Iago Soares dos Santos Faria

Desenvolvimento de uma API para envio de áudio, vídeo e texto no desenvolvimento de jogos

São José - SC

Julho/2018

Iago Soares dos Santos Faria

Desenvolvimento de uma API para envio de áudio, vídeo e texto no desenvolvimento de jogos

Monografia submetida à Coordenação de Engenharia de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma Bacharel em Engenharia de Telecomunicações.

Instituto Federal de Santa Catarina – IFSC

Campus São José

Engenharia de Telecomunicações

Orientador: Ederson Torresini

São José - SC

Julho/2018

Iago Soares dos Santos Faria

Desenvolvimento de uma API para envio de áudio, vídeo e texto no desenvolvimento de jogos

Monografia submetida à Coordenação de Engenharia de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma Bacharel em Engenharia de Telecomunicações.

Trabalho aprovado. São José - SC, 4 de junho de 2017:

Prof. Ederson Torresini, Me.
Orientador

Prof. Eraldo Silveira e Silva, Dr. Eng.
Coorientador

Prof. Tiago Semprebom, Dr. Eng.
Convidado

São José - SC
Julho/2018

Lista de abreviaturas e siglas

Sumário

1 Introdução

Desde o princípio o ser humano sempre se entreteve através de jogos, e os jogos interativos, em especial, necessitam de um fator fundamental: a comunicação. Com a evolução dos jogos e da comunicação, os jogos interativos mudaram muito ao longo do tempo, como veremos adiante.

Nos primórdios da humanidade, os seres humanos utilizavam unicamente a nossa própria fala para nos comunicar, impossibilitando a comunicação a distâncias longas e assim nossa comunicação síncrona[1] pouco evoluiu até no início da segunda metade do século XIX, quando foi descoberto a capacidade de converter sinais sonoros em energia elétrica e transmiti-los em um fio, criando a telegrafia e a telefonia analógica. A título de exemplo, em 1964, a empresa estadunidense AT&T realizou a primeira videoconferência utilizando os recursos da telefonia analógica, transmitindo 30 fotos em preto e branco a cada segundo - seu maior ponto negativo era o custo proibitivo (MONTEIRO, 2002). Enquanto isso os jogos interativos permaneceram iguais, se resumindo ao contato pessoal dos jogadores, impedindo que houvesse jogo dependendo da distância dos mesmos, já que o limite era até onde a voz humana e os gestos alcançassem, a criação da telefonia analógica e da possibilidade de videoconferência pouco impactou no modo de se comunicar nos jogos.

Na década de 1960, com o surgimento dos transistores e da eletrônica digital, veio a telefonia digital, melhorando a comunicação comparada a telefonia analógica, devido a menor interferência ao ruído, maior poder de processamento das centrais telefônicas, entre outros fatores. Com a criação de dispositivos eletrônicos mais avançados como fliperama e telejogo, foram criados os jogos eletrônicos, que permitiam o jogador ter o dispositivo como mais um jogador, ou seja, um jogo não precisaria ter necessariamente dois humanos, podendo ter como oponente ou integrante de sua equipe a própria máquina, criando um entretenimento eletrônico ou colocando mais um controle no dispositivo permitindo que outra pessoa jogasse como parceiro ou adversário, mas a comunicação ainda era a voz já que os jogadores ficavam próximos, pois nos dispositivos não havia a possibilidade de conversar com o outro jogador através do próprio jogo. Os avanço da telefonia digital, assim como os avanços da telefonia analógica, mais uma vez trouxeram pouco impacto real no ramo dos jogos interativos, seja pela complexidade na implantação e manutenção do sistema, seja o custo elevado.

Desde a criação da Internet, até a sua popularização da década de 1990, é possível perceber o avanço na comunicação: passamos a poder nos conectar com qualquer pessoa do mundo através de chats, e-mails e outras ferramentas em tempo real. Com o surgimento dela é que de fato possibilitou mudança na estrutura dos jogos interativos, como a Internet

[1] Os jogos por correspondência são conhecidos há séculos, por várias culturas e povos.

possibilita a conexão de usuários do mundo todo desde que eles tenham acesso a ela, criou- se a possibilidade dessas pessoas jogarem entre si simultaneamente, criando os jogos *online*. E no esteio das novas tecnologias e possibilidades as empresas de jogos criaram *chats* nos próprio jogos, onde a desvantagem era que para teclar no *chat* o jogador deveria parar de jogar, pois muitos botões faziam parte da jogabilidade. Ou seja, apesar dos avanços ainda havia restrições tecnológicas e de recursos computacionais.

Este foi o primeiro grande avanço, pois até o presente momento nunca na história da humanidade pessoas poderiam jogar e se comunicar a milhares de quilômetros.

A comunicação entre 2 dispositivos com a troca de áudio e vídeo demorou a evoluir no início da Internet porque as taxas eram muito baixas e não haveria videoconferências na Internet até que as conexões fossem capazes de suportar a quantidade de dados gerados numa captura de imagens a diversos quadros por segundo, pois o enlace nos anos 1990 era praticamente textitdial-up, com taxas de 14,4 a 56 kbps. Com o surgimento da banda larga os enlaces passaram a suportar taxas bem mais altas e foi possível a troca de áudio e vídeo entre dois dispositivos.

As primeiras experimentações de videoconferência foram no início da década de 1990 com aplicações proprietárias como CU-SeeMe. Depois, vieram os protocolos abertos: em 1996 o H.323; SIP, em 1999(PEREZ, 2015).

A Internet mudou radicalmente a comunicação humana, e prova disso é a telefonia *Voice over IP* (VoIP), que tem cada vez mais usuários e num futuro próximo deve ultrapassar em número de usuários a telefonia digital que ainda tem o maior número de usuários atualmente, como mostrado na Figura 1. Essa nova tecnologia foi muito aproveitada pelos jogos interativos *online*, que em vez de o jogador parar de jogar para digitar no *chat*, poderia apenas usar simultaneamente um microfone para conversar com outros jogadores, dessa vez de uma forma mais natural. Esse avanço criou diversas ferramentas no mundo dos jogos, como pode-se perceber nas transmissões ao vivo de jogos multiusuário, ou seja, um jogador que transmite seu jogo para que outras pessoas o vejam jogando, conceito que movimenta diversos usuários em serviços de transmissão em tempo real hoje em dia.

Atualmente, com a Internet sendo acessível para mais da metade da população, nosso acesso com ela se dá basicamente pelos navegadores, com o passar do tempo foram ganhando muita importância, hoje são capazes de avisar notificações do Facebook, vídeos novos de canais inscritos no Youtube, *e-mails* do Outlook, entre outras funções. A função do navegador foi mudada a ponto dele atualmente ter acesso direto aos dados que são recebidos na câmera e no microfone de seu dispositivo, com esse avanço foi criado o WebRTC, uma aplicação que permite uma conexão onde trafegam áudio, vídeo e outros tipos de dados sem a necessidade de instalação de *plugins*, como era feito há anos atrás.

Com a adoção de linguagens *Web* como HTML5, começaram a ser criados jogos

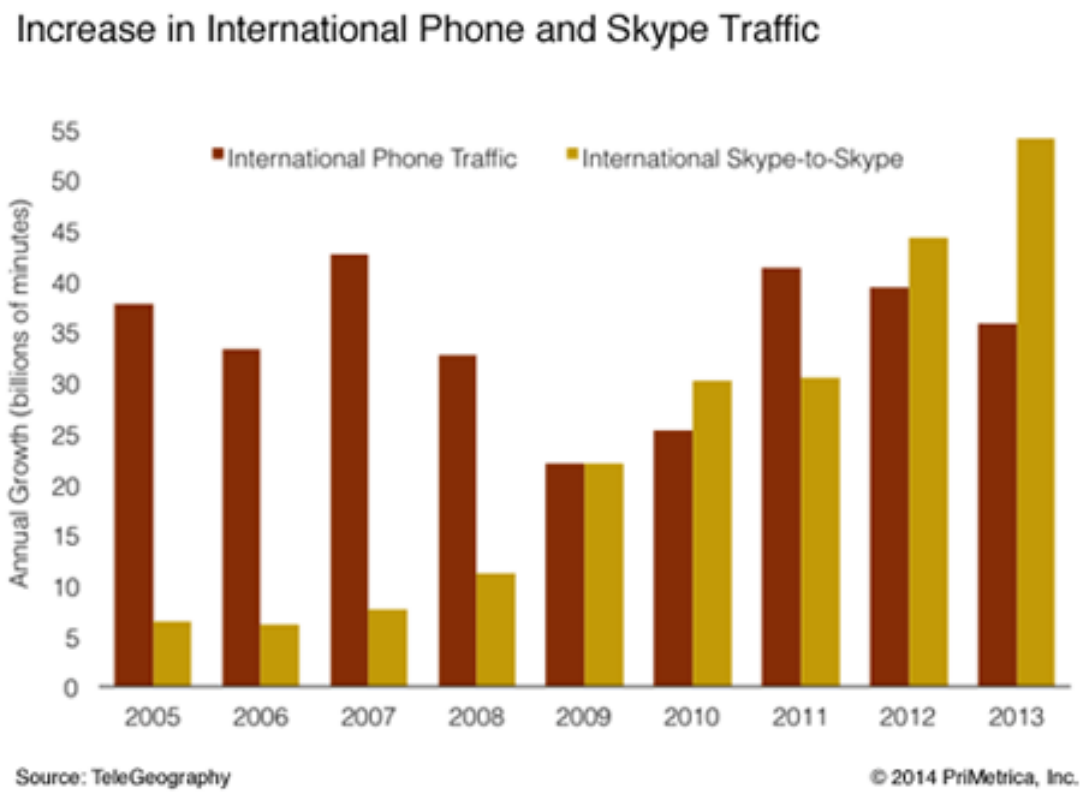

Figura 1 – Crescimento do Skype em comparação a telefonia digital. Fonte: Telegeography (2014).

