

Iago Soares dos Santos Faria

Estudo de caso de teste de compatibilidade de uma aplicação WebRTC

São José - SC

Fevereiro/2024

Iago Soares dos Santos Faria

Estudo de caso de teste de compatibilidade de uma aplicação WebRTC

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

Fevereiro/2024

Iago Soares dos Santos Faria

Estudo de caso de teste de compatibilidade de uma aplicação WebRTC

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

São José - SC, 16 de Fevereiro de 2024:

Prof. Ederson Torresini, Me.
Orientador

Prof. Cleber Jorge Amaral, Dr.
Convidado

Prof. Jorge Henrique Busatto Casagrande
Dr. Eng.
Convidado

São José - SC
Fevereiro/2024

Resumo

Com o passar dos anos a Internet foi incorporando a telefonia e assim se tornou o principal meio de transmissão de mídia das comunicações de voz e vídeo. Atualmente há o crescimento da utilização dos protocolos HTTP e *WebSocket* para esta finalidade, que é desempenhada através do WebRTC, uma ferramenta que é nativa dos navegadores mais populares, criada para funcionar em qualquer dispositivo que os utilize sem a necessidade de *plugins* ou *drivers*, diminuindo a complexidade da implementação de um serviço de comunicação. O objetivo deste trabalho é testar a interoperabilidade do WebRTC em cenários variados, onde há a comunicação entre dispositivos com sistema operacionais distintos e que se encontram em infraestruturas de rede diferentes, analisando os métodos e regras que são aplicados para a resolução dos possíveis conflitos que possam aparecer no estabelecimento da comunicação entre as partes, como endereçamento NAT e *codecs* ofertados. Para este objetivo, foi implementado um servidor Matrix para a sinalização da comunicação. O trabalho também apresenta os resultados das métricas de desempenho do sistema e suas respectivas análises.

Palavras-chave: WebRTC. Interoperabilidade. Matrix.

Abstract

Over the years, the Internet has incorporated telephony and thus has become the main medium for transmitting voice and video communication media. Currently, there is a growing use of the HTTP and WebSocket protocols for this purpose, which is performed through WebRTC. WebRTC is a tool that is native to most popular browsers and was created to work on any device that uses them without the need for plugins or drivers, reducing the complexity of implementing a communication service. The objective of this work is to test the interoperability of WebRTC in varied scenarios, where there is communication between devices with different operating systems and that are located in different network infrastructures. The focus is on analyzing the methods and rules used to resolve conflicts that may arise during the establishment of communication between parties, such as NAT addressing issues and incompatible codecs. To achieve this goal, a Matrix server was implemented for communication signaling. The work also presents the results of the system's performance metrics and their respective analyses.

Keywords: WebRTC. Interoperability. Matrix.

Lista de abreviaturas e siglas

CTIC <i>Coordenadoria de Tecnologia da Informação e Comunicação</i>	41
SIP <i>Session Initiation Protocol</i>	25
HTTP <i>HyperText Transfer Protocol</i>	21
SMTP <i>Simple Message Transfer Protocol</i>	26
RTP <i>Real-Time Protocol</i>	13
API <i>Application Programming Interface</i>	19
W3C <i>World Wide Web Consortium</i>	19
SRTP <i>Secure Real-Time Protocol</i>	20
TCP <i>Transport Control Protocol</i>	21
UDP <i>User Datagram Protocol</i>	21
NAT <i>Network Address Translation</i>	22
ICE <i>Interactive Connectivity Establishment</i>	22
STUN <i>Session Traversal Utilities for NAT</i>	22
TURN <i>Traversal Using Relays around NAT</i>	22

VoIP <i>Voice over IP</i>	27
RTcP <i>Real-time Transport Control Protocol</i>	35
SRTcP <i>Secure Real-time Transport Control Protocol</i>	35
SDP <i>Session Description Protocol</i>	27
VoIP <i>Voice over IP</i>	27
HEVC <i>High Efficiency Video Coding</i>	25
IP <i>Internet Protocol</i>	29
NIST <i>National Institute of Standards and Technology</i>	35
AES <i>Advanced Encryption Standard</i>	35
DCCP <i>Datagram Congestion Control Protocol</i>	29
SCTP <i>Stream Control Transmission Protocol</i>	29
RTT <i>Round-Trip Time</i>	29
OSI <i>Open Systems Interconnection</i>	29
IPv4 <i>Internet Protocol versão 4</i>	31
IPv6 <i>Internet Protocol versão 6</i>	31
W3C <i>World Wide Web Consortium</i>	19

CTIC	Coordenadoria de Tecnologia da Informação e Comunicação	41
AVC	<i>Advanced Video Coding</i>	25

Sumário

1	INTRODUÇÃO	15
1.1	Motivação	17
1.2	Objetivo	18
1.2.1	Objetivos Específicos	18
1.3	Organização do texto	18
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	WebRTC	19
2.1.1	Web API	19
2.1.2	<i>Voice e Video engine</i>	22
2.1.3	Transporte	22
2.2	Codecs	23
2.2.1	Codecs de áudio	23
2.2.2	Codecs de vídeo	23
2.3	Protocolos de sinalização da mídia no WebRTC	25
2.4	Matrix	25
2.4.1	Sinalização Híbrida	27
2.4.2	SDP	27
2.5	WebSockets	28
2.6	Questões a considerar no transporte de dados em tempo real	29
2.7	NAT	29
2.8	Questões a considerar na comunicação entre diferentes tipos de rede	31
2.9	Protocolos de transporte de mídia	33
2.9.1	<i>Real-Time Protocol (RTP)</i>	33
2.9.2	RTcP	35
2.9.3	SRTP/SRTCP	35
2.10	Métricas de desempenho de redes	35
2.10.1	Latência	35
2.10.2	Jitter	36
2.11	4G	36
2.12	5G	37
3	DESENVOLVIMENTO	39
3.1	Descrição geral do sistema	39
3.2	Metodologia	39
3.3	Requisitos	41

3.3.1	Requisitos funcionais	41
3.3.2	Requisitos não funcionais	41
3.4	Construção da infraestrutura	41
3.4.1	Características da infraestrutura	42
3.4.2	Instalação do TURN	42
3.4.3	Instalação do Matrix	43
3.4.4	Ferramentas de medição	47
4	RESULTADOS E TESTES	49
4.1	Codecs usados	50
4.1.1	Caminhos de mídia utilizados	51
4.1.2	Métricas de rede obtidas	53
5	CONCLUSÃO	57
5.1	Trabalhos futuros	58
	REFERÊNCIAS	59
	APÊNDICES	63

1 Introdução

A Internet mudou radicalmente a comunicação humana, tornando a mesma em tempo real como se estivéssemos conversando com alguém do nosso lado. No início, século XIX e meados do século XX, a comunicação tinha que passar por telefonistas que lhe perguntavam para quem gostaria de ligar e manualmente conectava os cabos referentes ao destinatário desejado. Cada região tinha um lugar onde ficavam estas telefonistas, elas sabiam cada morador que estivesse na rede e caso a ligação fosse para um morador fora da rede, havia conexões para outras regiões e lá haveria uma outra telefonista que lhe perguntaria o destinatário, assim acontecia até de fato a ligação chegar na pessoa desejada.

Com o crescimento dos usuários do serviço se tornou complicado um trabalho humano no gerenciamento das conexões e também a complexidade da identificação por nome ou localidade. Então vieram os primeiros comutadores que poupavam minutos, trabalho e pessoa. Sem as telefonistas e não podendo conversar com a máquina e sim o seu dispositivo que enviava o número desejado. Assim nasceu o número e telefone, LDN e LDI, no qual cada DDD por exemplo era uma central. Caso o número discado fosse fora do DDD da região a máquina já repassava para responsável, do mesmo modo que as telefonistas mas muito mais rápido.

