquando da entrada do PC1.

```
  845 13.7647750 192.168.1.70   192.168.1.68   WebSock  103 WebSocket Text [FIN]
  846 13.7664860 192.168.1.68   192.168.1.70   WebSock  118 WebSocket Text [FIN]
  847 13.7671040 192.168.1.68   192.168.1.70   WebSock  134 WebSocket Text [FIN]
  848 13.7679490 192.168.1.68   192.168.1.70   WebSock   90 WebSocket Text [FIN]
  849 13.7688550 192.168.1.68   192.168.1.70   WebSock  129 WebSocket Text [FIN]
  857 15.8225140 192.168.1.70   192.168.1.68   WebSock  108 WebSocket Text [FIN]
  858 15.8230800 192.168.1.68   192.168.1.70   WebSock  131 WebSocket Text [FIN]
```

```
Frame 845: 103 bytes on wire (824 bits), 103 bytes captured (824 bits) on interface 0
Ethernet II, Src: 68:17:29:fa:c9:f4 (68:17:29:fa:c9:f4), Dst: 00:1d:60:c3:73:b1 (00:1d
Internet Protocol Version 4, Src: 192.168.1.70 (192.168.1.70), Dst: 192.168.1.68 (192.
Transmission Control Protocol, Src Port: 53632 (53632), Dst Port: 80 (80), Seq: 483, A
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  1... .... = Mask: True
  .010 1011 = Payload length: 43
  Masking-Key: d6c8dc87
⊞ Payload
⊟ Unmask Payload
    [Text unmask: 5:::{"name":"criar ou juntar","args":["1"]}]
```

```
Frame 846: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 68:17:29:fa:c9:f4 (68:1
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.70 (192
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53632 (53632), Seq: 135,
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  0... .... = Mask: False
  .011 1110 = Payload length: 62
⊟ Payload
    Text: 5:::{"name":"log","args":[[">>> ","Sala 1 tem 0 cliente(s)"]]}
```

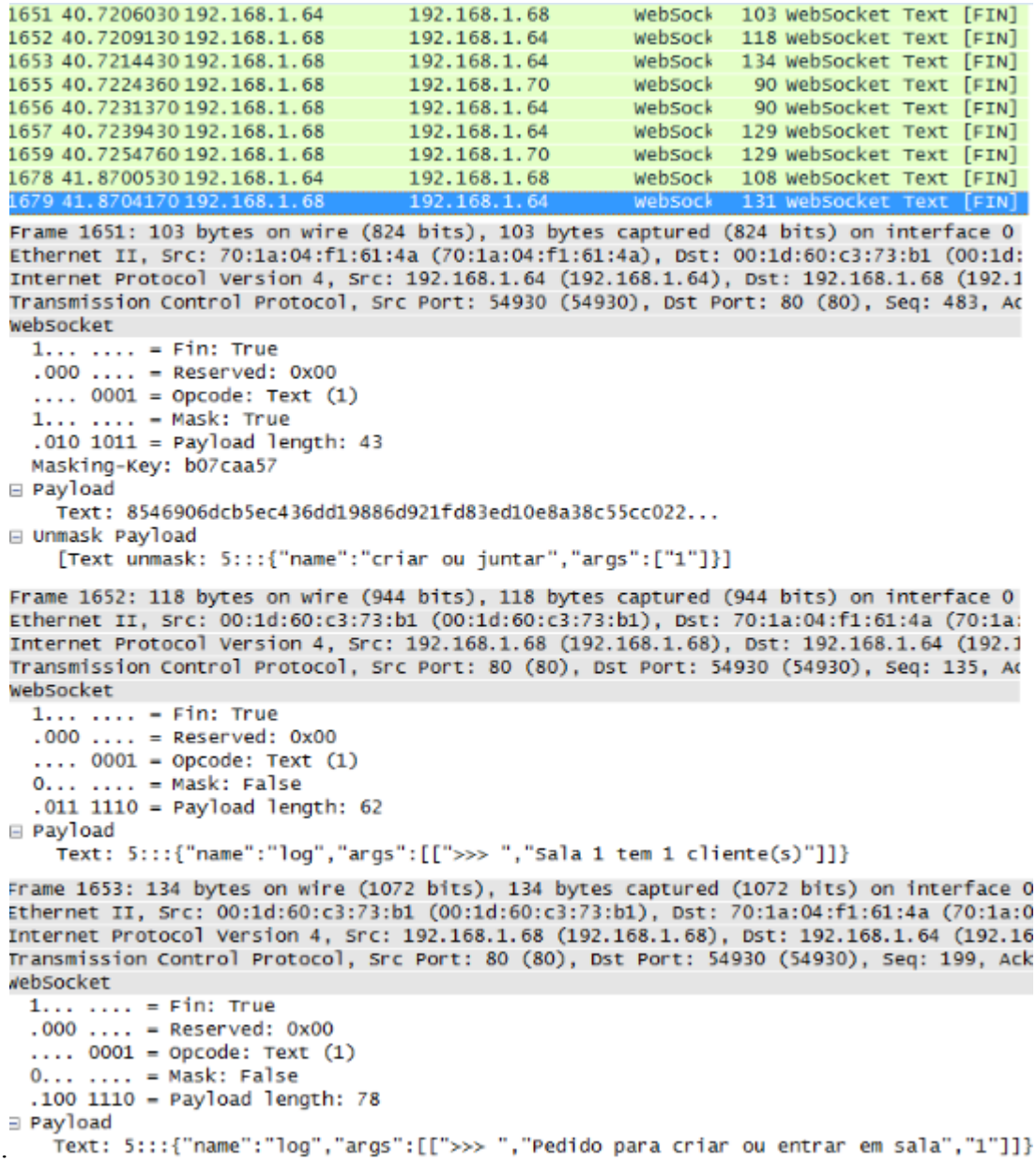
```
Frame 847: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 68:17:29:fa:c9:f4 (68:17:
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.70 (192.1
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53632 (53632), Seq: 199, Ac
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  0... .... = Mask: False
  .100 1110 = Payload length: 78
⊟ Payload
    Text: 5:::{"name":"log","args":[[">>> ","Pedido para criar ou entrar em sala","1"]]
```