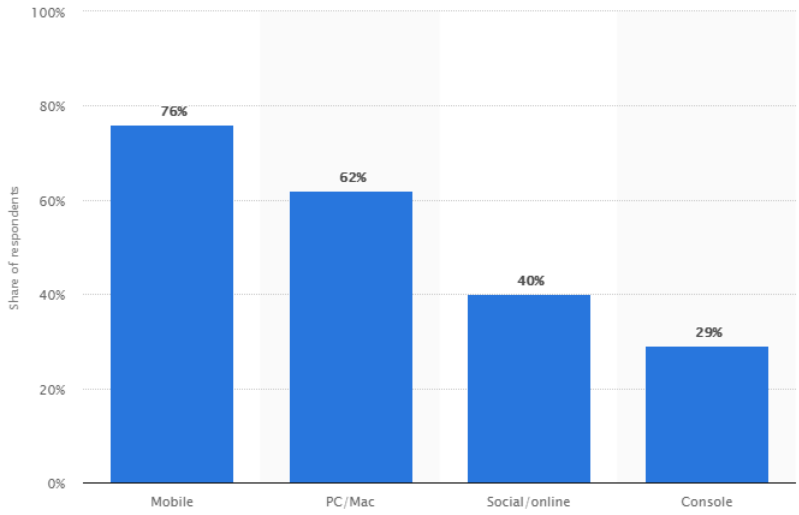

Figura 2 – Preferência dos consumidores aos tipos de jogos segundo as empresas. Fonte: Statista (2018).

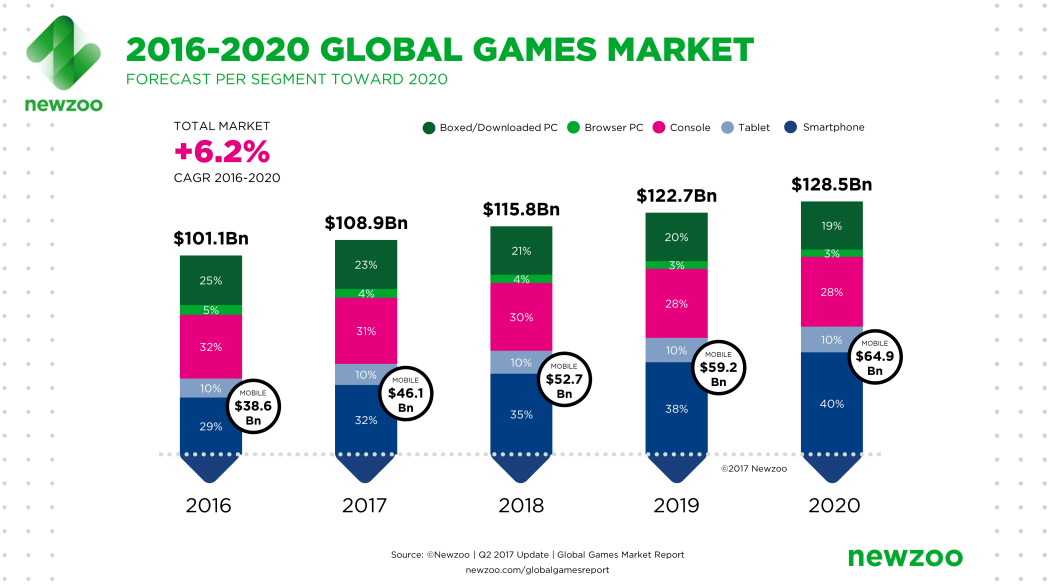

Figura 3 – Crescimento do mercado de jogos. Fonte: Newzoo (2018a).

Web mais complexos, como mostrado na Figura 2, há uma grande preferência dos usuários por jogos online. Utilizando o WebGL para a renderização de gráficos, tudo isso de forma nativa, sem adição de programas ou *plug-ins* adicionais. Seguindo esta tendência é criada uma possibilidade de transmitir áudio e vídeo de participantes de jogos sem a necessidade de *plug-ins* ou aplicativos externos através de uma ferramenta nativa dos navegadores, o WebRTC. Na Figura 3 e Figura 4 é visto como o mercado de jogos está aumentando e junto com ele os jogos de navegadores, portanto o WebRTC pode ser fundamental nesse crescimento aumentado a eficiência na comunicação dos jogos.

1.1 Motivação

O WebRTC tem como função o envio e recepção de voz, imagem e outros dados binários de navegador para navegador sem a necessidade de *plug-in* ou descarregar um aplicativo específico, ocasionando menos complexidade ao cliente final. Atualmente os navegadores mais utilizados já têm o suporte a WebRTC[2], mostrando uma possibilidade viável para o desenvolvimento de jogos *online*. A oferta de ambientes de desenvolvimento, *frameworks* e *Application Programming Interface* (API)s para auxiliar e mesmo agilizar a produção de novos jogos se apresenta bastante atrativa, e a prova disso são as várias iniciativas do mercado de jogos digitais *online*[3].

[2] https://caniuse.com/#search=webrtc
[3] <https://steamcommunity.com/updates/chatupdate>, acessado em 20/06/2018.

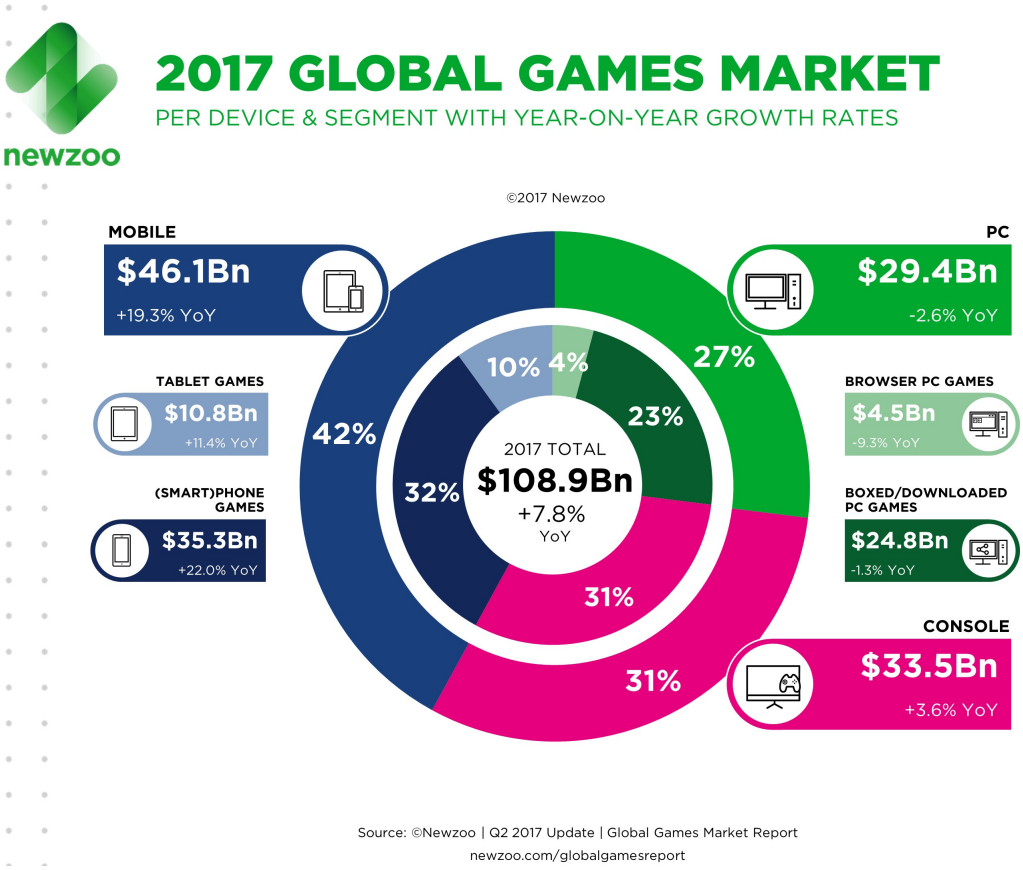

Figura 4 – Proporção do rendimento dos jogos de navegadores comparado a outras plataformas em 2017. Fonte: Newzoo (2018b).

1.2 Objetivos

1.2.1 Objetivo Principal

O objetivo principal deste trabalho é desenvolver uma API de um *framework* de suporte para o desenvolvimento de jogos digitais online.

1.2.2 Objetivos Específicos

- Levantamento de requisitos de jogos digitais de tabuleiro 2D.
- Definição de uma API, descrevendo as funções que o desenvolvedor poderá usar da biblioteca.
- Criação de uma biblioteca que possibilita a transmissão de informações do jogo, como também a comunicação dos jogadores.
- Implementar um jogo simples de tabuleiro para documentação e exemplo.

1.3 Organização do texto

O texto se encontra dividido em 3 capítulos, o Capítulo 2 é uma revisão sobre os conceitos básicos necessários a implementação dos objetivos, como a tecnologia WebRTC, comunicação web em tempo real, codecs, sinalização e protocolos. No Capítulo 3 é definido o sistema proposto baseado nos conceitos abordados na fundamentação teórica e uma visão geral do sistema.

2 Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica do trabalho, onde se encontra o estudo das tecnologias que serão utilizadas na implementação do projeto.

2.1 WebRTC

O WebRTC é uma solução de código aberto definido na RFC 7478 Holmberg, Hakansson e Eriksson (2015), capaz de fazer a comunicação de qualquer tipo de dado, como áudio, vídeo, arquivos, até sessão de jogos online com comunicação por voz, entre outros, tudo isso utilizando WebSockets. Com o WebRTC os desenvolvedores podem fazer aplicações que envolvam comunicação onde há troca de áudio e vídeo entre os participantes de forma mais fácil, abstraindo o conhecimento aprofundado que o mesmo deveria ter sobre codecs e suas funcionalidades, tais como redução de ruído, cancelamento de eco, detecção de voz ativa entre outros. Tais responsabilidades ficam atribuídas ao navegador permitindo que os desenvolvedores foquem apenas na aplicação podendo até adicionar funcionalidades caso o WebRTC não tenha para uma determinada aplicação. O WebRTC é uma solução flexível ao desenvolvedor pois não especifica um protocolo de sinalização padrão e também não especificando quais codecs devem ser usados.

O WebRTC tem preferência no uso de ferramentas de cógigo aberto, seu uso se dá através da linguagem JavaScript através de APIs, que dão ao desenvolvedor o acesso a recursos como câmeras e microfones do dispositivo, que foram definidas pela *World Wide Web Consortium* (W3C). Sua arquitetura é compostas por 2 partes: a primeira é a WebAPI onde estão as funções utilizadas pelos desenvolvedores para implementar suas aplicações. A segunda é a WebRTC C++ API, utilizada no desenvolvimento de navegadores (TOGO, 2015). A Figura 5 mostra a arquitetura do WebRTC.

2.1.1 Web API

A *Web* API, utilizada no desenvolvimento de aplicações de comunicação com áudio e vídeo em navegadores, é dividida em três partes: `MediaStream`, `PeerConnection` e `DataChannels`. Essas partes usadas de modos diferentes podem dar diversas funcionalidades as aplicações que serão desenvolvidas com o WebRTC.

