

WebRTC

Mariana Oliveira, 42355
LEIM 51N
RSCM, ADEETC, ISEL
Lisboa, Portugal
A42355@alunos.isel.pt

Tomás Carvalho, 42357
LEIM 51N
RSCM, ADEETC, ISEL
Lisboa, Portugal
A42357@alunos.isel.pt

Docentes:
Nelson Costa
RSCM, ADEETC, ISEL

Abstract—*WebRTC é uma tecnologia de comunicação em tempo real embutida no browser. O desenvolvimento do projeto começou em 2010 na Google, e passou pela adesão de diversos browsers. É uma comunicação que faz uso de um canal direto entre endpoints. Aplica uma série de protocolos para garantir a maior segurança e fiabilidade possível. As quais, apesar dos esforços, continuam a ser grandes desvantagens da implementação. Foi, no entanto, bem aceite pelos donos de mercado. E, atualmente, é uma tecnologia muito utilizada no seu setor.*

Keywords—*webrtc, comunicação, internet, browser, endpoint, servidor, audiovisual, open-source, empresas.*

I. INTRODUÇÃO

WebRTC (ou Web Real-Time Communication) é um projeto *open-source* que possibilita a comunicação (em tempo real) de áudio e vídeo, bem como a transferência de dados entre browsers ou aplicações nativas. Utiliza uma arquitetura *peer-to-peer* (P2P), sem necessidade de downloads, executáveis ou plugins. É uma especificação de HTML5 composta por várias APIs^[1] escritas em JavaScript (ao contrário da comunicação audiovisual tradicional, que utilizava C/C++).

Atualmente, para além do *video-chat*, permite *multi-party calling*, partilha de dados não audiovisuais e gravação e partilha de ecrãs/*screenshots* com armazenamento dos mesmos. Sendo utilizado na maioria dos browsers e por grande parte dos criadores de aplicações móveis de videochamadas, tem vindo a ganhar uma percentagem do mercado cada vez maior.

II. HISTÓRIA

A. Pré WebRTC

A GIPS – Global IP Solutions, foi fundada em Estocolmo em 1999. Com a premissa de resolver problemas de *delay* e *packet loss* em comunicações VoIP (na altura, a tecnologia usada pelas redes *circuit switched*^[2] tradicionais não estava preparada para comunicação por IP). Foram criadores de várias componentes adotadas, mais tarde, pela Google, no projeto WebRTC, inclusive diversos codecs de comunicação. Exemplos disso são a VoiceEngine^[3], a VideoEngine^[4] e os codecs narrowband iLBC e wideband iSAC.

A ON2, fundada em Nova Iorque em 1992, foi uma empresa especializada no desenvolvimento de codecs de vídeo. Criaram uma série de codecs chamada TrueMotion (destaque para o VP8).

B. Aquisição pela Google e Projeto WebRTC

Em 2010 a Google adquiriu a On2 e a GIPS. No mesmo ano lançou o WebM (projeto baseado na On2). Um formato de ficheiro que usava VP8 para substituir o vídeo/áudio disponibilizado pelo HTML5. Desenvolveu, também, como parte do projeto, o VP9, nova geração do codec de vídeo mencionado antes.

Em 2011 lançou o WebRTC (baseado em tecnologia GIPS). Incluído no projeto estava a VoiceEngine (juntamente com o AEC – cancelamento de eco – e a NetEQ – para baixar a latência e mantém QoS), a VideoEngine e os vários codecs criados pela GIPS. Tanto o WebM como o WebRTC foram disponibilizados em *open-source*.

A Google começou a trabalhar com organizações como a IETF^[5], a W3C^[6], ou a WHATWG^[7], para garantir o estabelecimento de normas para o projeto.

Ainda em 2011, foi desenvolvida a primeira implementação, pela Ericsson (empresa de telecomunicações sueca), usando uma biblioteca modificada do WebKit (motor de busca da Apple). Mais tarde (2014), viriam a criar e lançar a OpenWebRTC, como alternativa para iOS e OS X (macOS) (também preparado para Linux e Android).

Seguiu-se a adesão por parte dos principais browsers:

- 2011
 - Chrome;
- 2013
 - Firefox;
 - Chrome e Firefox para Android;
 - **Implementação da capacidade de comunicação entre browsers (inicialmente só Chrome e Firefox).**
- 2014
 - Opera;
 - Microsoft adiciona capacidades de ORTC^[8] ao Internet Explorer;
- 2015
 - Microsoft adiciona capacidades de ORTC ao Edge;
- 2017
 - Com o Safari 11, a Apple, finalmente adiciona capacidades WebRTC ao seu browser;

- Microsoft adiciona capacidades WebRTC ao Edge.

III. FUNCIONAMENTO

A. Comunicação e Codecs

Sem WebRTC, a comunicação "em tempo real" é feita através de web servers. O primeiro *endpoint* comunica com o seu servidor, este encaminha a informação para o servidor do *endpoint* de destino que, por sua vez, encaminha para o seu *endpoint* (ou, no caso de partilharem o mesmo servidor, *endpoint1* envia para servidor partilhado que reenvia para *endpoint2*).