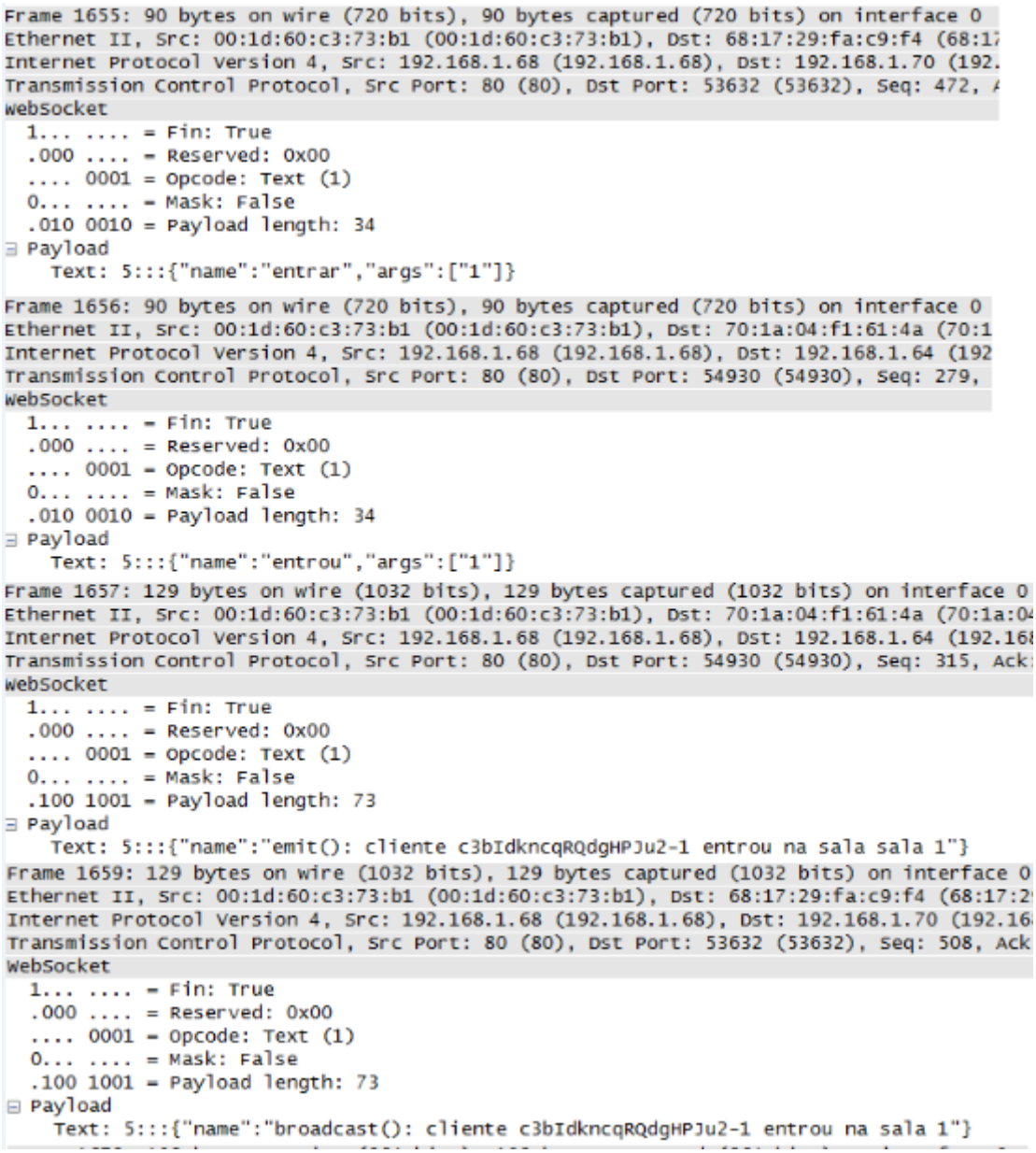
```
Frame 848: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 68:17:29:fa:c9:f4 (68:
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.70 (19
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53632 (53632), Seq: 279,
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  0... .... = Mask: False
  .010 0010 = Payload length: 34
⊟ Payload
    Text: 5:::{"name":"criada","args":["1"]}
```

```
Frame 849: 129 bytes on wire (1032 bits), 129 bytes captured (1032 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 68:17:29:fa:c9:f4 (68:17:2
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.70 (192.16
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53632 (53632), Seq: 315, Ack
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  0... .... = Mask: False
  .100 1001 = Payload length: 73
⊟ Payload
    Text: 5:::{"name":"emit(): cliente EJ9OXjRk9k_wfFTsu2-O entrou na sala sala 1"}
Frame 857: 108 bytes on wire (864 bits), 108 bytes captured (864 bits) on interface 0
Ethernet II, Src: 68:17:29:fa:c9:f4 (68:17:29:fa:c9:f4), Dst: 00:1d:60:c3:73:b1 (00:1d
Internet Protocol Version 4, Src: 192.168.1.70 (192.168.1.70), Dst: 192.168.1.68 (192.
Transmission Control Protocol, Src Port: 53632 (53632), Dst Port: 80 (80), Seq: 532, A
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  1... .... = Mask: True
  .011 0000 = Payload length: 48
  Masking-Key: 3b5a4c96
⊟ Payload
    Text: 0e6076ac407822f7563f6eac193729e5483b2bf319766ef7...
⊟ Unmask Payload
    [Text unmask: 5:::{"name":"message","args":["got user media"]}]
Frame 858: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 68:17:29:fa:c9:f4 (68:17:2
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.70 (192.16
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53632 (53632), Seq: 390, Ack
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  0... .... = Mask: False
  .100 1011 = Payload length: 75
⊟ Payload
    Text: 5:::{"name":"log","args":[[">>> ","Recebida mensagem: ","got user media"]]}
```