MediaStream

O *MediaStream* é função do WebRTC que fornece uma fonte de dados para o WebRTC (BORGES, 2013). É responsável por requerir os dados dados gerados pelo microfone e/ou câmera do dispositivo, que serão enviados através dele utilizando o protocolo de transporte

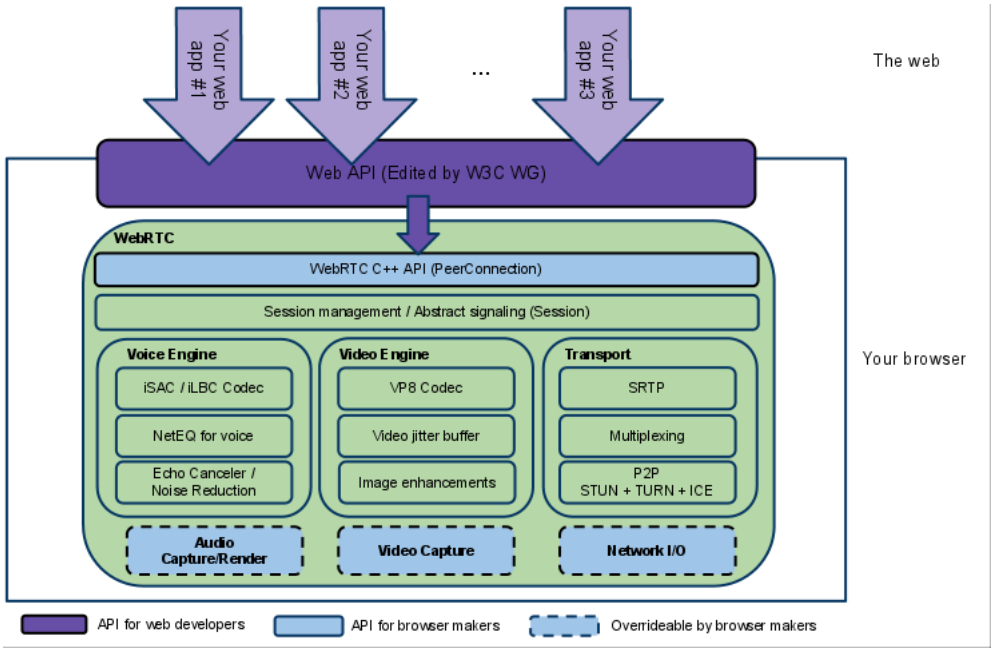

Figura 5 – Arquitetura WebRTC. Fonte: WebRTC (2018).

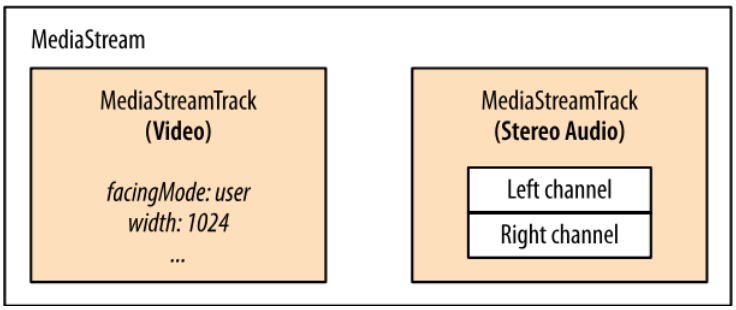

Figura 6 – Blocos do MediaStream. Fonte: Grigorik (2013).

Secure Real-Time Protocol (SRTP), que é uma exigência do WebRTC. Como já dito o WebRTC usa o JavaScript para gerar aplicações. Através da função *getUserMedia()* é feita a solicitação dos dados tanto no microfone quanto na câmera e a função *gotStream()* gera o fluxo de dados recebidos da função mencionada anteriormente. Na Figura 6 é mostrados os blocos que formam o MediaStream.

O *MediaStream* permite que o usuário escolha a qualidade da mídia a ser transmitida (taxa de bits, resolução de vídeo, etc.). É formado por um ou mais *MediaStreamTrack* como mostrado na figura ou seja, numa transmissão de vídeo com som, há no mínimo 2 *MediaStreamTrack*, onde são o áudio é um *MediaStreamTrack* e o vídeo outro, caso haja apenas uma fonte de áudio e vídeo, permitindo o uso de mais de uma câmera ou diversas fontes de áudio, possibilitando a criação de várias aplicações, pois não há na teoria um limite para a quantidade de *MediaStreamTrack*. Na Figura 7 é visto um exemplo de código em JavaScript do MediaStream.

PeerConnection

```
navigator.getUserMedia(constraints, gotStream, logError);

function gotStream(stream) { 7
  var video = document.querySelector('video');
  video.src = window.URL.createObjectURL(stream);
}
```

Figura 7 – Exemplo de código do MediaStream. Fonte: Grigorik (2013).

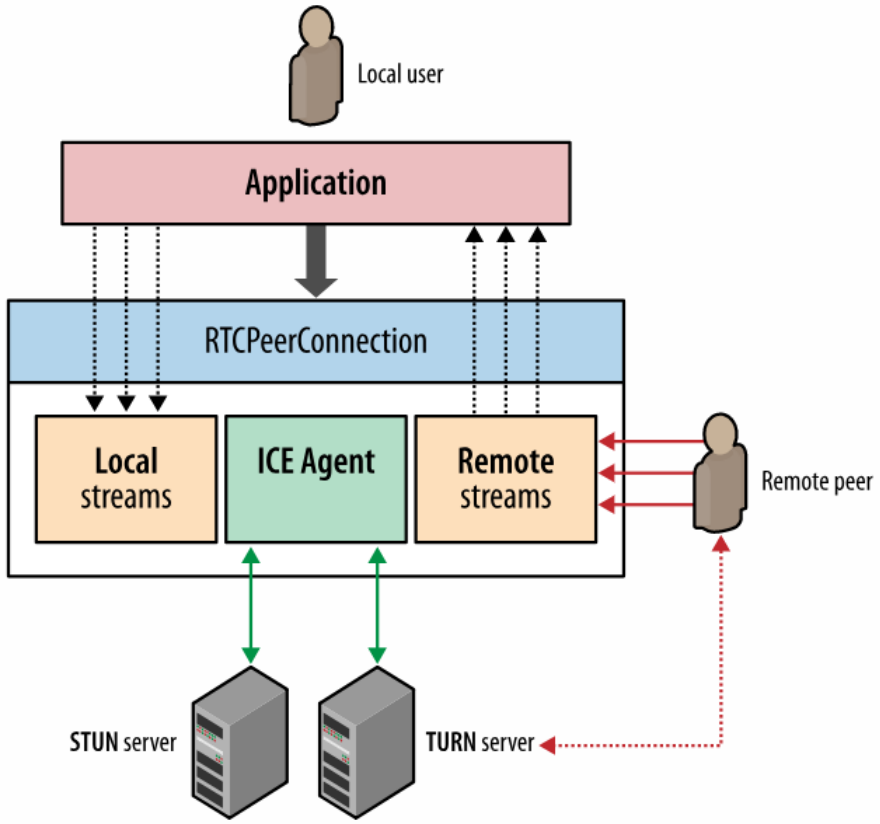

Figura 8 – Blocos do PeerConnection. Fonte: Grigorik (2013).

O *PeerConnection* é a parte da *WebAPI* que estabelece uma conexão estável e eficiente entre dois navegadores. Ele faz a negociação dos codecs que serão utilizados, criptografia e gerenciamento de banda, protegendo os desenvolvedores *Web* de diversas complexidades que existem ao se fazer uma conexão fim-a-fim (TOGO, 2015), também é feito em JavaScript de acordo com Figura 9. O *PeerConnection* faz tudo isso através de um canal de sinalização, implementado num servidor, utilizando geralmente WebSockets (FETTE; MELNIKOV, 2011) por ser um mecanismo bidirecional para troca de dados sobre *HyperText Transfer Protocol* (HTTP) como mostra Figura 8.

DataChannel

Já que o *MediaStream* e o *PeerConnection* fazem a captura e transmissão de áudio e vídeo, e como no WebRTC também há envio de outros dados que não sejam apenas esses, o *DataChannel* se responsabiliza pela transferência desses dados, usando também o

```
pc.createOffer(function(offer) { 5
  pc.setLocalDescription(offer); 6
  signalingChannel.send(offer.sdp); 7
```

Figura 9 – Exemplo de código do PeerConnection. Fonte: Grigorik (2013).

PeerConnection para a transmissão dos dados. Utilizando o `DataChannel` são esperadas altas taxas de transmissão e consequentemente menos latência entre a conexão dos participantes (TOGO, 2015). Os dados transmitidos através do *DataChannel* não precisam ser em tempo real podendo-se utilizar tanto *Transport Control Protocol* (TCP), causando maior latência devido a garantia de entrega, ou *User Datagram Protocol* (UDP), com menor latência mas com possível perda de pacotes. Em Holmberg, Hakansson e Eriksson (2015) há casos de uso de *DataChannel* para diversas aplicações, inclusive jogos.

Dados que podem ser transferidos:

- Arquivos;
- Mensagens instantâneas;
- Compartilhamento de tela;
- Jogos;

Com os itens acima dá pra entender a variedade de aplicações que podem ser concebidas, como chats, jogos online, ensino a distância, área de trabalho remota, entre outros.

2.1.2 *Voice e Video engine*

São mecanismos que formam o WebRTC responsáveis desde a captura do áudio da placa de som e de vídeo da câmera do dispositivo, tratamento, o que inclui redução de ruído entre outras técnicas, e envio para a interface de rede. Em Valin e Bran (2016) e Proust (2016) há considerações sobre codecs de áudio a serem utilizados.

2.1.3 Transporte

Em relação ao transporte da mídia, o WebRTC faz uso do protocolo SRTP, criptografa a mídia. Para clientes que estão por trás de *Network Adress Translation* (NAT), é preciso a utilização dos protocolos *Interactive Connectivity Establishment* (ICE), *Session Traversal Utilities for NAT* (STUN) e/ou *Traversal Using Relays around NAT* (TURN) no envio da mídia (TOGO, 2015).

2.2 Codecs

Codec é a abreviação de codificator/decodificator, a função básica do codificador é amostrar, quantizar e transformar os dados recebidos pelo microfone ou câmera, enquanto o decodificador tem a função básica de receber os bits, quantificá-los, amostrá-los e reproduzir na câmera ou no microfone do dispositivo. Os codecs podem ter funções adicionais como a compressão dos dados, redução de ruídos, detecção de voz ativa, cancelamento de eco, entre outros. A seção tratará de abordar apenas os codecs usados no WebRTC.