E assim permaneceu por anos, os avanços físicos permitiram que estas centrais analógicas se tornassem digitais, proporcionando o acréscimo de velocidade e tornando possível a realizações de chamadas de vídeo, um serviço ainda cara e pouquíssimo utilizado pela população comum.

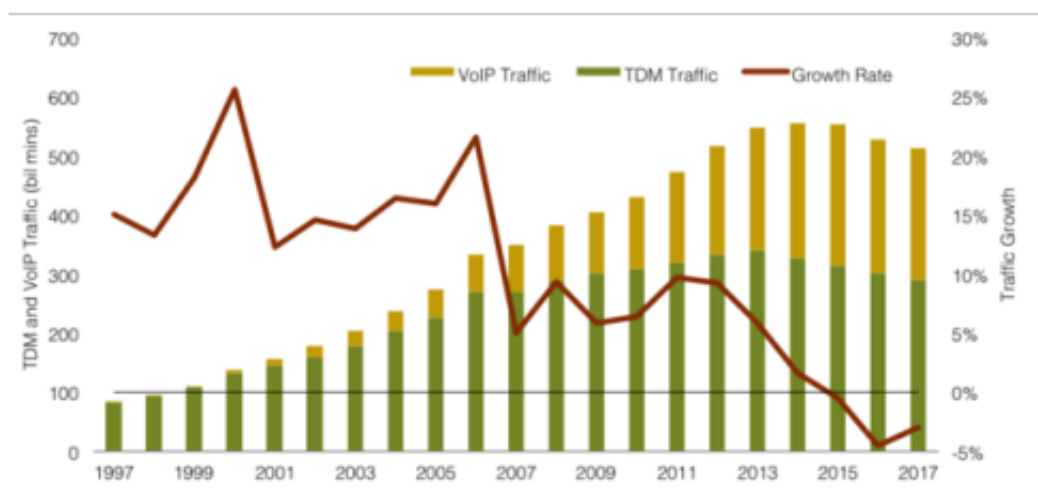
Com o advento da Internet, o número virou um endereço IP e avanços, a rede que

Figura 1 – Telefonista no início do século XX.



Fonte: (A..., 2016).

Figura 2 – Crescimento do VoIP frente a telefonia IP.



Fonte: Pagels (2019).

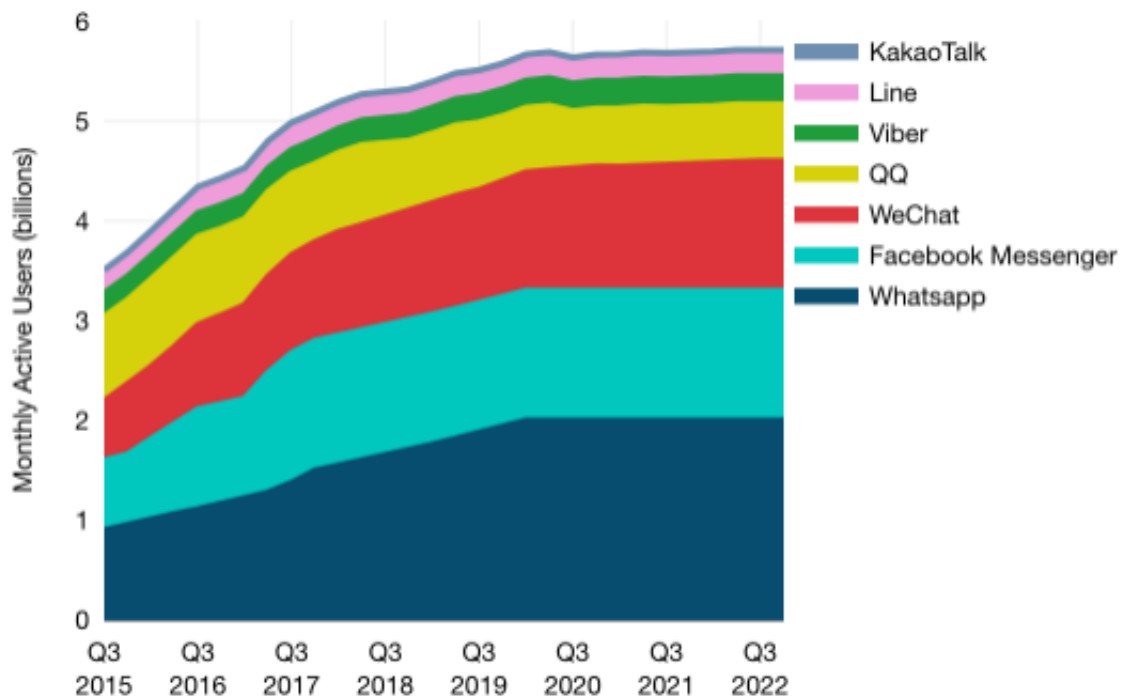
nasceu com o propósito de troca de arquivos, com o tempo houve o surgimento de chats, e na mesma época na telefonia digital começou a ter SMS. Posteriormente a aprimoramento dos dispositivos e protocolos, que permitiam altas velocidades, foram sendo trazido para a Internet serviços da telefonia dando início a telefonia IP, começando com a ligação de voz (VoIP) e logo depois as chamadas de vídeo entre pares, vindo mais tarde chamadas em grupos de áudio e vídeo, como é possível ver na [Figura 2](#).

A proliferação da Internet nos lares, o baixo custo, facilidade e os mesmos serviços da telefonia digital, mensagens de texto e ligações de voz e vídeo, somados a gama de conteúdo que a rede proporcionava através de sites, rede sociais, aplicativos móveis, compartilhamento de arquivos e mídia como músicas e filmes, coisa que não tinham rede telefônica, e poder fazer tudo isso em um mesmo dispositivo, fez que vários usuários passassem a abandonar a telefonia digital, como mostra a [Figura 3](#), a maior parte da população já usa tais aplicativos e todos estes tem a possibilidade de chamadas de voz e/ou vídeo.

A telefonia digital, e posteriormente a IP, utilizavam linguagens de nível mais baixo, assim como a telefonia digital, utilizando por exemplo, Fortran, C e C++, que eram usados nas centrais, nos dispositivos ao longo da rede, nos ATAs e telefones IPs. Os primeiros *softphones*, programas no computador que simulavam telefones físicos, também utilizavam C, C++ e outras linguagens de computadores como Java e *Python*.

Com a popularização do mundo Web e a criação de linguagens determinadas para este meio, como o *Javascript*, veio uma migração destas aplicações para tais linguagens, sendo assim possível a implementação de aplicações semelhantes em navegadores Web. Neste contexto surgiu o WebRTC em 2012. Com o esta nova tecnologia, tornou se mais fácil a comunicação VoIP através de navegadores, fazendo que atualmente os mais usados já venham com *WebRTC* implementado. Agora num simples navegador é possível se comunicar através de voz e vídeo,

Figura 3 – Número de usuários de aplicativos de comunicação.



Fonte: [TeleGeography...](#) (2022).

sem baixar nenhum programa.

Porém os problemas de interoperabilidade só aumentaram com o tempo devido a maior diversidade de dispositivos. Antes a comunicação era feita entre 2 telefones VoIPs, ou entre computadores, ou entre celulares, com o passar do tempo todos estes dispositivos se tornaram elegíveis a se comunicarem, além da variedade de sistemas operacionais de computadores, agora desenvolvedores tem que lidar com esta variedade e mais a variedade de sistemas operacionais mobile, sendo que tais sistemas tem versões diferentes entre outras características que podem gerar problemas. Temos também as particularidades das regras de cada operadora que muitas vezes não permitem o uso de algumas portas ou tipos de conexões. Estes fatos tornaram a comunicação cada vez mais complexa para poder lidar com as particularidades.