I. Comunicação entrada PC2

Segue-se o detalhe dos pacotes de rede capturados da comunicação aquando da entrada do PC2.



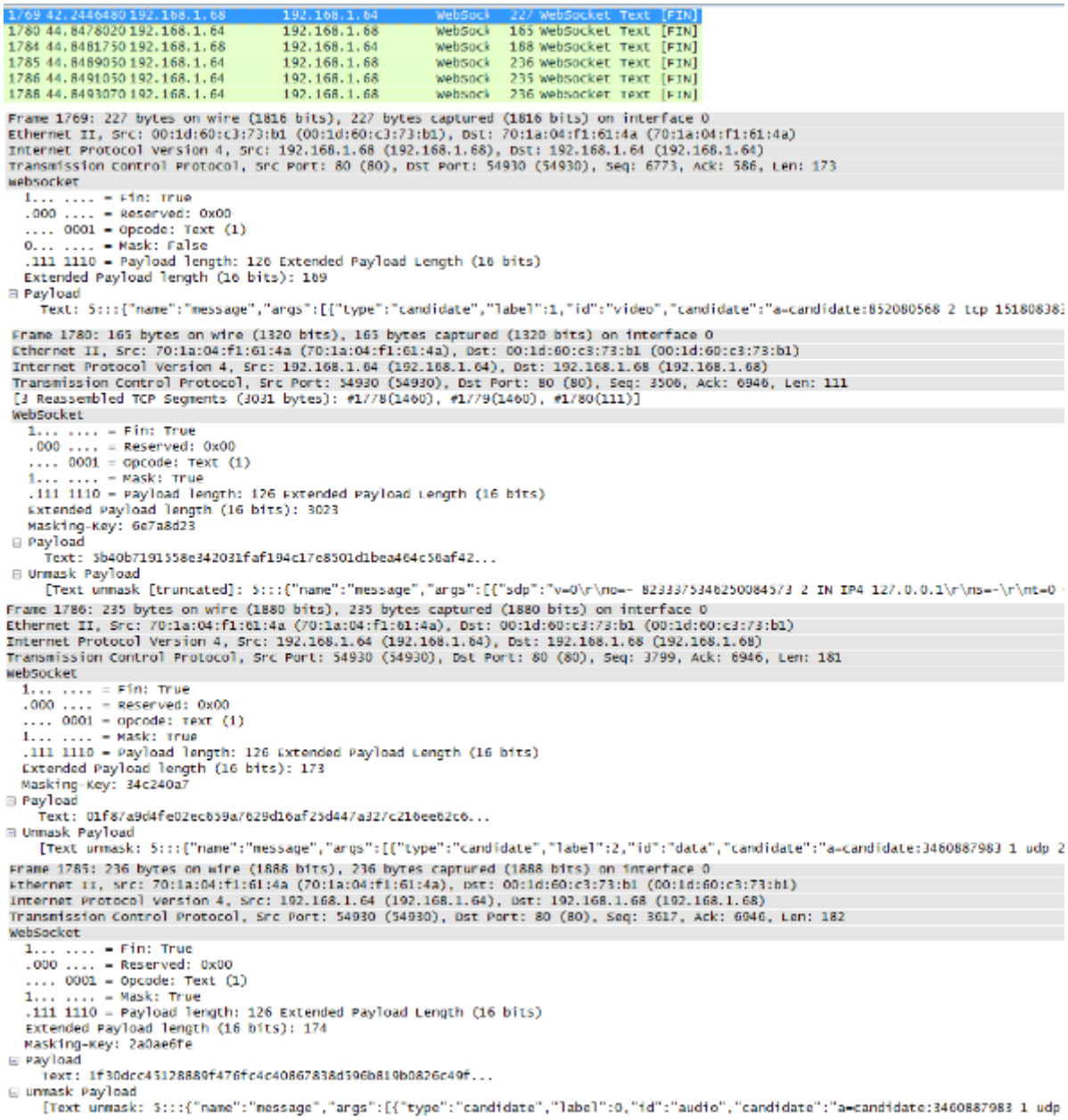

Frame 1655: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 68:17:29:fa:c9:f4 (68:17
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.70 (192.
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53632 (53632), Seq: 472,
WebSocket
1... .... = Fin: True
.000 .... = Reserved: 0x00
.... 0001 = Opcode: Text (1)
0... .... = Mask: False
.010 0010 = Payload length: 34
Payload
Text: 5:::{"name":"entrar","args":["1"]}
Frame 1656: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 70:1a:04:f1:61:4a (70:1
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.64 (192
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 54930 (54930), Seq: 279,
WebSocket
1... .... = Fin: True
.000 .... = Reserved: 0x00
.... 0001 = Opcode: Text (1)
0... .... = Mask: False
.010 0010 = Payload length: 34
Payload
Text: 5:::{"name":"entrou","args":["1"]}
Frame 1657: 129 bytes on wire (1032 bits), 129 bytes captured (1032 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 70:1a:04:f1:61:4a (70:1a:04
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.64 (192.168
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 54930 (54930), Seq: 315, Ack
WebSocket
1... .... = Fin: True
.000 .... = Reserved: 0x00
.... 0001 = Opcode: Text (1)
0... .... = Mask: False
.100 1001 = Payload length: 73
Payload
Text: 5:::{"name":"emit(): cliente c3bIdkncqRQdgHPJu2-1 entrou na sala sala 1"}