2.2.1 Codecs de áudio

Como já dito anteriormente o WebRTC prioriza usar codecs que também sejam código-livre e sem *royalties*, mesmo com essas restrições existem muitos codecs que satisfazem a premissa como Opus, Speex, Vorbis, iLBC, iSAC, G.711, G.722 e G.729. Mas o grande número de possibilidades não é uma solução, pois para que haja comunicação os 2 dispositivos envolvidos nela devem utilizar os mesmos codecs,, a não ser que haja uma terceira parte que sirva de *gateway*, mesmo havendo várias opções a negociação de codecs seria mais longa então há uma pressão para que escolham um codec padrão para possibilitar a interoperabilidade entre todos os clientes WebRTC. Mesmo se houver a escolha de um codec padrão isso não impede que uma aplicação use um codec específico, o que influenciaria apenas os seus usuários, não todos. Atualmente se recomenda uma taxa de amostragem de no mínimo de 8 KHz, que seria a mínima taxa de amostragem existente para codecs, ultimamente os codecs tidos como obrigatórios no WebRTC são[1]:

- Opus, um codec definido pela Valin, Vos e Terriberry (2012) tem características como frequência de amostragem de 8KHz a 48 KHz, 8 bits de quantização, taxa de 6 kbps a 510 kbps, tempo de pacote variando de 2,5 até 120 ms e latência padrão de 26,5 ms, mas que pode ser reduzido para 5 ms. Como visto ele é um codec bem flexível, tendo taxa de bits e frequência de amostragem variável, podendo de acordo com a situação da conexão diminuir a taxa ou até mesmo melhorar a qualidade do áudio, mas sua desvantagem é o custo de processamento para o dispositivo.

- G.711, também chamado de *Pulse Code Modulation* (PCM), é um padrão da *International Telecommunication Union* (ITU) muito usado na telefonia digital, lançado em 1972 que não realiza compressão da voz, apenas amostra e faz a quantização dos dados. Possui características como frequência de amostragem de 8KHz, 8 bits de quantização, taxa 64 kbps, tempo de pacote de 20ms, latência máxima de 125 μs e requer baixo processamento do dispositivo, já que não realiza a compressão e nenhum tipo de tratamento do áudio,

[1] <https://www.heise.de/newsticker/meldung/Zwei-Audio-Codecs-fuer-Echtzeit-Kommunikation-im-Browser-165661
html>

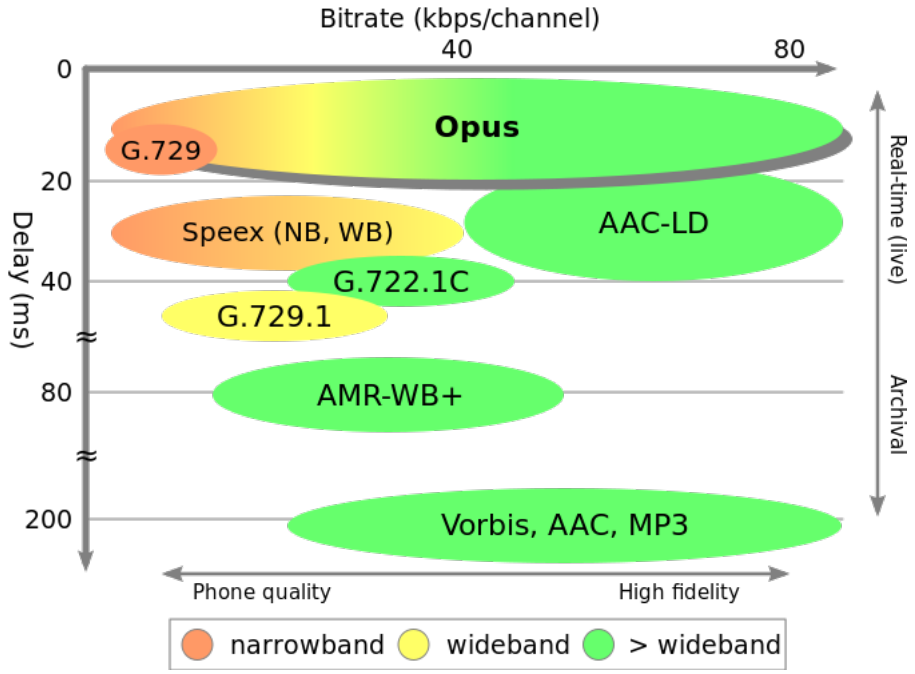

Figura 10 – Comparação do Opus em relação a outros codecs em relação a quantidade usada de banda e delay gerado para determinada qualidade áudio. Fonte: Opus (2012).

além de ser utilizado nos telefones digitais que não tinham uma capacidade alta de processamento.

O Opus tem uma excelente qualidade em praticamente todas as faixas de banda analisadas como visto na Figura 11, isso faz com que ele seja em muitos casos a melhor opção quando se busca qualidade de áudio e baixo delay, como mostraFigura 10. O G.711 é um codec muito utilizado em diversas tecnologias, como na telefonia digital, a adoção dele no projeto WebRTC facilita em muito a interoperabilidade com outros tipos de cenários já que desta forma torna-se muitas vezes dispensável a utilização de um *gateway* para fazer a transcodificação do áudio, ou seja, convertê-lo de Opus para G.711 (BORGES, 2013).

2.2.2 Codecs de vídeo

Como os codecs de áudio, ainda não há definição de um codec de vídeo padrão do WebRTC, aqui há bem menos opções devido ao fato de haver 2 codecs de vídeo que são muito mais populares (escolhidos pelos desenvolvedores de navegadores) que outros codecs: H.265[2] e o VP9[3].

- VP9, é a evolução do codec VP8, foi criado pela *On2 Technologies*, empresa que em 2010 foi comprada pela Google, o VP9 foi o primeiro codec a ser desenvolvido pela

[2] Novos equipamentos e aplicativos usam H.265, compatível com o legado que usa H.264.
[3] Substitui recentemente o codec VP8.

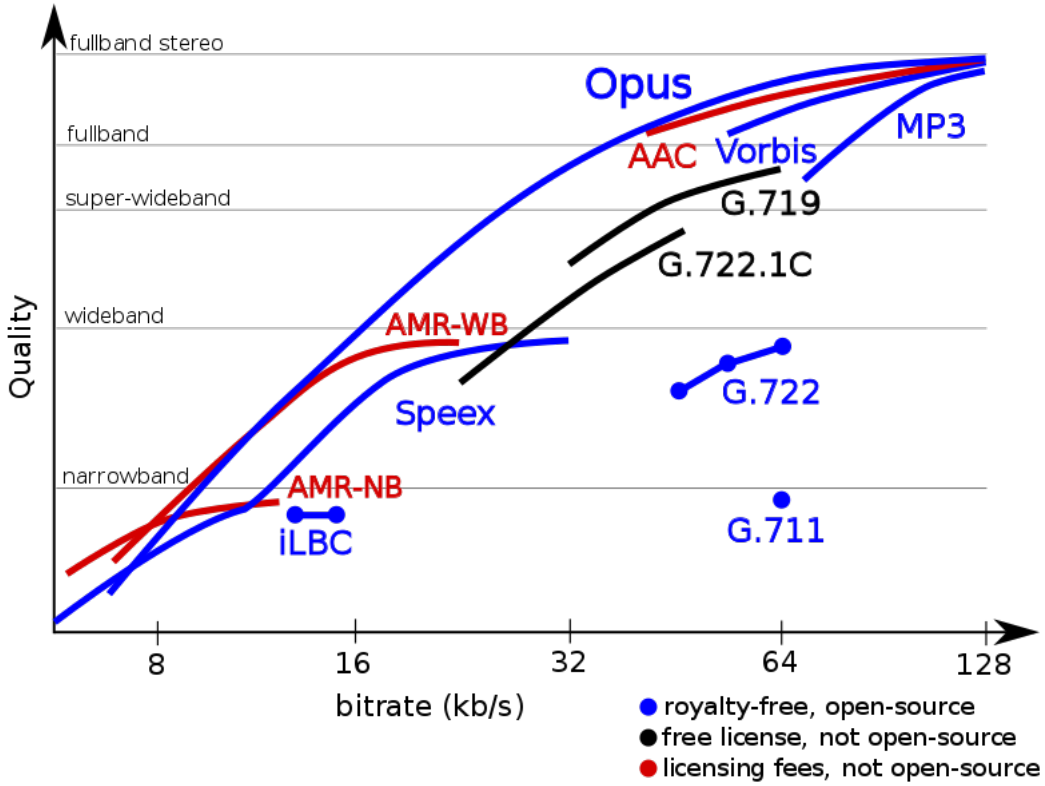

Figura 11 – Comparação do Opus em relação a outros codecs em relação a quantidade usada de banda para determinada qualidade áudio. Fonte: Opus (2012).

empresa após a compra, por curiosidade os codecs fabricados pela *On2 Technologies* tinham licensa proprietária antes da compra, sendo todos eles liberados após o fato. Sites como Youtube e dispositivos com sistema operacional Android o utilizam.

- H.265 (*High Efficiency Video Coding*), definido pelo ITU-T é, em conjunto com H.264, a família de codecs mais utilizada atualmente na distribuição de vídeos em alta definição. Utilizado em dispositivos com sistema IOS.

O H.265 é melhor que o VP9 quando se refere a qualidade, outro ponto a favor do H.265 é que diversos dispositivos o implementam em *hardware* fazendo que o seu consumo de energia seja significantemente menor, enquanto o VP9 geralmente é implementado em *software*, que consome mais energia. As grandes vantagens do VP9 são o fato de ser livre de *royalties*, sendo que essa é uma premissa do WebRTC desde o seu princípio, já para o uso do H.265 é necessário licença. O VP9 e o H.265 foram desenvolvidos para reduzir em 50% a taxa de bits utilizada pelos seus antecessores dada certa resolução.

Quando o projeto do WebRTC foi concebido os codecs mais populares da época eram os antecessores dos codecs citados acima, no caso H.264 e o VP8. Como o VP8, assim como o VP9, é implementado em software, portanto numa atualização de software automaticamente o dispositivo passa a usar o VP9. No caso do H.264, assim como o H.265, é implementado em hardware, ainda há suporte a ele pois ainda é presente em dispostivos IOS mais antigos.