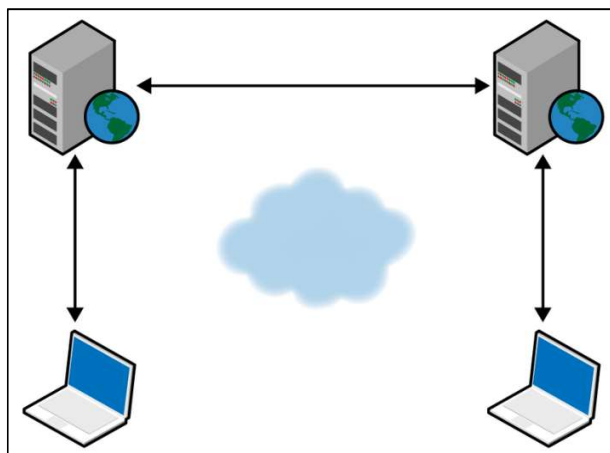


Figura 1 - Comunicação sem WebRTC

Nesta situação existe muito *delay* e latência, para além de ser virtualmente impossível atingir a comunicação em tempo real (RTC).

Um dos principais argumentos dos defensores do WebRTC é o facto de que um atraso de um mero segundo em videochamadas é o suficiente para destabilizar a conversação. E, enquanto isto não é grave em chamadas sociais, pode ser um fator crucial em comunicações empresariais ou de assistência técnica.

Com o WebRTC, é estabelecido um canal direto entre os *endpoints*. Os quais, a partir daí, comunicam sem intervenção do(s) servidor(es). Assim, o primeiro passo para comunicação com WebRTC é o estabelecimento do canal virtual. Visto que o mais provável não é que os *endpoints* estejam a comunicar na mesma rede local, mas sim em pontos remotos da internet, precisam de se conhecer/localizar primeiro. Isto é feito com recurso a um processo chamado de *signaling*. Os *endpoints* comunicam com um *signaling server*, que vai dar a conhecer a cada uma informação necessária ao estabelecimento da ligação direta. O servidor vai facilitar o estabelecimento, manutenção e fecho do canal de comunicação. Durante este processo são trocados três tipos de informação:

- Session Control Information: parte administrativa de criação, modificação, fecho da ligação e gestão de erros;

- Network Data: dá a conhecer os endereços IP (internos e externos), endereços MAC, portos de acesso, NATs, etc.;
- Media Data: negociação de codecs a usar na comunicação. Esta troca de informação é feita segundo o formato SDP.

Visto que o WebRTC não implementa nenhum método de *signaling*, este passo é deixado ao encargo do criador da aplicação ou dos *standards* sugeridos pelos browsers. Os principais métodos usados para a comunicação com o *signaling server* são o XMLHttpRequest (HTTP) ou o WebSocket^[9]. Sendo as principais diferenças (a favor) do WebSocket para o HTTP tradicional, o *full-duplex* e a comunicação bi-direcional em lugar do habitual *request-response*. *Signaling*, aqui, trata de estabelecer a ligação descrita anteriormente (sem WebRTC: e1 - s1 - s2 - e2). Feitas as conciliações e trocada a informação necessária será estabelecido o **canal direto**. Um **canal sem interferência de nenhum servidor**, visto que a informação é trocada diretamente entre *endpoints*, sem processamento ou *relay*. O que vai permitir uma comunicação mais segura. E, ao contrário do processo de *signaling*, a **criptação** dos dados trocados através do canal faz parte da implementação do WebRTC. Com segurança acrescida se for uma comunicação a partir do browser (por oposição a aplicações móveis ou de *desktop*). Dado que **corre na sandbox**^[10] do browser, não será uma ameaça para o sistema local. Outra característica do canal WebRTC é o facto de correr sobre UDP, por oposição a TCP (ligação mais fiável). Sendo o objetivo enviar *data* rapidamente (RTC), escolheu-se usar **UDP**. Como este é *connectionless*, não irá tentar reenviar *packets* perdidos, como seria o caso do TCP. Nesta situação a habitual vantagem do TCP verificar *packets* perdidos ou corrompidos e garantir o reenvio iria criar *jitter*, *delay* e levar à eventual quebra da ligação. Desta forma, é preferível não haver preocupação com qualidade ou perda de informação e, simplesmente, entregar o máximo de informação o mais rápido possível. Para justificar esta implementação, o WebRTC está, ainda, equipado com métodos para garantir a melhor QoS possível.