1.1 Motivação

O *WebRTC* tem como função a troca de dados de navegador para navegador através de *WebSockets* por meio de canais, por meio destes canais podemos também trocar dados de voz, imagem e mensagens utilizando funções que requisitam o acesso a câmera e microfone do dispositivo de forma nativa, 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*¹, e como qualquer dispositivo ligado a Internet tem um navegador, portanto podemos dizer que o *WebRTC* está presente na maioria os dispositivos ao redor do mundo e muitas vezes o usamos em nossas reuniões online entre família e trabalho.

Porém, como podemos ver, há diversos tipos de dispositivos, de diversas marcas, com sistemas operacionais distintos, estes tem diferença de *codecs*, protocolos e outras etapas. Até mesmo protocolos tem versões diferentes. Diante de tanta diferença, uma aplicação que utiliza *WebRTC* deve ter muitas redundâncias e generalidades para que uma grande quantidade de cenários possa ser abrangida.

Neste trabalho veremos, tomando uma aplicação como exemplo, sua compatibilidade a determinada variedade de dispositivos e arquiteturas de rede.

1.2 Objetivo

Avaliar a interoperabilidade de uma aplicação *WebRTC* entre 2 dispositivos, onde a mídia, áudio e vídeo, será trocada entre os pares em infraestruturas diferentes.

1.2.1 Objetivos Específicos

- Implementar aplicação simples para a realização dos testes.
- Avaliar as métricas de rede como tempo de estabelecimento da conexão, perda de pacotes, *jitter*, pacotes fora de ordem.

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 de testes proposto baseado nos conceitos abordados na fundamentação teórica e uma visão geral do sistema.

¹ <<https://caniuse.com/#search=webrtc>>, acessado em 20/05/2023.

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 um projeto conduzido por vários desenvolvedores de navegadores para oferecer uma solução de comunicação em tempo real nativa dessas aplicações, definido no documento da [W3C \(2023\)](#) e na [Standardization \(2023\)](#), capaz de fazer a comunicação de qualquer tipo de dado, como áudio, vídeo, arquivos, entre outros, tudo isso utilizando canais bidirecionais para trocas e dados entre os dispositivos.

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.

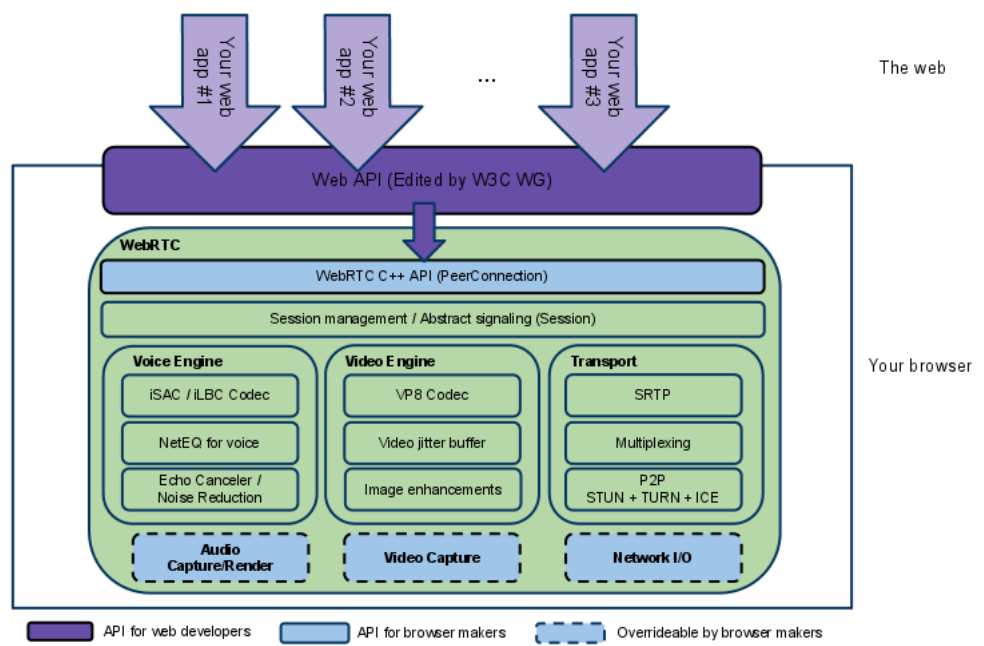
O *WebRTC* tem preferência no uso de ferramentas de código aberto, seu uso se dá através da linguagem JavaScript através de API, que dão ao desenvolvedor o acesso a recursos como câmeras e microfones do dispositivo, que foram definidas pela *World Wide Web Consortium* ([W3C](#)). Sua arquitetura é composta por 2 partes: a primeira é a WebAPI onde estão as funções utilizadas pelos desenvolvedores para implementar suas aplicações.

A [Figura 4](#) mostra as seções dentro da arquitetura do WebRTC. Na seção *Transport* é tratado sobre a forma de transmissão e recepção dos dados, com o quadro P2P mostrando as 3

2.1.1 Web API

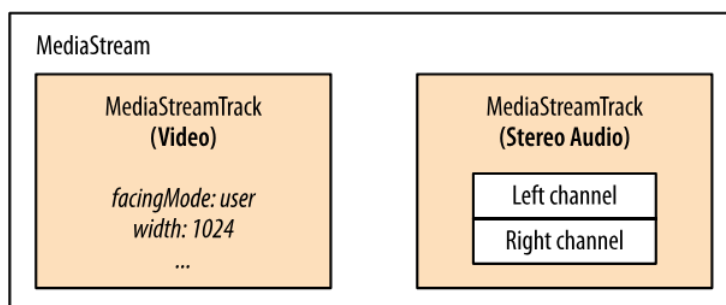
A *Web Application Programming Interface* ([API](#)), utilizada no desenvolvimento de aplicações de comunicação com áudio e vídeo em navegadores, é dividida em três partes: *MediaStream*, *PeerConnection* e *DataChannels*. Essas partes usadas de modos diferentes podem dar diversas funcionalidades as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

Figura 4 – Arquitetura WebRTC.



Fonte: WebRTC (2018).

Figura 5 – Blocos do MediaStream.



Fonte: Grigorik (2013).

MediaStream

O *MediaStream* é função do WebRTC que fornece uma fonte de dados para o WebRTC (BORGES, 2013). É responsável por requerir os dados gerados pelo microfone e/ou câmera do dispositivo, que serão enviados através dele utilizando o protocolo de transporte *Secure Real-Time Protocol (SRTP)*, que é uma exigência do WebRTC. 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 5 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). O *MediaStream* formado por um ou mais *MediaStreamTrack* como mostrado na figura ou seja, numa transmissão de vídeo com som, há no mínimo

Figura 6 – Exemplo de código do MediaStream.

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

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

Fonte: Grigorik (2013).

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 6 é visto um exemplo de código em JavaScript do MediaStream onde é requisitado ao usuário o acesso à alguma fonte de mídia.