2.3 Protocolos de sinalização da mídia no WebRTC

O WebRTC não define uma sinalização específica, permitindo inclusive que o desenvolvedor faça uma sinalização própria para o gerenciamento da mídia, podendo até ser baseada em sinalizações já conhecidas. Os protocolos que serão abordados abaixo serão o SIP e o JSEP, além da possibilidade de haver uma sinalização híbrida, apesar disso tem outros protocolos que seriam úteis como o Jingle, XMPP, mas não serão estudados pelo fato do SIP ser um protocolo já muito usado no mundo VoIP e o fato do JSEP ser um protocolo novo e promissor.

2.3.1 SIP

O *Session Initiation Protocol* (SIP) é o protocolo texto de sinalização VoIP mais utilizado atualmente, é um protocolo da camada de aplicação, é definido por Rosenberg et al. (2002), sua função é estabelecer, manter e finalizar sessões multimídia, implementando uma comunicação baseada em requisições e respostas, utilizando os protocolos da Figura 13. Por ser um protocolo texto é baseado nos protocolos HTTP e *Simple Message Transfer Protocol* (SMTP), que também são protocolos texto da camada de aplicação. Suas características são:

- Localização do usuário – Determina o dispositivo onde o usuário está no momento.
- Disponibilidade do usuário – É capaz de saber se um usuário está online ou não, indicando se o usuário está disponível para uma comunicação.
- Recursos do usuário – Informa quais codecs o usuário possui para que seja iniciada uma chamada futuramente.
- Estabelecimento de sessão – Determina os parâmetros da chamada como as portas de sinalização que serão usadas.
- Gerenciamento de sessão – Transferência de sessão, adição de participantes, chamada em espera entre outras funcionalidades, além de modificações de parâmetros como a mudança de codecs durante a chamada.

Uma rede SIP é composta por entidades SIP lógicas que são diferente de entidades físicas, ou seja, apesar das entidades possuírem funções diferentes podem estar em um mesmo dispositivo. As entidades podem ser clientes enviando requisições e servidores enviando respostas. As entidades lógicas são:

- Agente Usuário (*User Agent* (UA)): é o *endpoint* de uma comunicação SIP (Telefones IP, *softphones*), o dispositivo capaz de gerar chamadas. O UA pode ser *User Agent Client* (UAC), caso envie requisições e receba respostas, ou *User Agent Server* (UAS), caso receba requisições e envie respostas.

- Servidor Proxy (*Proxy Server*): é a entidade que faz o intermédio de sessões SIP, cada domínio SIP (SIP URI) tem seu servidor proxy, ele origina pedidos em nome de outros usuários, reescrevendo o cabeçalho dos pacotes e os encaminhando. Ele apenas lê os campos referentes ao destino do pacote, a descrição (*Session Description Protocol* (SDP)) e outros campos não são relevantes para ele. Em alguns cenários pode haver mais de um servidor proxy.

- Servidor de Redirecionamento (*Redirect Server*): aceita pedidos SIP, faz a verificação de localidade do endereço de destino para que o usuário o contate diretamente. Apenas envia respostas do tipo 3xx, que são respostas de redirecionamento.

- Servidor de Registro (Registrar *Server*): sempre usado conjunto com o Servidor Proxy, pode haver também um Servidor de Redirecionamento, ele apenas recebe requisições `REGISTER`, caso receber outra requisição envia uma mensagem de erro 501 (*Not implemented*). Sua função é registrar os usuários que estão na rede. Muito importante na definição da disponibilidade e localização do usuário.

O SIP implementa requisições, as fundamentais são essas:

- `INVITE`: utilizado para estabelecer uma sessão.
- `ACK`: utilizado para confirmar a chegada de uma resposta referente a uma requisição.
- `PRACK`: utilizada como uma confirmação provisória de chegada de uma requisição.
- `BYE`: utilizada para finalizar uma chamada.
- `CANCEL`: utilizada para cancelar uma requisição pendente.
- `OPTIONS`: utilizada para requisitar as características de um *endpoint*.
- `REGISTER`: utilizada para registrar um *endpoint*, apenas servidores de registro podem recebê-lo.
- `INFO`: utilizada durante a chamada para transmitir sinalização.

O SIP, assim como o HTTP, implementa respostas formadas por 3 dígitos onde o primeiro dígito identifica o tipo da resposta:

- `1XX`: respostas provisórias que indicam que ao receber tal requisição medidas estão sendo tomadas.
- `2XX`: respostas de sucesso, foi recebido a requisição, a partir do fato medidas foram tomadas e todas elas obtiveram sucesso.

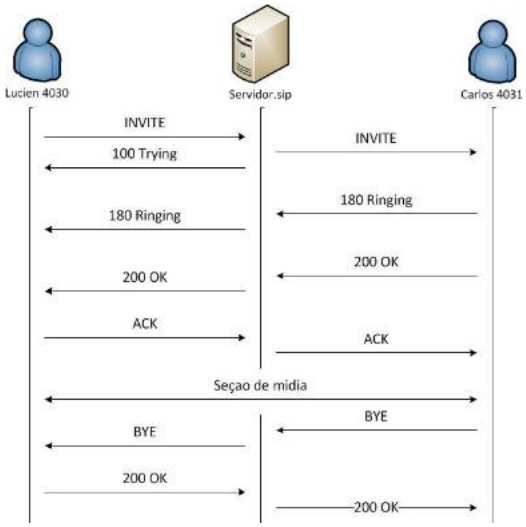

Figura 12 – Diagrama de funcionamento do SIP. Fonte: Delphini ().

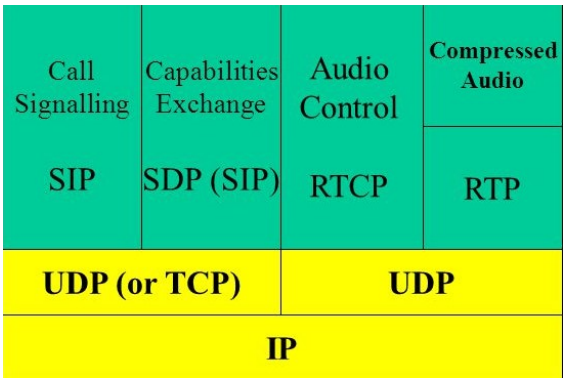

Figura 13 – Pilha de protocolos utilizads numa comunicação com protocolo SIP. Fonte: Delphini ().

- `3XX`: respostas de redirecionamento, enviadas pelo servidor de redirecionamento indicando uma rota melhor a ser usada.
- `4XX`: respostas que indicam erros no cliente(endpoint).
- `5XX`: respostas que indicam erro no servidor.
- `6XX`: respostas que indicam erros globais, que se referem a rede num todo.

A Figura 12 mostra as mensagens de uma comunicação básica usando o protocolo SIP. O SIP, entretanto, não é um modelo cliente-servidor. E como WebRTC utiliza HTTP, esse cliente-servidor, é preciso alguma forma de estabelecer um canal bidirecional para transportar a sinalização entre os agentes (UACs e UASs). Uma possibilidade é o *WebSocket*.

2.3.2 JSEP

O *JavaScript Session Establishment Protocol* (JSEP), como já diz o nome, é um protocolo de estabelecimento de sessão que usa JavaScript, linguagem Web que será abordada

adiante. O JSEP é um protocolo recente que permite total controle da máquina de estados do JavaScript. Atualmente o JSEP ainda não possui nenhuma documentação final oficial - está em fase de rascunho[4]. O JSEP possui interoperabilidade com o SIP e outros protocolos de sinalização, ou seja, pode haver comunnicação entre dispositivos onde um utiliza o JSEP e outro usa o SIP.

2.3.3 Sinalização Híbrida

Numa conexão VoIP há também a possibilidade dos dispositivos que a estabeleceram possuírem sinalizações diferentes, um conceito conhecido como interoperabilidade, permitindo que um lado da conexão use a sinalização SIP ou JSEP enquanto o outro lado utiliza uma sinalização própria ou vice-versa. Pode-se, para tal, utilizar um tradutor de sinalização (*signaling gateway*) como previsto em Ong et al. (1999).

2.3.4 SDP

O SDP é um protocolo texto da camada de aplicação, definido em Handley, Jacobson e Perkins (2006). O fato de ser um protocolo baseado em texto, assim como SIP, facilita a análise ao desenvolvedor. Não é um protocolo de sinalização mas está sempre atrelada a um, pois não possui um mecanismo de transporte próprios, portanto o SDP não transporta mídia, sua função é apenas descrição de sessão. Há outros protocolos de descrição de sessão, mas ele é o mais amplamente usado em comunicações VoIP, por isso foi tomado para estudo.

Sua função é configurar, iniciar e manter uma sessão, é geralmente utilizado em conjunto com o SIP e o *Real-Time Protocol* (RTP), onde descreve as características do fluxo que deverá ser estabelecido entre 2 usuários como os codecs que serão usados e frequência de amostragem que irão usar, o tempo da sessão, identificador de sessão, as portas que servirão para a troca de mídia, protocolo de transporte da mídia, tipo de mídia que será trocada e o endereço para onde a mídia será trafegada.

2.4 *WebSockets*

A Internet foi concebida baseada no mecanismo de pedido e resposta de HTTP, ou seja, quando um usuário acessa uma página há uma requisição pelo conteúdo da mesma e a conexão é desfeita, nada irá acontecer até que ele clique em outra página para atualizar as informações desta nova página. Para solucionar isso veio o AJAX e o sistema de keep-alive no HTTP/1.1, com o nome já diz procura manter viva a conexão, porém, toda a comunicação HTTP continuava comandada pelo cliente, que devia interagir ou utilizar temporizador que atualizasse a página(SOUTO, 2013).

[4] <https://github.com/rtcweb-wg/jsep>

Como já dito anteriormente o HTTP usa um modelo cliente-servidor, ou seja, apenas o cliente faz requisições, o servidor não envia dados a um cliente sem que haja uma requisição do mesmo. Isso impossibilita aplicações em tempo real como *websites* de notícias, acompanhamento de lances de jogos esportivos, redes sociais, jogos online de navegador, entre outras aplicações. Diante disso foi usada por muito tempo o método de sondagem longa (*long polling*), que permite o servidor enviar dados atualizados para os clientes. O método de sondagem longa usa uma conexão HTTP persistente, que é uma conexão que tem menos latência que uma conexão não persistente pelo fato de não precisar renegociar a conexão TCP após a primeira requisição já ter sido enviada, já que a conexão pode ser utilizada para outras requisições. O método tem um problema, o *overhead* de HTTP prejudicava aplicações intolerantes a atrasos, com jogos *online*(TOGO, 2015).