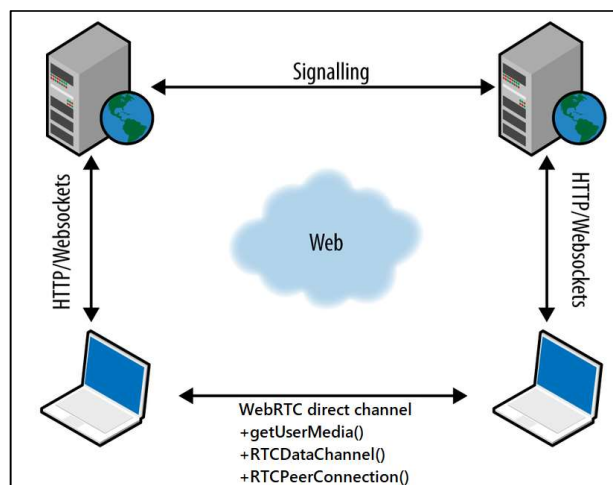


Figura 2 - Comunicação com WebRTC

Aqui, finalmente, faz-se uso das APIs referidas. Enquanto o *signaling* facilita a troca de *metadata* necessária à gestão da ligação, o `RTCPeerConnection()` estabelece a referida ligação e coordena a troca de dados através da mesma.

O projeto WebRTC disponibiliza uma série de APIs que permitem ou facilitam a sua utilização, sendo as seguintes as mais cruciais:

- **RTCPeerConnection()**: funciona de maneira semelhante ao *signaling*, no sentido em que gere a ligação entre *endpoints* remotos, trocando os mesmos tipos de informação (Session Control, Network e Media). É com esta API que se estabelece realmente a ligação;
- **getUserMedia()**: ganha acesso microfone, câmara e ecrã (para partilha de ecrã) do computador;
- **RTCDATAChannel()**: trata da transmissão de *data* não audiovisual.

Tanto a API `getUserMedia()`, como a `RTCDATAChannel()` correm sobre a `RTCPeerConnection()`.

Existe, no entanto, uma ameaça a esta ligação direta – **NATs** (Network address Translation) e **firewalls**. Em redes fechadas ou sem existência de NATs, o funcionamento é como descrito anteriormente. Mas, novamente, o mais provável é a comunicação ser feita entre *endpoints* localizados em pontos distantes na rede global. Comunicação entre *endpoints* que não estejam na mesma LAN (nomeadamente, na internet), envolve mais uns passos.

O WebRTC já prevê que o SDP não conhece os IPs externos das NATs ou as restrições de portos (esta informação não será trocada sem ajuda). Assim, é implementado o protocolo ICE (Interativa Connectivity Establishment), um método para encontrar a melhor ligação (mais) **direta** possível entre dois *endpoints*. O objetivo é estabelecer a intendida comunicação sem intervenção de servidor. O ICE vai gerar uma lista de possíveis candidatos para *media traversal* que podem ser usados em aplicações de WebRTC para atravessar NATs/firewalls. Os mais utilizados são:

- **STUN** (Session Traversal Utilities for NAT): permite aos clientes descobrirem o seu IP externo e tipo de NAT atrás do qual estão. Podendo informar o respetivo utilizador remoto com quem estão a tentar iniciar ligação, para se conectarem através do IP externo, através do router (NAT) e chegando ao IP interno final;

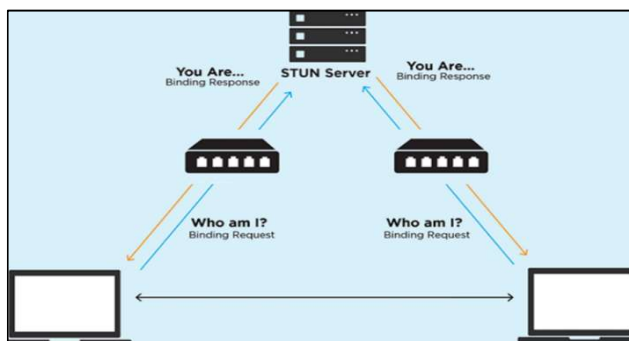


Figura 3 - Funcionamento de Servidor STUN

- **TURN** (Traversal Using Relays around NAT): em casos “extremos” de restrições organizacionais ou de rede, *firewalls* ou falhas na ligação direta é usado outro método. TURN consiste no *relay* da informação através dos servidores designados. Aqui, a palavra chave é “around”.

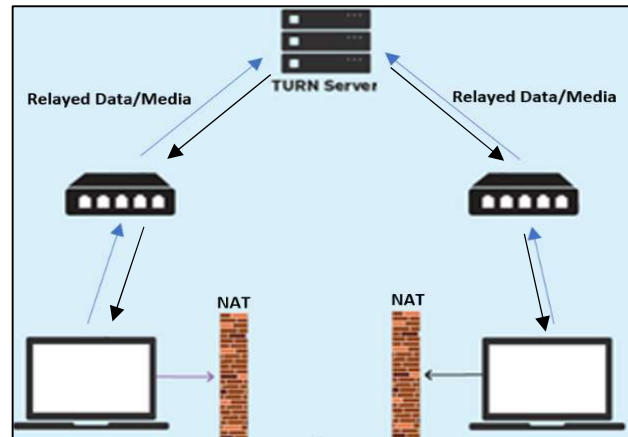


Figura 4 - Funcionamento de Servidor TURN

Como mencionado, durante o estabelecimento da ligação dá-se, também, a negociação dos codecs. Esta negociação é feita de uma lista pré-estabelecida de codecs aceites pelo WebRTC.