Frame 1659: 129 bytes on wire (1032 bits), 129 bytes captured (1032 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 68:17:29:fa:c9:f4 (68:17:2
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.70 (192.16
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53632 (53632), Seq: 508, Ack
WebSocket
1... .... = Fin: True
.000 .... = Reserved: 0x00
.... 0001 = Opcode: Text (1)
0... .... = Mask: False
.100 1001 = Payload length: 73
Payload
Text: 5:::{"name":"broadcast(): cliente c3bIdkncqRQdgHPJu2-1 entrou na sala 1"}

```
Frame 1678: 108 bytes on wire (864 bits), 108 bytes captured (864 bits) on interface 0
Ethernet II, Src: 70:1a:04:f1:61:4a (70:1a:04:f1:61:4a), Dst: 00:1d:60:c3:73:b1 (00:1d:6
Internet Protocol Version 4, Src: 192.168.1.64 (192.168.1.64), Dst: 192.168.1.68 (192.16
Transmission Control Protocol, Src Port: 54930 (54930), Dst Port: 80 (80), Seq: 532, Ack
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  1... .... = Mask: True
  .011 0000 = Payload length: 48
  Masking-Key: 2a9f9394
⊟ Payload
    Text: 1fa5a9ae51bdfdf547fab1ae08f2f6e759fef4f108b3b1f5...
⊟ Unmask Payload
    [Text unmask: 5:::{"name":"message","args":["got user media"]}]
```

```
Frame 1679: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 70:1a:04:f1:61:4a (70:1a:04
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.64 (192.168.
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 54930 (54930), Seq: 390, Ack:
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  0... .... = Mask: False
  .100 1011 = Payload length: 75
⊟ Payload
    Text: 5:::{"name":"log","args":[[">>> ","Recebida mensagem: ","got user media"]]}
```