Para solucionar esse problema foi criado o *WebSocket*, que é uma tecnologia que permite a comunicação nos dois sentidos de uma conexão TCP, entre cliente e servidor. Ou seja, a qualquer momento tanto cliente tanto servidor podem começar a enviar dados, diferente do método Ajax, onde a conexão fica aberta, e após uma resposta do servidor é fechada e depois aberta novamente, ou o método de sondagem longa que fica enviando requisições, mesmo que o servidor não tenha nenhuma resposta. Com a utilização de WebSocket é possível conceber aplicações que necessitem de baixa latência como jogos online.

2.4.1 SIP sobre *WebSocket*

É definido em Castillo, Villegas e Pascual (2014) e muito utilizado, já que no SIP as requisições e respotas podem vir tanto do cliente quanto do servidor, definindo um sub-protocolo de *WebSocket*, para a troca de mensagens SIP. Assim como o SIP, ela é composta por algumas entidades que são:

- SIP *WebSocket Client*: entidade responsável por abrir ligações de *WebSockets* de saída se comunicando pelo sub-protocol *WebSocket SIP*.

- SIP *WebSocket Server*: entidade responsável por escutar ligações de entrada usando o mesmo sub-protocolo do *WebSocket Client*, *WebSocket SIP*, para se comunicar.

2.5 Linguagens Web

Como o trabalho é o desenvolvimento de um jogo simples e uma aplicação utilizando WebRTC, todas as linguagens utilizadas são linguagens Web.

2.5.1 HTML5

O *HyperText Markup Language* (HTML), é uma linguagem de marcação criada em 1993, que é utilizada na concepção de páginas Web, sendo responsável pelo conteúdo das mesmas.

O *HyperText Markup Language* versão 5 (HTML5) é composto por marcações (*tags*) que servem para indicar a função e a informação sobre cada elemento da página web, podendo ser imagens, *hyperlinks* e outros tipos de mídia. A nova versão do HTML, HTML5, inclui marcações de vídeo e áudio que são essenciais ao WebRTC (BORGES, 2013), conforme o exemplo a seguir:

```
<video width="640" height="480" controls>
    <source src="video.webm" type="video/webm">
    <source src="video.mp4" type="video/mp4">
    Seu navegador não suporta o elemento <code>audio</code>.
</video>

<audio src="audio.ogg" controls autoplay loop>
    Seu navegador não suporta o elemento <code>audio</code>.
</audio>
```

2.5.2 CSS

O *Cascading Style Sheet* (CSS), é uma linguagem criada em 1996 que é usada para definir visualmente o layout gráfico dos documentos HTML, como os tipos das fontes usadas na página, cores, alturas e larguras, posicionamento, efeitos, entre outros elementos de uma página web, criando link pra uma página que contém estilos, deixando as páginas Web mais atrativas aos usuários (TOGO, 2015).

2.5.3 Javascript

É a linguagem Web mais utilizada atualmente e usada na criação de aplicações com WebRTC, foi criada em 1995 e utilizada pela primeira vez no navegadores NetScape 2.0. É uma linguagem orientada a objetos. O JavaScript escreve funções que são embarcadas ou incluídas em páginas HTML, suas aplicações são orientadas a eventos realizando comunicações assíncrona, ou seja, imediatas. O JavaScript é interpretado somente nos navegadores, diferente das demais linguagens Web que são compiladas, e no lado do cliente permitindo applicações *peer-to-peer*. Com o JavaScript é possível criar efeitos nas páginas Web oferecendo mais interatividade e com o usuário (BORGES, 2013).

O JavaScript pode oferecer 2 tipos de serviços, o *frontend* que é visual ao usuário, enquanto o *backend* é o que acontece no sistema a cada ação do usuário, deixando ao desenvolvedor a escolha de onde será o processamento da aplicação, no cliente ou no servidor. Há protocolos de estabelecimento de sessão como o JSEP, que utilizam exclusivamente a linguagem para seu funcionamento. Combinado com o HTML5, o JavaScript permite o desenvolvimento de jogos para navegador.

2.6 Questões a considerar no transporte de dados em tempo real

No modelo *Open Systems Interconnection* (OSI), a camada de Aplicação é acessível ao programador.

A camada de transporte é a camada 4 tanto no modelo de camadas TCP/*Internet Protocol* (IP) quanto no modelo OSI, é a responsável pelo transporte dos dados na Internet, realizando a definição das portas por exemplo. Nessa camada temos 2 protocolos bem conhecidos e difundidos, o TCP e o UDP, como outros protocolos menos conhecidos que podem ser usados em transmissão em tempo real como o *Datagram Congestion Control Protocol* (DCCP) e o *Stream Control Transmission Protocol* (SCTP). Como o WebRTC permite o envio de diversos tipos de dados, o envio dos mesmos pode ser feita com diferentes protocolos de transporte.

No caso de uma transmissão de áudio e vídeo em tempo real o protocolo UDP e mais vantajoso que o TCP, pois são mídias que necessitam de baixa latência e toleram perdas. O UDP se encaixa no caso já que é orientado a datagrama, não tem negociações longas no estabelecimento de uma conexão, controle de congestionamento ou retransmissão como o TCP. Num cenário de grande perda de pacotes ou o recebimento de pacotes fora de ordem, o UDP irá reproduzir o fluxo, no áudio por exemplo haveria picotes, no TCP deveria ser implementado um *buffer* para que não ocorra esse problema, já que uma perda de pacotes no TCP implica que ele terá que ser retransmitido após o estouro do *Round-Trip Time* (RTT), enquanto o reenvio do pacote não for recebido a reprodução do fluxo permanecerá parado.

No caso de uma transferência de arquivos ou mensagens, é estritamente necessário a confirmação que esses dados foram de fato entregues e sem erros, a transmissão deste tipo de dados permite também uma maior latência, dando ao TCP uma melhor escolha no cenário.

Protocolos como SCTP e DCCP, que na teoria são melhores que o TCP e UDP, pois têm uma latência alta e emprega controle de congestionamento e confirmação de entrega não são viavéis devido a falta de suporte que os rotedores têm a esses protocolos (DOMINGOS, 2014).

2.7 Questões a considerar na comunicação entre diferentes tipos de rede

Um dos grandes problemas das comunicações na Internet é quando uma das partes se encontra atrás de NAT ou *firewalls*. Uma comunicação com WeRTC pode ser facilmente bloqueada por um *firewall*, já que por ser um mecanismo de segurança da rede sua função é justamente supervisionar qualquer pacote que entra no roteador de borda da rede local, podendo bloquear pacotes se baseando nas portas e endereços de envio e destino. Neste caso o que pode ser feito é a renegociação de portas buscando a conexão nas portas que são abertas pelo *firewall*.

Pode acontecer que o usuário fique em uma rede que use NAT, se a aplicação for usada apenas para o cenário de redes *Internet Protocol* versão 6 (IPv6) não haveria este problema, pois o IPv6 veio para solucionar a falta de endereços do *Internet Protocol* versão 4 (IPv4), fato que deve demorar para acontecer no IPv6 já que os endereços tem 128 bits, apesar do conceito de NAT existir em IPv6. No caso o endereço do usuário da aplicação é o endereço do roteador de borda da sua rede, não seu endereço real, caso um outro usuário vá se conectar a esse e tomar o endereço público como endereço real daquele usuário, a conexão não será com ele. Para solucionar isso foram criados 3 protocolos: ICE, STUN e TURN.

O protocolo ICE serve para resolver esses problemas pode até utilizar de funcionalidades dos protocolos TURN e STUN, sendo baseado neste último e tendo uma topologia conforme a Figura 14. Suas etapas são:

- Identificação dos endereços candidatos: antes de receber ou enviar uma oferta SDP, já são identificados os endereços para cada fluxo de mídia, em cada parte que deseja se comunicar, isso é feito utilizando os protocolos TURN e STUN.

- Avaliação e priorização dos candidatos: após a obtenção dos endereços candidatos eles são classificados de acordo como o caminho ótimo para a conexão, pode fazer isso através do uso de um algoritmo ou mecanismos que configuram o protocolo de acordo com o cenário.

- Envio dos endereços candidatos: os endereços candidatos obtidos na primeira etapa são inclusos nas mensagens de oferta e resposta trocadas entre os agentes.

- Checagem de conectividade: após receber os endereços candidatos cada um dos agentes ICE dos usuários identifica os candidatos locais e remotos, depois começa a fazer testes de conectividade seguindo uma ordem de acordo com a categoria e prioridades da conexão, isso é feito usando o protocolo STUN.

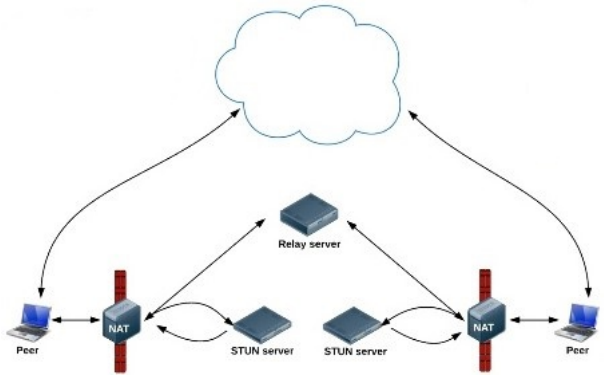

Figura 14 – Arquitetura WebRTC. Fonte: Mészáros (2014).

- Estabelecimento da sessão: após averiguar a conectividade das partes o protocolo ICE envia uma oferta SDP aos candidatos com conectividade com o intuito de informar o melhor caminho a eles.

- Manutenção da conexão: com a conexão já estabelecida, o ICE utilizando alguns recursos do STUN, irá enviar regularmente pacotes *Keep Alive* realizando a manutenção do fluxo, tornando a conexão persistente não necessitando renegociar a conexão TCP após a primeira requisição já ter sido enviada, para fazer a manutenção do fluxo de dados.

O STUN é o outro mecanismo para solucionar o problema de comunicação atrás de NAT, como já dito pode ser usado em conjunto com o ICE, além de servir como base a ele. O STUN foi definido primeiramente Rosenberg et al. (2003) e atualizada em Rosenberg et al. (2008). Ele é capaz de identificar o tipo de NAT que a rede em que atua está utilizando, além de outras características, possibilitando descobrir o IP e porta pública correspondente ao IP e porta interna de um dispositivo, com a Figura 15 pode ficar mais claro.