Verifica-se que os **codecs utilizados[anexo]** atualmente no projeto não diferem muito dos utilizados (ou desenvolvidos) pela GIPS e pela On2:

- Codecs de Vídeo
 - VP8: é um codec *open-source*, desenvolvido pela On2. Faz parte da WebM e foi o codec raiz do WebRTC. É suportado por todos os browsers que implementam WebRTC.
 - VP9: desenvolvido a partir do VP8 está disponível no Chrome e Firefox.
 - H.264 (AVC): ao contrário dos anteriores, não é *open-source*. Exatamente por essa razão é um problema para o término do desenvolvimento da API (WebTRC APIs). Visto que, independentemente da comunicação sem servidores ser mais barata, nalgumas situações não compensa o uso de *software* proprietário.
- Codecs de Áudio:
 - Opus: usado no Edge, Firefox e Safari. É um codec *open-source* baseado nos codecs CELT (Xiph.Org) e SILK (Skype).
 - G.711 PCM (a/μ law): também *open-source*, é o codec base da comunicação, visto ser o de pior qualidade.
 - G.722: codec *Open-source*, baseado no G.711.
 - iLBC (internet Low Bitrate Codec): desenvolvido pela GIPS. Usa uma licença

BSD^[11]. Não é aceite em todos os browsers.

- o iSAC (internet Speech Audio Codec): também original da GIPS. Também usa uma licença BSD. E também não é aceite em todos os browsers.

Todos os codecs de áudio, bem como o H.264 são *lossy* (por oposição a codecs *lossless*). Isto significa que usam aproximações na compressão da *media* transmitida e não se preocupam em transmitir as variações não compreendidas pelos olhos/ouvidos humanos.

B. Desenvolvimento de Aplicações

Do ponto de vista do criador não é excessivamente complicado. Com conhecimento básico de redes e servidores e de programação (JavaScript e HTML + CSS), é bastante simples e rápido construir uma interface modesta de comunicação WebRTC. Como o WebRTC é uma PaaS^[12], é uma questão de incluir as APIs vistas anteriormente (pelo menos as principais. Quaisquer funcionalidades adicionais requerem algum trabalho de pesquisa) necessárias para as funcionalidades pretendidas e ter em consideração os *standards* do projeto (seja do próprio WebRTC, seja do browser utilizado ou do sistema operativo).

Procedeu-se a um pequeno **tutorial** (muito simplificado, só para demonstrar as APIs `RTCPeerConnection()` e `getUserMedia()`), onde foi usado código já desenvolvido (para testes em *localhost*), visto ser a parte mais trabalhosa e que foge um pouco à área de redes em questão.

Usou-se uma arquitetura simples, com um computador a fazer de servidor para alojar o website e outros dois computadores, os clientes, que acedem ao servidor e estabelecem a ligação direta entre eles, como explicado no funcionamento. Foram escolhidas as seguintes ferramentas:

- Node.js^[13] para capacidades de servidor
- ngrok.exe^[14] para criação do link de acesso ao servidor

Tanto o Node.js como o ngrok.exe são usados no “servidor”. Nos clientes apenas foi necessário Chrome e Firefox para verificar a comunicação *inter-browser*. Procedeu-se ao desenvolvimento:

1. Criou-se uma pasta de trabalho – *app*;
2. Na pasta *app*, criou-se uma subpasta *public*;
3. Onde foram criados os ficheiros:
 - a. *index.html* – a *interface* do website/aplicação. Onde são definidos os *containers* necessários para o *display* de vídeo, imagens recebidas, etc. Não foi usado CSS neste tutorial;
 - b. *client.js* – ficheiro com as APIs necessárias em cada cliente. Gere ofertas e respostas. Cria variáveis globais para guardar o número da *chatroom*, os *endpoints* ligados, *streams* de áudio e vídeo locais e remotos. Lida ainda com a `RTCPeerConnection()` e com as ligações (do respetivo cliente) aos servidores;
4. Criou-se o ficheiro *server.js* na raiz: é aqui que é especificado como é feito o *signaling*. São geridos os

packages a instalar, nomeadamente o *express*^[15] e *socket.io*^[16], e registados os clientes conectados (o ficheiro usado foi alterado para permitir mais do que um cliente);

5. Na linha de comandos acedeu-se à diretoria da pasta raiz para inicializar e instalar os *packages* necessários:
 - a. `npm[17] init -y`
 - b. `npm install -S express@4.15.4 socket.io@2.0.3`
 - i. pode ser necessário correr o comando ‘`npm audit fix`’, no caso de se receber uma mensagem a indicar incompatibilidades na instalação;
6. De seguida fez-se o *upload* do programa:
 - a. Neste caso, inicializou-se o servidor no porto especificado (3000) da máquina que vai funcionar como servidor. Na linha de comandos, na diretoria do projeto introduzir ‘`node server.js`’;

```
C:\Users\tomas\Desktop\app>node server.js
listening on 3000
a user connected
a user connected
a user connected
create or join to room 1
1 has 0 clients
create or join to room 1
1 has 1 clients
create or join to room 1
1 has 2 clients
a user connected
create or join to room 1
1 has 3 clients
```