PeerConnection

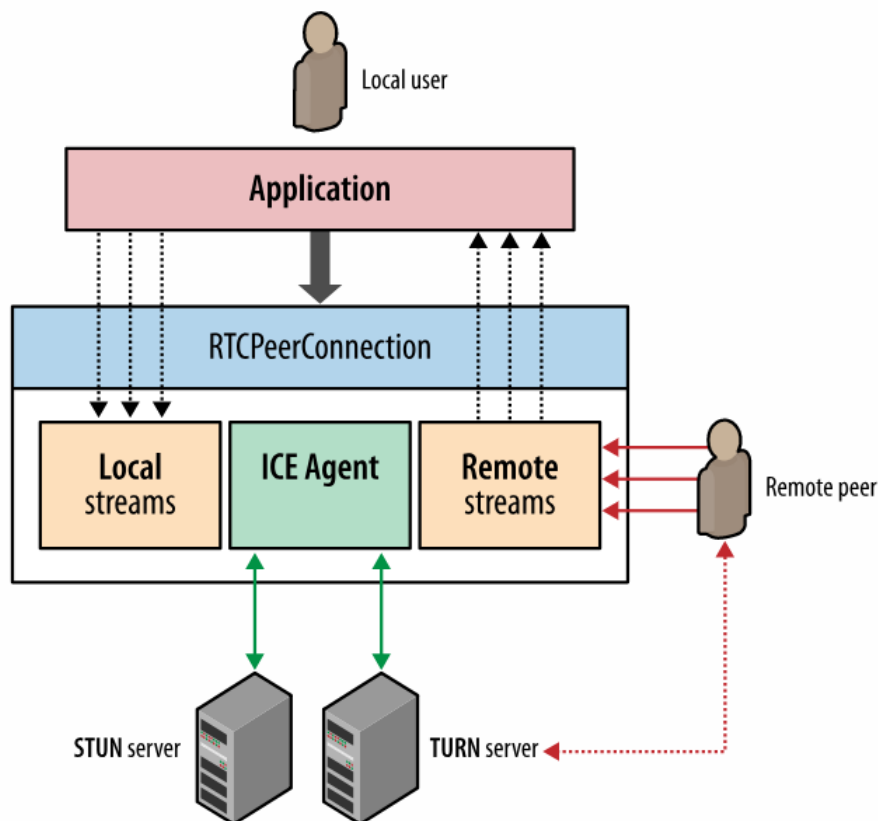
O *PeerConnection* é a parte da *WebAPI* que estabelece uma conexão estável e eficiente entre dois navegadores. Ele faz a negociação dos codecs que serão utilizados, criptografia e gerenciamento de banda, protegendo os desenvolvedores *Web* de diversas complexidades que existem ao se fazer uma conexão fim-a-fim (TOGO, 2015), também é feito em JavaScript de acordo com Figura 8. 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 7.

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 *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 que envolvem arquivos, mensagens instantâneas, compartilhamento de tela.

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.

Figura 7 – Blocos do PeerConnection.



Fonte: Grigorik (2013).

Figura 8 – Exemplo de código do PeerConnection.

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

Fonte: Grigorik (2013).

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. Segundo 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 Address 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), que serão abordados mais a frente no capítulo.

2.2 Codecs

Codec é a abreviação de *codificador/decodificador*, 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 para que seja possível a reprodução da mídia 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.

2.2.1 Codecs de áudio

Como 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 usuários que usariam este *codec* específico, não todos. Ultimamente os *codecs* tidos como obrigatórios no *WebRTC* são¹:

- Opus, um *codec* definido por 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.

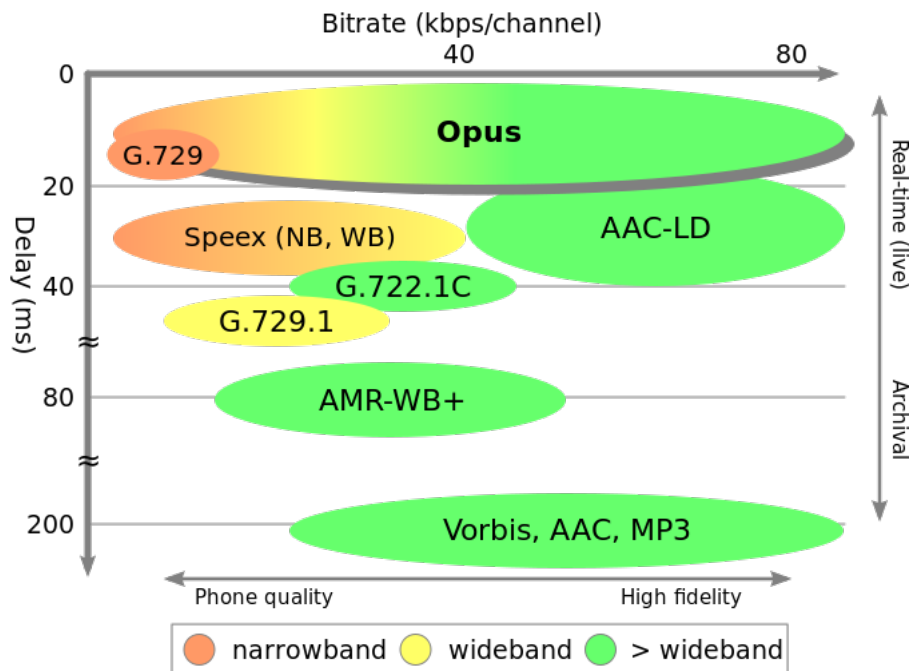
O Opus tem uma excelente qualidade em praticamente todas as faixas de banda analisadas como visto na Figura 10, isso faz com que ele seja em muitos casos a melhor opção quando se busca qualidade de áudio e baixo *delay*, como mostra Figura 9, conforme Valin e Bran (2016), o Opus deve ser um *codec* presente em qualquer comunicação por voz *WebRTC*.

2.2.2 Codecs de vídeo

De acordo com a RFC 7742 (ROACH, 2016), requisita que numa comunicação *WebRTC* com vídeo é necessário ao menos ter os *codecs* VP8 e H.264, que já são os mais utilizados.

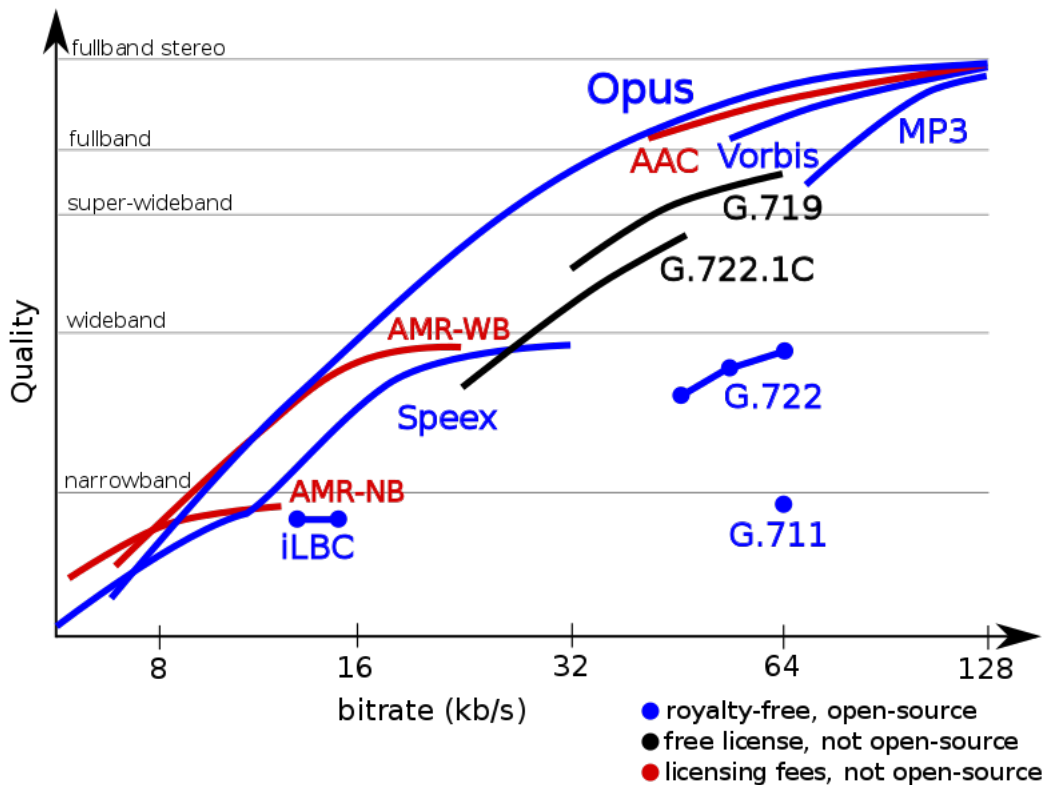
¹ <<https://www.heise.de/newsticker/meldung/Zwei-Audio-Codecs-fuer-Echtzeit-Kommunikation-im-Browser-165661.html>>

Figura 9 – 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).