Apesar de útil o STUN funciona apenas para 3 tipos de NAT, num cenário de NAT simétrico a ação do STUN não surtirá efeito, nos outros tipos de NAT o STUN funciona mas agindo de forma diferente em alguns deles. No caso do NAT *full cone*, o STUN pode descobrir o endereço IP e número de porta pública do dispositivo local sozinho mas nos tipos *restricte cone*, tanto NAT quanto *port*, é preciso que o dispositivo envie um pacote a outra parte da conexão.

O TURN é o último dos mecanismos para solucionar os problemas de comunicações atrás de NAT. Definido em Mahy, Matthews e Rosenberg (2010), atua como um intermediário na comunicação, diferente do STUN: age como um *relay*, que se encontra geralmente na rede pública e tem a função de retransmissão dos pacotes de dados entre os agentes, consumindo muitos recursos do servidor, como processamento e largura de banda, sendo a última opção para estabelecer a comunicação entre dois agentes (VARANDA, 2008), como mostra a Figura 16.

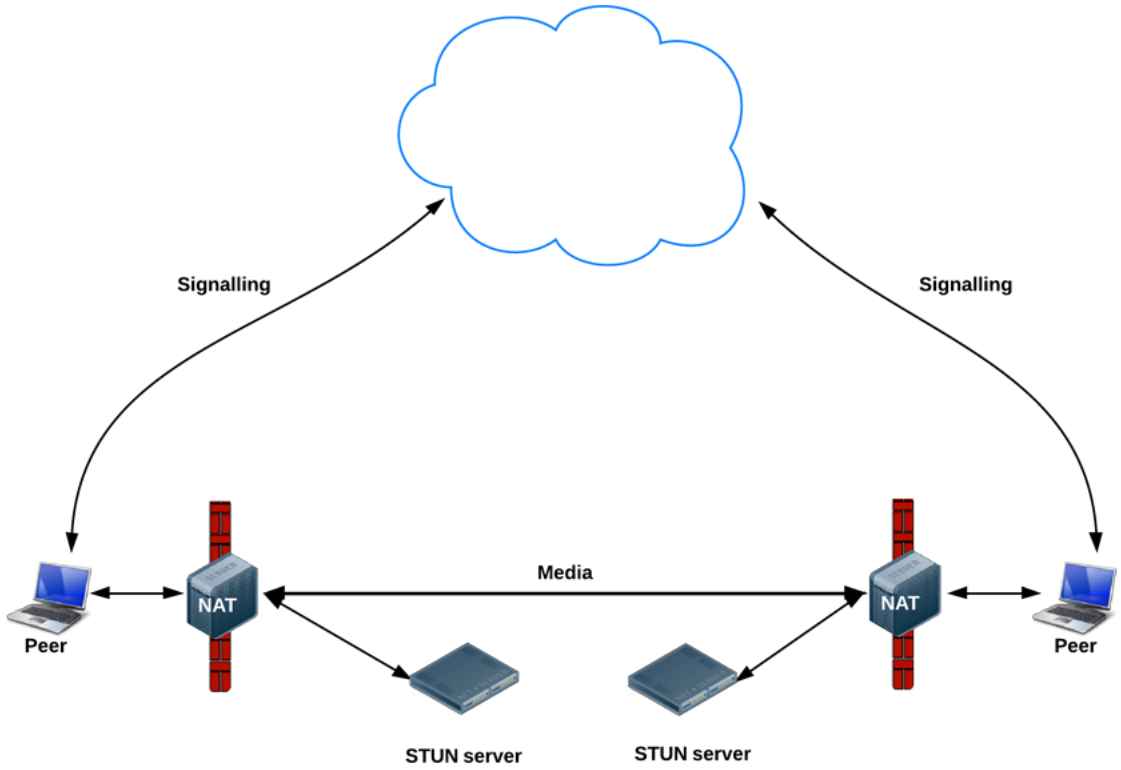

Figura 15 – Arquitetura WebRTC. Fonte: Sam Dutton (HTML5 Rocks) (2016).

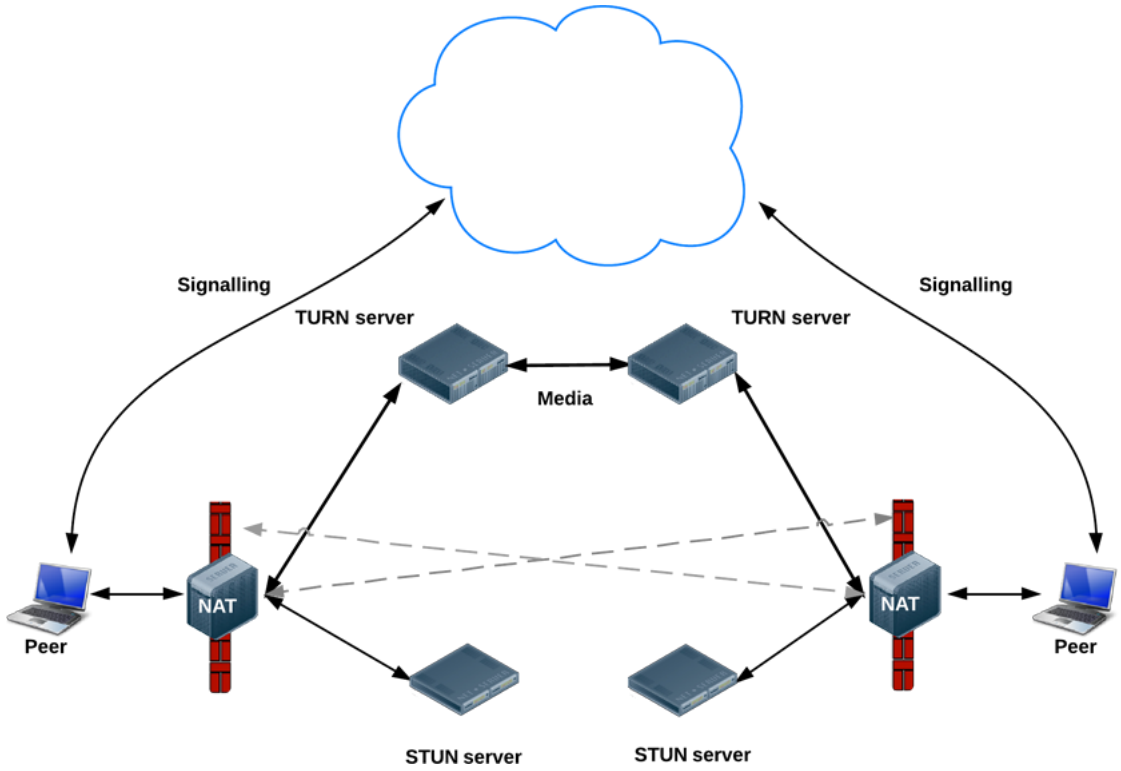

Figura 16 – Arquitetura WebRTC. Fonte: Sam Dutton (HTML5 Rocks) (2016).

3 Proposta

Este capítulo apresenta o que será feito no trabalho, baseado nos conhecimentos estudados e abordados no Capítulo 2.

3.1 Descrição geral do sistema

O sistema tem como objetivo fazer uma API que terá funções de um *framework* de desenvolvimento de jogos. Na API será possível utilizar as funções pré-estabelecidas pela aplicação, que permitirão ao usuário transmitir os dados de qualquer tipo de jogo e estabelecer a comunicação entre os jogadores deste jogo.

Na figura Figura 17 é possível ver como foi pensado o sistema. As mensagens 1 e 2 são referentes ao pedido do usuário para criar uma sala de jogo, recebendo uma confirmação do servidor nas mensagens 3 e 4. Nas mensagens 5 a 12 o jogador com quem irá jogar lista as salas disponíveis e com o retorno dos dados obtidos nas mensagens 7 e 8, loga na qual o usuário A está. A partir deste ponto é dado o início ao jogo por qualquer um dos usuários, na figura é o usuário A, havendo a troca de mídia do jogo e da comunicação entre os usuários. Após o término do jogo é dada a opção de começar um outro jogo, decisão que pode ser tomada por qualquer usuário, na figura o usuário B decide sair.

A figura Figura 17 descreve um cenário bem simples, os usuários que estão numa sessão tem a permissão de convidar outro usuário que esteja disponível, no chat, além da possibilidade

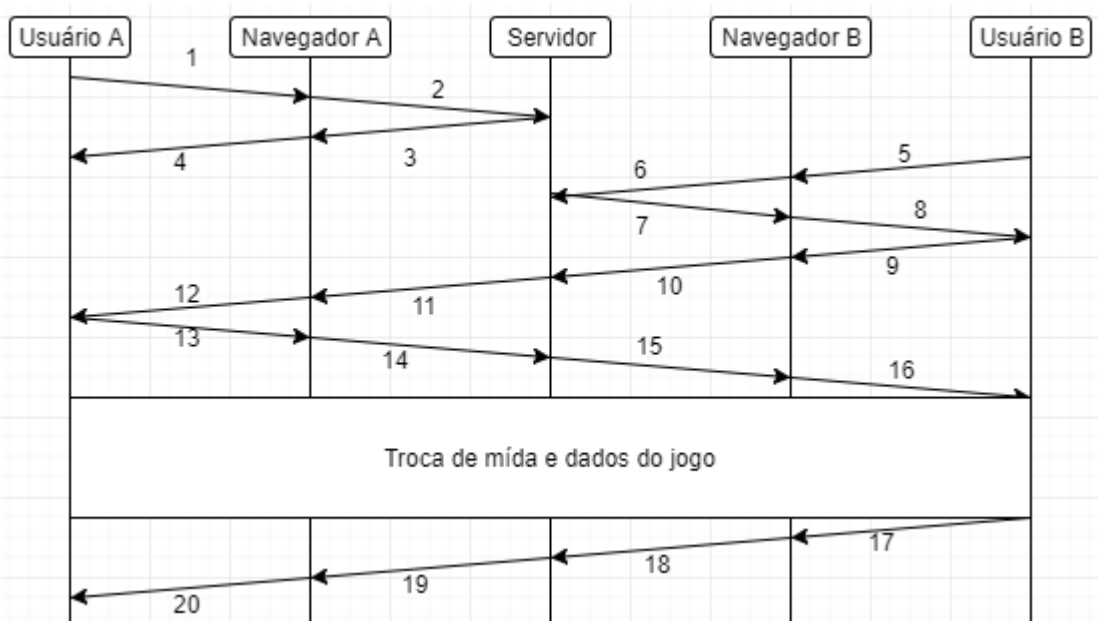

Figura 17 – Exemplo do sistema proposto no trabalho. Fonte:Soares,2018.

de pausar o jogo por determinado período de tempo dado pela API.