Figura 6 - Inicialização e gestão de registos no servidor

- b. E criou-se o link de acesso ao website. Novamente, na linha de comandos (noutra instância), ‘`ngrok http localhost:3000`’.

```
Session Status      online
Session Expires    7 hours, 52 minutes
Update             update available (version 2.3.35, Ctrl-U to update)
Version            2.3.29
Region             United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding          http://6040ca45.ngrok.io -> http://localhost:3000
                   https://6040ca45.ngrok.io -> http://localhost:3000

Connections
  ttl  opn  rt1  rt5  p50  p90
   37   0   0.02 0.05 0.99 102.24

HTTP Requests
-----
GET /socket.io/      101 Switching Protocols
GET /socket.io/      200 OK
GET /socket.io/      200 OK
GET /socket.io/socket.io.js 304 Not Modified
GET /client.js       304 Not Modified
GET /                 304 Not Modified
GET /socket.io/      101 Switching Protocols
GET /socket.io/      200 OK
```

Figura 7 - Criação e gestão de acesso ao link

C. Utilização de Aplicações

Ainda, no tutorial, testou-se o funcionamento da aplicação. Experimentou-se uma comunicação entre dois clientes (em redes diferentes).

Do ponto de vista do utilizador é uma questão de aceder ao link da aplicação (website). No website, cada cliente deve escolher o número (ou nome) da *chatroom* (deve ser a mesma nos diferentes clientes), previamente estabelecida para garantir comunicação. De seguida irá aparecer um *pop-up* com os pedidos de acesso, nomeadamente ao microfone e câmara do dispositivo (este é o trabalho da API `getUserMedia()`, visto ser necessário recetores e transmissores em cada cliente). Os clients aceitam os pedidos e a ligação é estabelecida – ligação direta sem que o servidor tenha conhecimento de qualquer informação trocada pelos clientes, apenas registos e acessos.

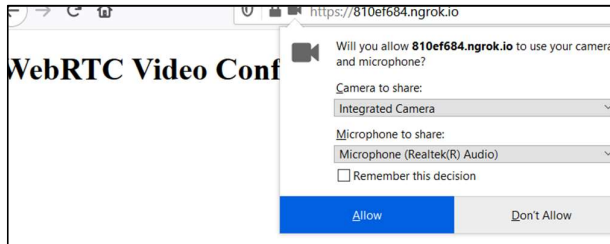


Figura 8 - Pedidos de permissão no Firefox

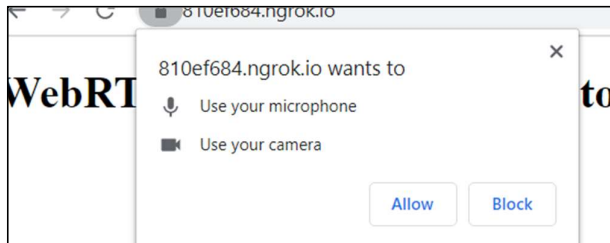


Figura 9 - Pedidos de permissão no Chrome

IV. APLICAÇÕES E ÁREAS DE MERCADO

A. Tipos de Aplicações e Casos de Uso

Existem duas categorias principais de empresas ou aplicações que fazem uso do WebRTC:

- *Developers* intermediários: desenvolvem aplicações standardizadas (PaaS) ou a pedido para outrem. Muitas empresas com conhecimentos de telecomunicações WebRTC e programação (mas, talvez, sem plataforma ou setor de implementação) escolhem facultar as suas soluções de comunicação RTC a empresas mais focadas noutras áreas e com menos conhecimento/recursos para o desenvolvimento de aplicações de videochamada;
- *Developers* finais: aplicações desenvolvidas para uso da própria empresa ou para distribuição ao público. Empresas focadas em oferecer plataformas de videochamadas (sociais ou de teleconferência) ao público ou empresas do setor tecnológico/de telecomunicações que preferem a sua própria implementação nos seus projetos (mesmo em aplicações de uso próprio e não para distribuição)

O projeto WebRTC já disponibiliza todos os recursos de maneira organizada, prontos para facilitar o desenvolvimento da aplicação, num conjunto de APIs (*building-blocks*), deixando os *developers* com um trabalho mais facilitado e mais possibilidade de foco em melhorias de fiabilidade,

design, segurança, etc. No entanto, algumas empresas oferecem esse know-how necessário para fazer determinadas especificações ou melhorias às empresas com falta do mesmo. Alguns exemplos de empresas que desenvolvem *software* para terceiros são:

- webRTC.ventures oferece soluções de áudio e vídeo em todas as áreas da comunicação audiovisual;
- VeriShow foca-se em soluções de *co-browsing* para assistência ao cliente.