Figura 10 – Comparação do Opus em relação a outros *codecs* em relação a quantidade usada de banda para determinada qualidade áudio.



Fonte: Opus (2012).

- VP8, é a evolução do *codec* VP7, foi criado pela *On2 Technologies*, empresa que em 2010 foi comprada pela Google, sendo concebido previamente com código proprietário e após a compra seu código foi aberto, por este fato é bastante utilizado no sistema operacional *Android*, que prioriza código aberto, e pelos serviços oferecidos pela Google como Youtube, sendo o principal *codec* ofertado nas reprodução dos vídeos da plataforma.
- H.264, *Advanced Video Coding* (AVC), definido pelo ITU-T em 2004, vem de uma família de *codecs* mais utilizada atualmente na distribuição de vídeos em alta definição, na indústria de jogos e edição de vídeo.

Cada um dos *codecs* tem seus pontos positivos e negativos, H.265 é geralmente implementado em *hardware* fazendo que o seu consumo de energia seja significativamente menor, enquanto o VP8 geralmente é implementado em *software*, que consome mais energia. As grandes vantagens do VP8 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.264 é necessário licença.

Quando o projeto do *WebRTC* foi concebido os *codecs* mais populares da época ainda são o VP8 e H.264 *codecs* citados acima, porém já há as versões mais novas de cada um, VP9 e H.265 respectivamente. O VP9, assim como o VP8, é implementado em *software*, portanto numa atualização de *software* automaticamente o dispositivo poderia usar o VP9. No caso do H.265, também conhecido como *High Efficiency Video Coding* (HEVC), assim como o H.264, é implementado em *hardware*, portanto é presente apenas em dispositivos mais novos, mantendo assim o suporte ao H.264. O VP9 e o H.265 foram desenvolvidos para reduzir em 50% a taxa de bits utilizada pelos seus antecessores dada certa resolução.

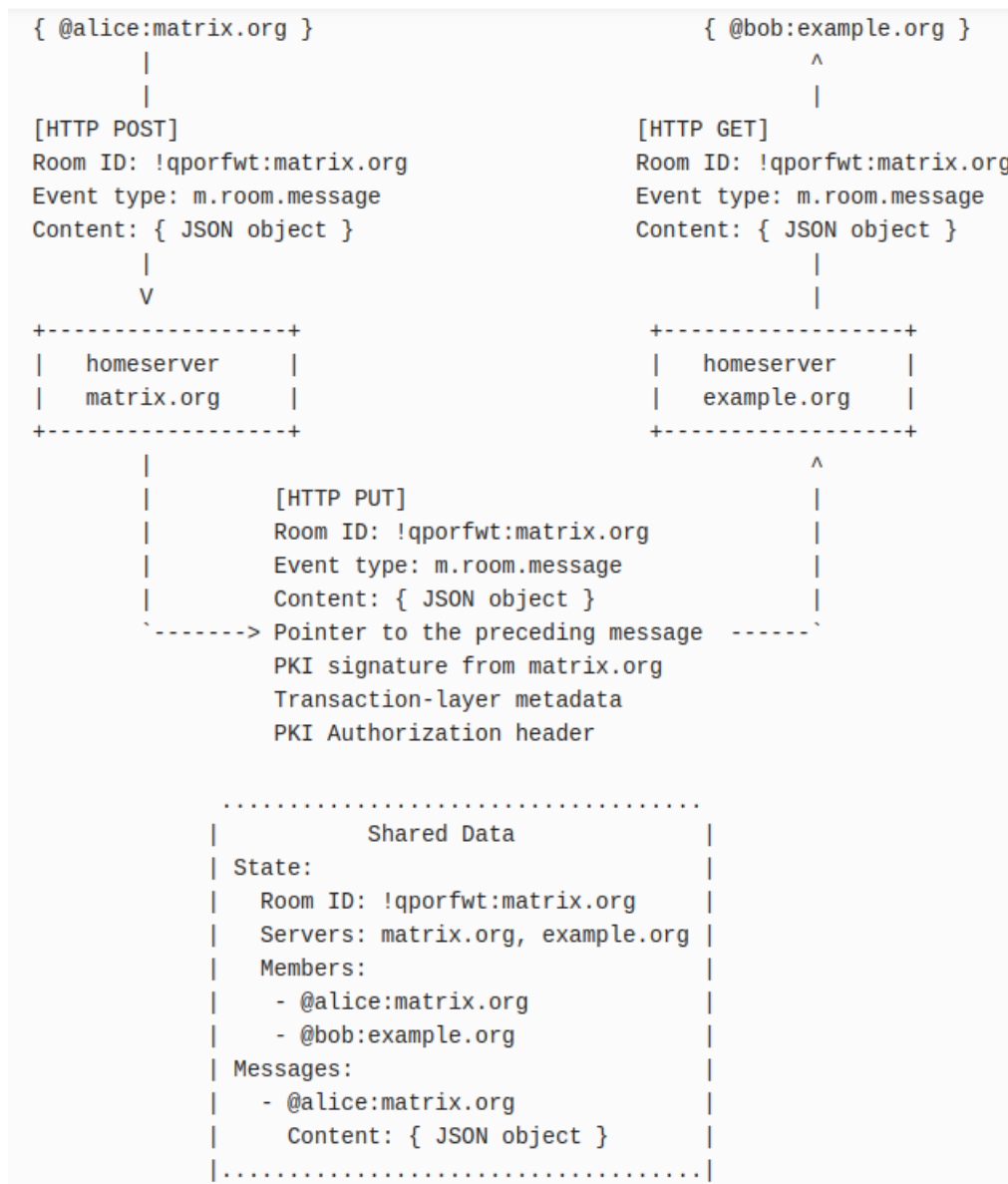
2.3 Protocolos de sinalização da mídia no WebRTC

Como visto o *WebRTC* é uma ferramenta que permite a aquisição de dados de mídia do computador e cria canais para envio de dados, portanto não há sinalização, fazendo que esta parte seja de responsabilidade do desenvolvedor que pode criar uma sinalização própria para o gerenciamento da mídia, podendo até ser baseada em sinalizações já conhecidas. Há diversos protocolos VoIP comuns hoje, como o SIP e H.323, há outros protocolos que seriam úteis como o MGCP, Jingle, XMPP, Matrix, entre outros, além da possibilidade de haver uma sinalização híbrida.

2.4 Matrix

O Matrix é um protocolo que permite a construção de redes descentralizadas e suporta comunicações em tempo real de envio de mensagens, chamadas de voz e vídeo. A arquitetura do Matrix é cliente-servidor e tem semelhanças com protocolos como *Session Initiation Protocol*

Figura 11 – Arquitetura do Matrix.