As regras da API serão apenas para a comunicação dos jogadores, limitando o desenvolvedor de jogos a funções que criem, listem, saem de salas de jogos, convite de jogadores que estão disponíveis na sua plataforma e o início, pausa e término de uma partida. Sobre os dados do jogo o desenvolvedor terá liberdade de enviar qualquer tipo de dado, já que para isso será usado um `DataChannel` específico, permitindo que qualquer jogo utilize esta API.

3.2 Metodologia

Atualmente, praticamente todos os navegadores mais populares já têm suporte ao WebRTC[1]. Diante disso, o que definiu o navegador para este trabalho foram as funcionalidades oferecidas de desenvolvimento e depuração de código, e o Google Chrome[2] se mostra funcional para tal.

Será dada a preferência ao protocolo SIP, dada a base instalada de aplicações que o usam, com a possibilidade de utilizar o *JavaScript Session Initiation Protocol* (JsSIP), aplicação que utiliza a linguagem Web JavaScript para implementar a sinaização SIP. Além disso, há outros trabalhos do Instituto Federal de Santa Catarina (IFSC) como o de Júnior (2018) que implementa um servidor SIP escalável. O JSEP, na teoria, se apresenta como uma solução bastante atrativa para aplicações Web rodando nativamente em navegadores, mas ainda não tem versão oficial do padrão.

Como o foco deste trabalho é o desenvolvimento do *framework*, será dada atenção secundária ao jogo desenvolvido nessa plataforma, sendo portanto usado mais para validação do modelo proposto. Foi decidido, pois, a implementação de um jogo em duas dimensões: de tabuleiro, como por exemplo Go ou damas, que têm regras simples a definir. Em relação ao ambiente de desenvolvimento, apesar de haver várias plataformas, tais como Unity3D[3] e Unreal Engine[4], preferiu-se implementar o jogo diretamente em HTML5 e CSS.

No jogo desenvolvido para o trabalho será disponibilizado um placar, um contador de tempo, um histórico de movimentos, um chat e a opção por comunicação de voz e vídeo. Em relação aos comandos de controle do jogo, o usuário poderá iniciar, pausar e sair da partida, o comando de sair poderá ser feito a qualquer momento, o comando de pausar interrompe qualquer atividade do jogo por um período de tempo, 60 segundos, tempo suficiente para o usuário que deu a pausa fazer algo e que não faça o outro usuário esperar por tanto tempo.

Com este formato pensado para o jogo talvez seja preciso utilizar um modelo publish-subscribe, onde um usuário publica mensagens que são destinados a outros usuários através do

[1] <https://caniuse.com/#search=webrtc>.
[2] <https://developers.google.com/web/tools/chrome-devtools/>.
[3] <https://unity3d.com/>.
[4] <https://www.unrealengine.com/>.

servidor, para isso poderi ser utilizado o *Message Queueing Telemetry Transport* (MQTT) ou *Extensible Messaging and Presence Protocol* (XMPP).

3.3 Requisitos

Após algumas pesquisas envolvendo projetos utilizando WebRTC, foram definidos os requisitos para que o projeto fosse implementado de acordo com o que foi decidido.

3.3.1 Requisitos funcionais

O objetivo deste projeto é desenvolver uma API que permite determinadas funções relacionadas ao desenvolvimento de jogos, permitindo um usuário consegue interagir com outros de maneira prática. Esta seção apresenta os requisitos que foram definidos inicialmente no projeto.

- Estabelecimento de uma comunicação de áudio e vídeo entre dois dispositivos.
- O sistema de chat deverá funcionar em tempo real e deve mostrar a disponibilidade do usuário.
- Em relação as salas, o usuário poderá criar, sair e convidar um usuário disponível a entrar na sala em que ele está.
- Em relação ao jogo o usuário poderá iniciar,pausar e terminar o jogo.

3.3.2 Requisitos não funcionais

Esta seção está relacionada ao uso da aplicação em requisitos importantes mas não primordiais no projeto como segurança, pois não haverá tráfego de informações confidenciais na comunicação e requisitos de desempenho devido a implementação de um jogo de tabuleiro é permitido uma latência até grande na parte de dados do jogo.

- Tratando de segurança nenhum agente externo poderá ter acesso aos dados trafegados.
- O jogo deverá ter uma latência baixa, quase imperceptível ao usuário.

3.4 Cronograma

Na tabela Tabela 1 mostra as etapas pensadas para a conclusão do trabalho proposto, começando a partir das férias.

Mês	Desenvolvimento do jogo 2D	Estabelecimento de uma comunicação de áudio e vídeo no WebRTC	Implementação do framework	Desenvolvimento da API e TCC
Julho	X			
Agosto	X	X		
Setembro		X	X	
Outubro			X	
Novembro			X	X
Dezembro				X

Tabela 1 – Cronograma do trabalho

Referências

BORGES, F. S. *WebRTC: Estudo e Analise do Projeto*. São José, 2013. 57 p. Citado 3 vezes nas páginas 17, 22 e 29.

CASTILLO, I. B.; VILLEGAS, J. M.; PASCUAL, V. *The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)*. [S.l.], 2014. Citado na página 28.

DELPHINI. Citado na página 26.

DOMINGOS, P. V. C. d. A. P. *Análise de Desempenho de Transmissão de Mídia com Protocolos de Criptografia*. São José, 2014. 158 p. Citado na página 30.

FETTE, I.; MELNIKOV, A. *The WebSocket Protocol*. [S.l.], 2011. <http://www.rfc-editor.org/rfc/rfc6455.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc6455.txt>. Citado na página 19.

GRIGORIK, I. *High Performance Browser Networking: What every web developer should know about networking and web performance*. O'Reilly Media, 2013. ISBN 9781449344764. Disponível em: <https://www.amazon.com/High-Performance-Browser-Networking-performance/dp/1449344763?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1449344763>. Citado 3 vezes nas páginas 18, 19 e 20.

HANDLEY, M.; JACOBSON, V.; PERKINS, C. *SDP: Session Description Protocol*. [S.l.], 2006. <http://www.rfc-editor.org/rfc/rfc4566.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc4566.txt>. Citado na página 27.

HOLMBERG, C.; HAKANSSON, S.; ERIKSSON, G. *Web Real-Time Communication Use Cases and Requirements*. [S.l.], 2015. <http://www.rfc-editor.org/rfc/rfc7478.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc7478.txt>. Citado 2 vezes nas páginas 17 e 20.

JúNIOR, B. A. K. *Serviço Distribuído de Áudio e Videoconferência baseado em SIP e WebRTC para Aplicações Web*. São José, 2018. Citado na página 36.

MAHY, R.; MATTHEWS, P.; ROSENBERG, J. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. [S.l.], 2010. <http://www.rfc-editor.org/rfc/rfc5766.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc5766.txt>. Citado na página 32.

MONTEIRO, M. S. M. *TV interativa e seus caminhos*. Campinas, 2002. 84 p. Citado na página 11.

MéSZáROS, M. *WebRTC*. 2014. <https://www.slideshare.net/rootkiskacsa/webrtc-educonf-porto>. Citado na página 32.

Newzoo. *The Global Games Market Will Reach $108.9 Billion in 2017 With Mobile Taking 42%*. 2018. <https://newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42/>. Citado na página 14.

Newzoo. *Mobile Revenues Account for More Than 50% of the Global Games Market as It Reaches $137.9 Billion in 2018*. 2018. <https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/>. Citado na página 15.

ONG, L. et al. *Framework Architecture for Signaling Transport*. [S.l.], 1999. Citado na página 27.

Opus. *Codec Landscape*. 2012. <http://opus-codec.org/comparison/>. Citado 2 vezes nas páginas 22 e 23.

PEREZ, P. R. *Contribución a las arquitecturas flexibles para la multiconferencia web adaptable a la demanda*. Madrid, 2015. 179 p. Citado na página 12.

PROUST, S. *Additional WebRTC Audio Codecs for Interoperability*. [S.l.], 2016. Citado na página 20.

ROSENBERG, J. et al. *Session Traversal Utilities for NAT (STUN)*. [S.l.], 2008. <http://www.rfc-editor.org/rfc/rfc5389.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc5389.txt>. Citado na página 32.

ROSENBERG, J. et al. *SIP: Session Initiation Protocol*. [S.l.], 2002. <http://www.rfc-editor.org/rfc/rfc3261.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc3261.txt>. Citado na página 24.

ROSENBERG, J. et al. *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. [S.l.], 2003. <http://www.rfc-editor.org/rfc/rfc3489.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc3489.txt>. Citado na página 32.

Sam Dutton (HTML5 Rocks). *ICE and WebRTC: What Is This Sorcery? We Explain...*. 2016. <https://temasys.io/webrtc-ice-sorcery/>. Citado na página 33.

SOUTO, R. *WebSockets*. 2013. Acesso em 08/12/2017. Disponível em: <https://prezi.com/hyv8u1sl6qjx/rtfmwebsockets/>. Citado na página 27.

Statista. *Customer preferred gaming platforms according to gaming companies worldwide as of August 2016*. 2018. <https://www.statista.com/statistics/608933/gaming-companies-customer-preferred-gaming-platforms-worldwide/>. Citado na página 13.

Telegeography. *Skype traffic continues to thrive*. 2014. <https://www.telegeography.com/products/commsupdate/articles/2014/01/15/skype-traffic-continues-to-thrive/>. Citado na página 13.

TOGO, R. O. *Implementação e avaliação de cenário de convergência telefonia-rede integrando serviços de VoIP e vídeo chamada com o uso de WebRTC*. São José, 2015. 83 p. Citado 5 vezes nas páginas 17, 19, 20, 28 e 29.

VALIN, J.; BRAN, C. *WebRTC Audio Codec and Processing Requirements*. [S.l.], 2016. Citado na página 20.

VALIN, J.; VOS, K.; TERRIBERRY, T. *Definition of the Opus Audio Codec*. [S.l.], 2012. Citado na página 21.

VARANDA, J. H. de O. *ICE: Uma solução geral para travessia de NAT*. Distrito Federal, 2008. 98 p. Citado na página 32.

WebRTC. *Architecture*. 2018. <https://webrtc.org/architecture/>. Citado na página 18.