Tanto as empresas mencionadas anteriormente, como as que serão agora listadas (que fazem uso da tecnologia para o próprio desenvolvimento) lançaram aplicações em diversas áreas do mercado. Sendo algumas das principais (áreas):

- Telecomunicações: aplicações de chamadas áudio/vídeo (VoIP) já são bem conhecidas. E, desde 2010, têm vindo a apostar cada vez mais na implementação da tecnologia WebRTC. Facebook Messenger, Mozilla Hello e Google Hangouts são alguns dos exemplos.
- IT: tem se vindo a apostar, também, em projetos de Internet of Things^[18] ou assistentes digitais. A Amazon tem vindo a implementar na maioria dos seus projetos, como Alexa, Chime ou o Mayday.
- Assistência ao cliente: há muita procura por soluções que facilitem as relações cliente-vendedor. Muitas empresas capitalizaram nesse sentido. A SightCall desenvolve soluções de assistência vendedor-cliente e vendedor-vendedor. Mais uma vez, a Amazon tem vindo a implementar WebRTC nas suas aplicações de assistência ao cliente.
- Saúde (Telehealth): inclui empresas como a MedZed que fornece consultas por videochamada.
- Educação (EdTech): o grupo COMPENTUM foca-se na área da educação online, aposta em explicações online, entrevistas a alunos, treino linguístico, etc.
- Entretenimento: não tão explícito como nas áreas anteriores, mas o WebRTC também começa a ingressar na indústria. Desde *live video streaming* a *gaming* começam a aparecer pequenas empresas (principalmente PaaS) dedicadas a melhorar as experiências de entretenimento. Mas, para além, da implementação óbvia na transmissão de áudio ou vídeo começa-se a testar as capacidades de transmissão de todo o tipo de *data*, incluindo não audiovisual (`RTCDataChannel()`). A GENBAND (empresa americana que desenvolve software de comunicação em tempo real) desenvolveu uma solução de escalabilidade – WebRTC Gateway, para comunicação P2P cliente-servidor, com o objetivo de criar experiências de jogo mais rápidas. Outro exemplo desta implementação (WebRTC em jogos) é o jogo The Hobbit: The Battle of the Five Armies.

Outras empresas fazem, ainda, uso, não da óbvia vantagem na comunicação por videochamada, mas de outras aplicabilidades menos implementadas ou com menos foco. São essas vertentes:

- **Armazenamento:** Pipe Recording (addpipe.com) foca-se na gravação e armazenamento de vídeo e áudio. Tem uma base de dados complexa preparada com tecnologia WebRTC para receber e armazenar quaisquer dados. Principalmente áudio, vídeo e *screenshots*. Tirando, assim, proveito de todas as capacidades do WebRTC;
- **Mensagens de Texto:** diversas *startups* e pequenas empresas focam-se inicialmente na comunicação por texto. Sendo bastante simples de implementar e, não tendo tantas implicações como vídeo ou áudio no que toca a codecs, é um bom ponto de partida;
- **Partilha de Ficheiros:** mais uma vez, tirando proveito da API de transmissão de dados não audiovisuais (RTCDataChannel()), há pequenas empresas só interessadas nessa vertente não audiovisual do WebRTC. Como é o caso da Blaze ou da FilePizza.

B. Alternativas e Desvantagens

Tal como as suas vantagens, também as desvantagens advêm do facto de ser uma tecnologia *open-source* e ainda não estar completamente afinada (esteve e, ainda vai estar, em período de testes durante muito tempo). Muitas empresas, como foi o caso da Apple (que durante muito tempo não aceitou completamente a tecnologia), ou a Cisco (que continua a tentar forçar o uso de codecs proprietários) continuam a ter reservas relativamente ao uso do WebRTC. Alguns grandes nomes continuam a recusar-se adotar a tecnologia, como o Skype ou o Whatsapp. Se bem que tanto a Microsoft como o Facebook já implementam WebRTC, portanto, talvez seja uma questão de não concentrar todos os recursos na mesma alternativa.

A questão do *open-source*, é uma preocupação para alguns por causa da encriptação e da segurança. É verdade que o WebRTC inclui APIs que lidam com essa parte. E é verdade que um canal dedicado sem interferência dos servidores é mais seguro do que um canal aberto com uma lista infundável de *hops* e de *hosts* com acesso à informação transmitida. Mas não é garantido que seja suficiente, dado que não existe nenhuma NAT ou *firewall* entre o utilizador local e o remoto, ao contrário da comunicação tradicional através de servidores. Com todas as *NAT traversals*, todo o contornar de *firewalls* e restrições, a dúvida levanta-se sobre as APIs de segurança. Em comunicação com servidores a segurança é garantida por esses mesmos servidores. Servidores controlados por empresas de renome já estabelecidas. Visto ser domínio público, não existe nenhuma corporação que possa garantir a 100% a sua fiabilidade, ou qualidade de serviço. Ao contrário do Skype ou do Flash que, apesar de estarem em decadência são “caras conhecidas” que podem dar garantias.

Esta será uma discussão infundável de risco/recompensa. O facto é que o ser humano tem medo do desconhecido. Por isso vai continuar a haver resistência à implementação de novas tecnologias.