Fonte: [CIC \(2023\)](#).

(SIP) e Simple Message Transfer Protocol (SMTP), do e-mail, onde há servidores que seriam os responsáveis por mediar as chamadas e servir de *proxy* conectando a usuários em outros servidores.

Assim como o SIP, o usuário se autentica no servidor, após isso como mostrado na [Figura 11](#), sua sinalização usa o protocolo HTTP para realizar funções de gerenciamento de salas, estas são as principais funções:

- **CREATE:** Tem a função de criar a sala.
- **INVITE:** Função que envia o convite a um usuário
- **JOIN:** Função que requisita a entrada do usuário na sala

- LEAVE: Função usada para que o usuário deixe a sala.
- KICK: Removido um usuário da sala.
- BAN: Dá ao usuário um *status*, caso estiver na sala será removido, se tiver fora não será possível entrar.
- UNBAN: Retorna as características adicionais do usuário que estava banido.

Com exceção do CREATE, em todos os outros comandos é passado o *roomID*, identificador da sala. Na função KICK, o usuário pode voltar a sala.

O Matrix tem um sistema de *power level*, que é a hierarquia, onde nem todos os usuários podem banir ou realizar um KICK em qualquer um, mantendo assim uma ordem no servidor, o criador da sala tem *power level* 100, enquanto o usuário comum é 0 por padrão. Tais valores podem ser modificados.

2.4.1 Sinalização Híbrida

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

2.4.2 SDP

O *Session Description Protocol* (SDP) é um protocolo texto da camada de aplicação, definido por 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 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.

2.5 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).

Como 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 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, como ligações de voz e vídeo online (TOGO, 2015).

Nos protocolos de transporte como o TCP e UDP já existe o conceito de Sockets, que é canal aberto entre 2 dispositivos onde é possível a transmissão de qualquer tipo dado.

Tal solução não havia para protocolos de aplicação como o HTTP e para solucionar esse problema foi criado o *WebSocket*, que é uma tecnologia que permite a comunicação nos dois sentidos de uma conexão HTTP, entre cliente e servidor. Ou seja, a qualquer momento tanto cliente quanto 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.

Este canal provido pelo *WebSocket* pode ser usado para transmitir dados mais importantes de uma aplicação *WebRTC*, já que uma quantidade mínima de dados referentes a voz e vídeo podem ser perdidos sem que a ligação seja finalizada, estes dados podem ser transmitidos pelo meio comum, enquanto os dados de estabelecimento da conexão são extremamente importantes, a perda de qualquer pacote de sinalização acarreta na não iniciação ou falha na manutenção da mesma, o *WebSocket* se torna uma ótima ferramenta para garantir um estabelecimento seguro de uma conexão *WebRTC*.

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 é 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 viáveis devido a falta de suporte que os roteadores têm a esses protocolos (DOMINGOS, 2014).

2.7 NAT

O NAT, definido por Srisuresh e Holdrege (1999) e criado em 1994, é um mecanismo de tradução de endereços de rede, com o objetivo de possibilitar conectividade entre nós de uma rede privada com a Internet (TOGO, 2015).

O NAT foi concebido para ser uma solução a falta de endereços IPv4 sem que fosse necessário alterar toda a topologia já existente, pois o endereço tem 32 bits permitindo no máximo um pouco mais que 4 bilhões de endereços diferentes. Com esse conceito um IP válido poderia representar muitos dispositivos aumentando a capacidade da rede IPv4.

A especificação do uso de NAT se dá em IPv4 (REKHTER et al., 1996) e em IPv6 (HABERMAN; HINDEN, 2005). O NAT faz que o dispositivo tenha um endereço e porta diferentes na rede local e na rede externa, onde o endereço na rede externa é correspondente ao IP do roteador de borda da rede local.

Como o NAT permite que um dispositivo tenha endereço diferente do original mascarando o verdadeiro, com o tempo o NAT foi utilizado com propósitos de segurança apesar de não ser seu apelo original.

De acordo com o tráfego de pacotes que são roteados, o roteador preenche uma tabela de roteamento relacionando o IP dos dispositivos da rede local com os IPs recebidos da rede externa. Então ao receber uma resposta oriunda da rede externa de um endereço de origem específico, o roteador é capaz de identificar para qual dispositivo da rede local esse pacote é destinado.

Existem 4 tipos de NAT que são:

- *Full-cone* NAT: é o único tipo de NAT onde as portas internas são mapeadas no endereço externo.
- *Restricted-cone* NAT: nele o endereço interno é mapeado no endereço externo, mas para algum dispositivo externo estabelecer uma conexão com um dispositivo que está nesta rede primeiro o dispositivo desta rede deve se conectar ao dispositivo externo para a troca de pacotes. Portanto este tipo de NAT bloqueia qualquer outro dispositivo externo que deseja se conectar a dispositivos na rede interna sem que nenhum deles tenha feito contato com esses dispositivos.
- *Port-restricted-cone* NAT: o endereço IP e das portas internas são mapeados no endereço externo, mas para algum dispositivo externo estabelecer uma conexão com um dispositivo que está nesta rede primeiro o dispositivo desta rede deve se conectar ao dispositivo externo para a troca de pacotes e quando o dispositivo externo desejar trocar pacotes com ele deveria utilizar a mesma porta em que foi estabelecida a primeira conexão. Portanto este tipo de NAT bloqueia qualquer outro dispositivo externo que deseja se conectar a dispositivos na rede interna sem que nenhum deles tenha feito contato com esses dispositivos ou que já tenha feito contato mas a porta utilizada por este dispositivo seja diferente a porta usada no primeiro contato.
- *Symmetric* NAT: neste tipo de NAT os endereços IP e de porta internos são mapeados no endereço externo mas caso haja mais de uma conexão que se comunique com este dispositivo na mesma porta interna, haverá o mapeamento distinto para as 2 conexões. Para se enviar pacotes ao dispositivo da rede interna o dispositivo externo já deve ter tido recebido um pacote do dispositivo da rede interna. Portanto este tipo de NAT bloqueia

qualquer outro dispositivo externo que deseja se conectar a dispositivos na rede interna sem que nenhum deles tenha feito contato com esses dispositivos.

O NAT é um grande problema em comunicações em tempo real quando o cenário opera sobre IPv4, inclusive para o SIP e o WebRTC. Para resolver esse problema temos os servidores ICE, STUN e TURN (TOGO, 2015).

2.8 Questões a considerar na comunicação entre diferentes tipos de rede

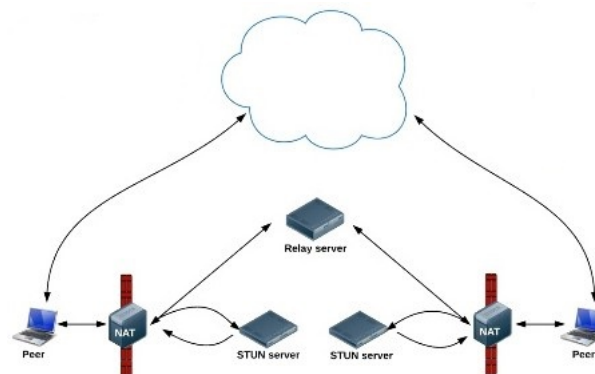
Um dos grandes problemas das comunicações na Internet é quando uma das partes se encontra atrás de NAT ou *firewalls*. Uma comunicação com WeRTC pode ser facilmente bloqueada por um *firewall*, já que por ser um mecanismo de segurança da rede sua função é justamente supervisionar qualquer pacote que 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 12. 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.

Figura 12 – Arquitetura WebRTC.



Fonte: Mészáros (2014).