J. Comunicação entre PC1 e PC2

Segue-se o detalhe dos pacotes de rede capturados da comunicação entre PC1 e PC2.



K. Mensagem fim de comunicação

Segue-se o detalhe dos pacotes de rede capturados da comunicação aquando o fim da comunicação.

```
1946 78.1367230 192.168.1.68      192.168.1.70      WebSock   120 WebSocket Text [FIN]
1947 78.1377610 192.168.1.68      192.168.1.64      WebSock    93 WebSocket Text [FIN]

Frame 1946: 120 bytes on wire (960 bits), 120 bytes captured (960 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 68:17:29:fa:c9:f4 (68:17:2
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.70 (192.16
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53632 (53632), Seq: 12340, A
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  0... .... = Mask: False
  .100 0000 = Payload length: 64
⊟ Payload
    Text: 5:::{"name":"log","args":[[">>> ","Recebida mensagem: ","bye"]]}
```

```
1947 78.1377610 192.168.1.68      192.168.1.64      WebSock    93 WebSocket Text [FIN]

Frame 1947: 93 bytes on wire (744 bits), 93 bytes captured (744 bits) on interface 0
Ethernet II, Src: 00:1d:60:c3:73:b1 (00:1d:60:c3:73:b1), Dst: 70:1a:04:f1:61:4a (70:1a:04
Internet Protocol Version 4, Src: 192.168.1.68 (192.168.1.68), Dst: 192.168.1.64 (192.16
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 54930 (54930), Seq: 11931, A
WebSocket
  1... .... = Fin: True
  .000 .... = Reserved: 0x00
  .... 0001 = Opcode: Text (1)
  0... .... = Mask: False
  .010 0101 = Payload length: 37
⊟ Payload
    Text: 5:::{"name":"message","args":["bye"]}
```

L. Código da aplicação multi-parceiro

```
<!DOCTYPE html>
<html>
	<head>
		<script src="http://simplewebrtc.com/latest.js"></script>
		<title>MSTIO - Roberto Oliveira Rocha 2012/2014 - Aplicação WebRTC
MultiParceiro</title>
		<style>
		#VideoLocal{
			min-width: 25%;
			width: 25%;
			max-width: 25%;
			margin:auto;}
		#VideosParceiros{
			min-width: 100%;
			width: 100%;
			max-width: 100%;
			margin:auto;}
		#VideosParceiros video{
			min-width: 24%;
			width: 24%;
			max-width: 24%;}
		</style>
	</head>
	<body>
		<div id="VideosParceiros"></div>
		<div id="VideoLocal" muted></div>
		<script>
		var webrtc = new SimpleWebRTC({
			localVideoEl: 'VideoLocal',
			remoteVideosEl: 'VideosParceiros',
			autoRequestMedia: true});
		webrtc.on('readyToCall', function () {
			room = prompt('Que sala que pretende aceder?');
			webrtc.joinRoom(room);});
		</script>
	</body>
</html>
```