Como referido, no caso da Cisco, coloca-se, ainda, o problema da integração. Sendo que algumas organizações levam mais tempo a aceitar e implementar a tecnologia, irá continuar a haver problemas de compatibilidade, o que já não existe em tecnologias estabelecidas, como VoIPs tradicionais. Especificamente, os codecs utilizados complicam a situação, dado que uma tecnologia *open-source* deveria usar codecs *open-source*, no entanto algumas companhias querem introduzir codecs proprietários.

O crescimento do WebRTC criou mercado para uma série de aplicações alternativas (competição). Algumas *startups*, outras já estabelecidas. Algumas com a sua própria tecnologia e outras que fazem uso do WebRTC. Novamente, o facto de ser disponibilizado permite a criação de outras *satecks*, *interfaces* ou APIs de RTC. Alguns exemplos são o ORTC, aplicado pela Microsoft, ou o OpenWebRTC, da Ericsson. No GitHub existem diversas bibliotecas que tentam implementar tecnologia RTC ou mesmo WebRTC, é o caso da autobahn ou a gentev. E, para os browsers que não permitem WebRTC, tem-se *plugins* ou extensões como o webrtc4all + sipml5 que equipam o browser com a tecnologia em questão. Outros exemplos são empresas/aplicações com métodos de RTC diferentes do discutido. Métodos como comunicação por servidores, já referidos, seja por WebSocket ou HTTP (ou HTTP/2^[19] (SPDY^[20])), seja *open-source* ou proprietária, é competição. É o caso do Skype, do Flash (RTMP^[21] + RTMFP^[22]), do Discord ou do WebTorrent (implementa *stream* de vídeo P2P em tempo real usando arquitetura de comunicação torrente).

V. FUTURO E CONCLUSÕES

O WebRTC é uma solução para um problema cada vez mais atual – entrega ou troca de informação de forma instantânea, sem atrasos, e segura. Muito mais agora do que em 2011, no princípio do projeto (e, já nessa altura, foi adotado rapidamente pela maioria dos donos de mercado), vivemos num mundo cada vez mais rápido e com tecnologias cada vez mais avançadas. É expectável que se necessite/queira opções de comunicação velozes e sem falhas (como argumentam os defensores do projeto – um atraso de um segundo pode ser crucial).

Atualmente existem dezenas de aplicações diferentes. Toda a explosão da área de IoT e evolução das comunicações VoIP criou um espaço na indústria. Acrescido do facto de ter sido disponibilizado em *open-source*, como foi dito antes, permitiu uma evolução exponencial das tecnologias (tanto do WebRTC, como das alternativas ou das tecnologias RTC).

Para além de todas as aplicações, listadas anteriormente, que implementam ou “revendem” a tecnologia, o GitHub regista mais de 13 mil repositórios, 6 milhões de excertos de códigos e mais de 5 milhões de *commits*^[23]. O Google regista mais 6.5 milhões de resultados, incluindo diversos tutoriais e palestras. Bem como repositórios da Google *codelabs* e do próprio projeto webrtc.org/. Isto reforça a ideia de que o projeto está a ser bem aceite pelo público, sejam utilizadores, sejam *developers* ou organizações. Estão todos a ajudar a tecnologia a evoluir.

Toda esta adesão por parte do público e da indústria bem como a criação de uma comunidade muito dinâmica e interessada no projeto que impulsionou o seu desenvolvimento cria um ciclo virtuoso de reforço positivo. Visto que toda a tecnologia, todos os tutoriais e código disponibilizado ajudam à criação de novas empresas, e estas empresas, bem como toda a comunidade de repositórios e códigos evoluem e pedem melhor tecnologia.

O WebRTC já está presente em diversas áreas de mercado, para além das telecomunicações, como verificado na listagem das empresas e setores do WebRTC (página 5). Cada vez mais setores reconhecem as aplicações do projeto e experimentam novas implementações. Uma tecnologia que só se imagina no setor tecnológico, está, de repente em áreas de educação e saúde. E todo o reconhecimento que se começa a dar às restantes APIs permite o aparecimento de um segmento de empresas razoavelmente diferente (seja armazenamento ou transferência de dados). Desta maneira, o mercado das tecnologias WebRTC não para de crescer.

Há 6 anos só tinham lugar no mercado norte americano. Mas com todo o reconhecimento, começaram a espalhar-se pelos principais mercados (europeu e sudeste asiático). E, atualmente, são tecnologias presentes e acessíveis em todo o mundo.

O que começou como uma série de codecs em empresas com pouco reconhecimento na indústria, envolve, agora, os maiores donos de mercado. É o caso da AT&T, Oracle, Cisco, Google, IBM, Apple, Microsoft e muitos outros.

Por todas estas razões a valorização de mercado das tecnologias WebRTC tem vindo a ser cada vez mais elevada. E pelas mesmas razões nas últimas atualizações de 2019 o *forecast* dos próximos 7 anos previa uma valorização de mais de 50 Bilhões \$US.