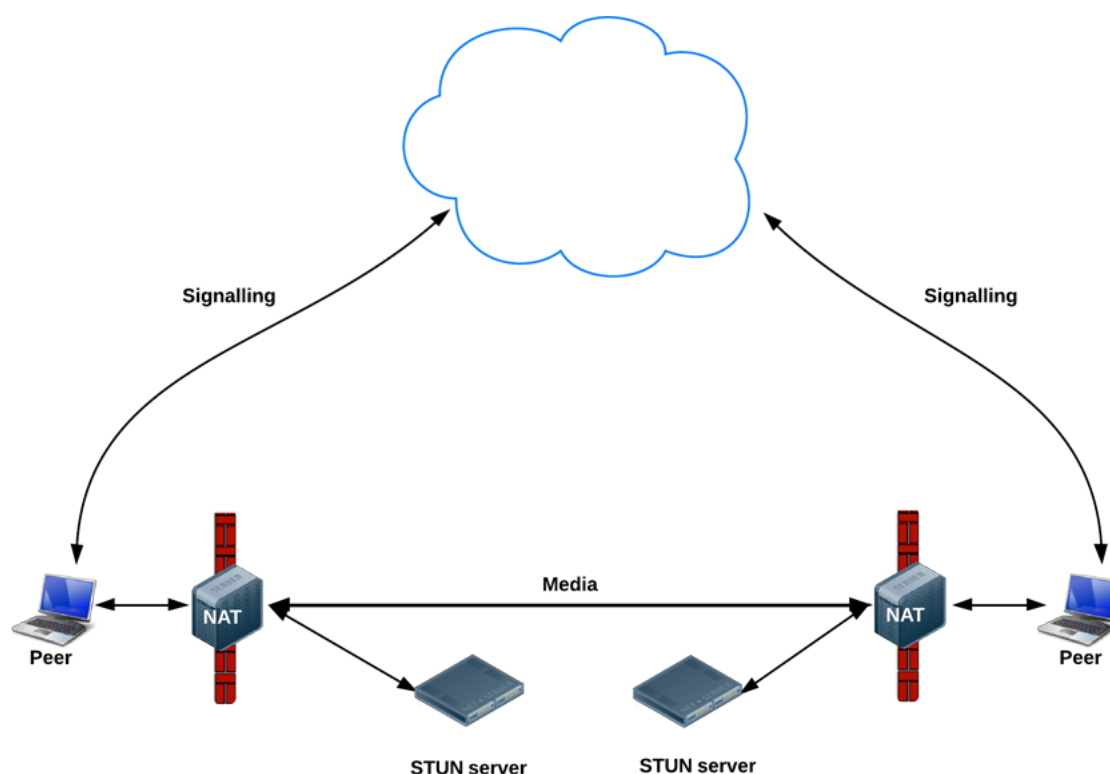
- Checagem de conectividade: após receber os endereços candidatos cada um dos agentes ICE dos usuários identifica os candidatos locais e remotos, depois começa a fazer testes de conectividade seguindo uma ordem de acordo com a categoria e prioridades da conexão, isso é feito usando o protocolo STUN.
- 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 um protocolo para solucionar o problema de comunicação atrás de NAT e pode ser usado em conjunto com o ICE, além de servir como base a ele. O STUN foi definido primeiramente por Rosenberg et al. (2003) e atualizada por 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 13 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 *restrictive cone*, tanto NAT quanto *port*, é preciso que o dispositivo envie um pacote a outra parte da conexão.

O TURN é um protocolo para solucionar os problemas de comunicações atrás de NAT. Definido por Mahy, Matthews e Rosenberg (2010), atua como um intermediário na

Figura 13 – Arquitetura WebRTC.



Fonte: [Sam Dutton \(HTML5 Rocks\)](#) (2016).

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 quando não é possível uma conexão direta entre os mesmos, os clientes requisitam portas ao servidor TURN para serem usadas no transporte das mídias, portanto consome muitos recursos do servidor, como processamento e largura de banda, sendo uma opção menos recomendável para estabelecer a comunicação entre dois agentes ([VARANDA, 2008](#)), como mostra a [Figura 14](#).

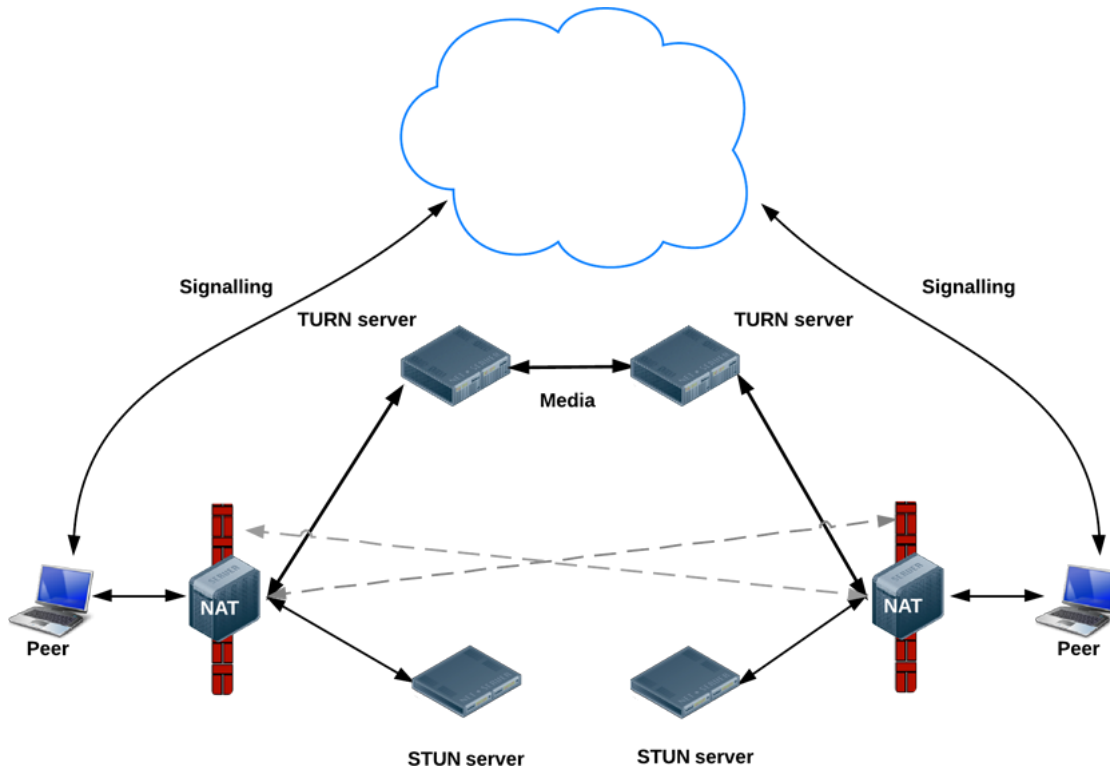
2.9 Protocolos de transporte de mídia

Os protocolos de transporte de mídia são diferentes dos protocolos de transporte convencionais como TCP, UDP e SCTP, seu propósito é apenas enviar mídia e necessitam dos protocolos citados para endereçamento IP e portas.

2.9.1 RTP

O [RTP](#), definido por [Schulzrinne et al. \(2003\)](#), tem como função o transporte de mídias em tempo real, como áudio, vídeo ou dados. Usa geralmente [UDP](#) como protocolo de transporte, uma vez que esse não é sensível a perda nem é orientado a conexão, sendo melhor em relação a atrasos de envio e recebimento de dados que o protocolo TCP que implementam garantia de entrega, controle de fluxo e congestionamento que consequentemente

Figura 14 – Arquitetura WebRTC.