VI. ANEXO

	VoIP tradicional	WebRTC
<i>Signaling</i>	SIP H.323	
Transporte audiovisual	RTP RTCP	RTP RTCP
Segurança	SRTP(SIP) H.235(H323)	SRTP
NAT traversal	ICE(SIP) H.450.x(H.323)	ICE STUN/TURN
Codecs Áudio	H.263 H.264	VP8
Codecs Video	G.7XX	G.711 Opus iLBC iSAC

Table 1 - Resumo das principais diferenças entre VoIP tradicional e WebRTC

Audio						
Codec	Bitrate kbs	Frame ms	Delay ms	Comp. type	MIPS	MOS
WB						
OPUS	6-510	Múltiplos de 2.5	2.5-60 -CELT e 2.5-20 -SILK	PCM	0 a 10	-
G722	48, 56, 64	0.0625	1.5625	SB - ADPCM	5	4.1
iSAC	10 a 32	30 a 60	63 a 123	-	.6 a 10	-
NB						
G711 PCMA	64	0.125	0.25	PCMA	1	4.1
G711 PCMU	64	0.125	0.25	PCMU	1	4.1
iLBC	13.33	30	40	LPC	18	3.8
iLBC	15.2	20	50	CELP	15	3.9
Video						
Codec	Bitrate kbs	Frame FPS	Pixel ratio	Res	Comp.	SSIM (1000 kbs)
VP8	Arbitrary	Arbitrary	1:1	16384 x 16384	lossy	19.1
VP9	Arbitrary (VP8/2)	Arbitrary	1:1	65536 x 65536	lossy	21.5
H264 /AVC	Variable	>300	-	8192 x 4320	lossy	21

Tabela 1 - Comparação dos codecs usados

VII. GLOSSÁRIO

1. Application Programming Interface é um conjunto de funções ou procedimentos que facilitam o desenvolvimento de aplicações.
2. Método de implementação de uma rede de telecomunicações através de um canal dedicado de comunicação entre dois nós da rede. Ex.: rede de telefonia tradicional. Quando é efetuada uma chamada é criado um canal designado entre os nós intervenientes.
3. *Framework* para o tratamento de audio.
4. *Framework* para o tratamento de video.
5. Internet Engineering Task Force é uma organização voluntária que desenvolve e promove standards da internet.
6. World Wide Web Consortium é a organização internacional que mantém os *standards* da World Wide Web.
7. Web Hypertext Application Technology Working Group é uma comunidade de interessados na evolução do HTML. Membros centrais são

- empresas responsáveis pelos principais browsers no mercado.
8. Object-Real Time Communications é mais uma API que permite comunicação entre clientes móveis e servidores ou browsers em tempo real. Parte do grupo W3C ORTC, fundado pela Hookflash, inc. (companhia de software de RT IPVoice).
 9. Protocolo de criação de canais de comunicação *full-duplex*.
 10. É equiparável a uma Máquina Virtual. No sentido em que permite correr/testar programas que não se queiram correr no sistema. Principal diferença será que uma *sandbox* é um programa que permite correr outros programas, enquanto que uma VM recria/virtualiza um OS onde se correm os programas.
 11. Licença de código aberto com poucas restrições. A primeira utilização foi relativa ao Berkley Software Distribution (BSD), um OS baseado em Unix.
 12. Platform as a Service é o termo usa para o que se pode chamar de *buiding-block* no desenvolvimento de uma aplicação.
 13. JavaScript *runtime environment*.
 14. Executável para a linha de comandos que permite criação de tuneis e links virtuais.
 15. *Framework* em JavaScript para Node.js para o desenvolvimento de aplicações web.
 16. Uma biblioteca JavaScript, que permite comunicação em tempo real e comunicação *full-duplex*. Utiliza o protocolo WebSocket.
 17. Gestor de *packages* de JavaScript
 18. Comunicação de múltiplos dispositivos de múltiplas *medias* interrelacionados através da internet.
 19. Revisão do HTTP, derivado do protocolo SPDY.
 20. Protocolo de rede desenvolvido pela Google para transportar dados pela internet. A sua redução de latência é através da compressão, multiplexação e priorização.
 21. Real Time Messaging Protocol é um protocolo de comunicação por texto em tempo real utilizado pelo Flash.
 22. Real Time Media Flow Protocol é um protocolo de transmissão de dados em tempo real utilizado pelo Flash.
 23. Upload dos códigos para um banco de dados (ex.: repositório de dados da GitHub).

VIII. BIBLIOGRAFIA

- webrtc.org (appr.tc – app de teste)
- io13webrtc.appspot.com - Google Developers WebRTC presentation
- bloggeek.me - funcionamento
- youtube.com
 - youtube.com /channel/UCjFO5t0MLyQaidKGpGoRewg – FullStack
- webrtcglossary.com - definições
- onsip.com – definições e funcionamento
- searchcloudcomputing.techtarget.com - definições
- stackoverflow.com – funcionamento
- searchcloudcomputing.techtarget.com - definições
- en.wikipedia.org - definições e história
- quora.com – empresas, casos de uso
- developer.mozilla.org - funcionamento
- onsip.com - funcionamento
- codelabs.developers.google.com - tutorial
- github.com
 - github.com/agilityfeat/webrtc-video-conference-tutorial/pull/2/files?file-filters%5B%5D=dotfile - tutorial
- news.google.com – previsões de mercado