Fonte: [Sam Dutton \(HTML5 Rocks\)](#) (2016).

causa aumento de latência da rede com a utilização de *buffers* afetando a instantaneidade das mensagens referente a chamada. Os principais serviços providos pelo RTP são:

- Diversos tipos de *payloads*: cada um deles é destinado a um conteúdo específico.
- Sequenciamento de pacotes: otimiza a reconstrução dos fluxos multimídia, que podem desordenar ao longo do caminho, evitando a perda de continuidade da reprodução do som.
- Monitoramento da entrega: através do método do item acima, faz que a aplicação tenha o conhecimento da taxa de perdas da conexão.
- Marcação temporal: permite a sincronização e o cálculo da variação de atraso.

A definição das portas para o tráfego da mídia é feita pelos protocolos de sinalização, no caso o SDP, não pelo RTP, o que esse é capaz de fazer em relação a definição de portas é alocar portas pares para o protocolo UDP. Como a sinalização o tráfego de mídia também sofre a interferência da ação do NAT e do *firewall*.

2.9.2 RTcP

O *Real-time Transport Control Protocol* (RTcP) é um sub-protocolo de controle do fluxo RTP definido por Schulzrinne et al. (2003) juntamente com o RTP, sua função é monitorar a qualidade do serviço e transportar informações dos participantes de uma sessão RTP em andamento, semelhante ao ICMP. Suas principais funções são:

- Retorno sobre o desempenho da aplicação e da rede: pode modificar a taxa de bits do fluxo RTP, consequentemente a qualidade da mídia, para que não haja um número elevado de perdas, através de envio de pacotes periodicamente para monitoramento da rede.
- Informações para sincronização de fluxos de áudio e vídeo: evita a sensação incômoda do áudio não corresponder com as ações que acontecem no vídeo.

2.9.3 SRTP/SRTCP

O SRTP e o *Secure Real-time Transport Control Protocol* (SRTcP) são protocolos de transporte de mídia, definidos por Baugher et al. (2004). Sua função é garantir integridade, confiabilidade e segurança aos dados que trafegam em fluxo RTP/RTcP usando criptografia e autenticações evitando que terceiros capturem os dados trafegados, e são obrigatórios em uma conexão utilizando WebRTC.

Utiliza o sistema *Advanced Encryption Standard* (AES) como codificador/decodificador padrão da mídia, um sistema validado e definido como padrão criptográfico em 2001 pelo *National Institute of Standards and Technology* (NIST).

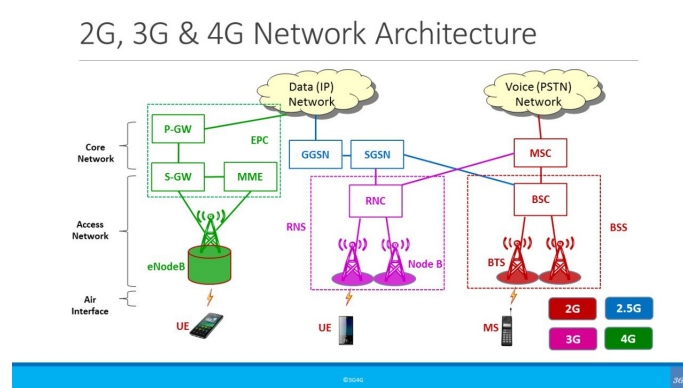
2.10 Métricas de desempenho de redes

2.10.1 Latência

A latência de rede é o tempo que leva para um pacote de dados viajar de um ponto para outro em uma rede. A latência pode variar dependendo de vários fatores, como a distância física entre os dispositivos, o meio da transmissão de dados, o modelo ou a qualidade ou a capacidade da infraestrutura de rede, a quantidade de tráfego na rede e a eficiência dos protocolos de comunicação utilizados.

Quando a latência é alta, os pacotes de dados levam mais tempo para chegar ao seu destino, o que pode causar atrasos perceptíveis e afetar negativamente a experiência do usuário, especialmente em aplicativos que exigem uma comunicação em tempo real, como chamadas de voz ou videoconferências.

Figura 15 – Arquitetura das redes móveis 2G, 3G e 4G.



Fonte: High-level... ().

2.10.2 Jitter

Jitter é a variação no atraso de entrega dos pacotes de dados em uma rede, enquanto a latência se concentra no tempo total de viagem dos pacotes, o *jitter* se concentra nas variações desse tempo. A causa do *jitter* pode ser acontecimentos pontuais como congestionamento de rede, problemas de roteamento, interferências ou flutuações na capacidade de processamento dos dispositivos de rede.

Altos níveis de *jitter* podem ter um impacto negativo em aplicativos que dependem de uma comunicação em tempo real, como chamadas de voz pela Internet (VoIP), videoconferências ou transmissões ao vivo. O *jitter* excessivo pode levar a atrasos, interrupções ou problemas de sincronização durante a comunicação, afetando a qualidade e a experiência do usuário.

Para mitigar o *jitter*, é possível utilizar *buffers* de armazenamento nos dispositivos, que ajudam a minimizar as variações no atraso de entrega dos pacotes e a manter uma comunicação mais estável e consistente.

2.11 4G

O 4G, foi uma tecnologia lançada em 2009 que foi a primeira que utiliza a Internet para todos os serviços de telefonia. Anteriormente a parte de dados, isto inclui ligações por aplicativos que usam Internet como *Whatsapp* ou *Skype*, ia para a Internet, enquanto as ligações realizadas pelo número do celular iam para a *PSTN*, a rede de telefonia digital, com o possível ver na Figura 15. A partir do 4G ambos os dados passaram a utilizar a internet. Dentro do 4G temos o *LTE*, que é uma das tecnologias que compõe esta arquitetura.

2.12 5G

O 5G surgiu em 2018 e assim como o 4G funciona inteiramente sobre IP, e tem uma arquitetura e frequências diferentes que permitem mais velocidade e conectividade, com o foco em baixa latência (5G...,).

3 Desenvolvimento

Este capítulo apresenta o que foi 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 análise de cenários onde diferentes dispositivos em redes distintas estabelecem uma comunicação *WebRTC*.

O primeiro cenário consiste entre 2 dispositivos, onde haverá transmissão de áudio e vídeo na comunicação entre eles, 1 será um dispositivo que estará na rede móvel com IP público e outro um computador com IP privado.

No segundo cenário terão 2 dispositivos, onde haverá transmissão de áudio e vídeo na comunicação entre eles, e ambos serão computadores porém estarão em redes privadas diferentes.

Entre tais dispositivos terá um único servidor no qual a sinalização e mídia irão passar, a arquitetura do sistema proposto mostrada na [Figura 16](#).

3.2 Metodologia

Os dispositivos usados serão 2 computadores, um *Windows* outro *Linux*, e um dispositivo móvel *Android*.

Atualmente, os navegadores mais populares já têm suporte ao *WebRTC*¹. Diante disso, o que definiu os navegadores para este trabalho foram as funcionalidades oferecidas de desenvolvimento, depuração de código, representação e popularidade, que serão o Google Chrome e o Mozilla Firefox.² Ambos navegadores satisfazem os requisitos, o *Chrome* representa a família de navegadores que usam a *engine Blink*³ enquanto o *Firefox* representa os navegadores que usam *Gecko*⁴. Esta diferença de *engines* não deve causar muitos problemas.

Será usado o Matrix para fazer a sinalização de tais dispositivos, já que o *WebRTC* não tem uma sinalização.

Sobre o tipo de conexão a ser usada no trabalho, foram escolhidos os tipos mais comuns que é a rede doméstica com endereço privado nos dispositivos que estão conectados nela, e a

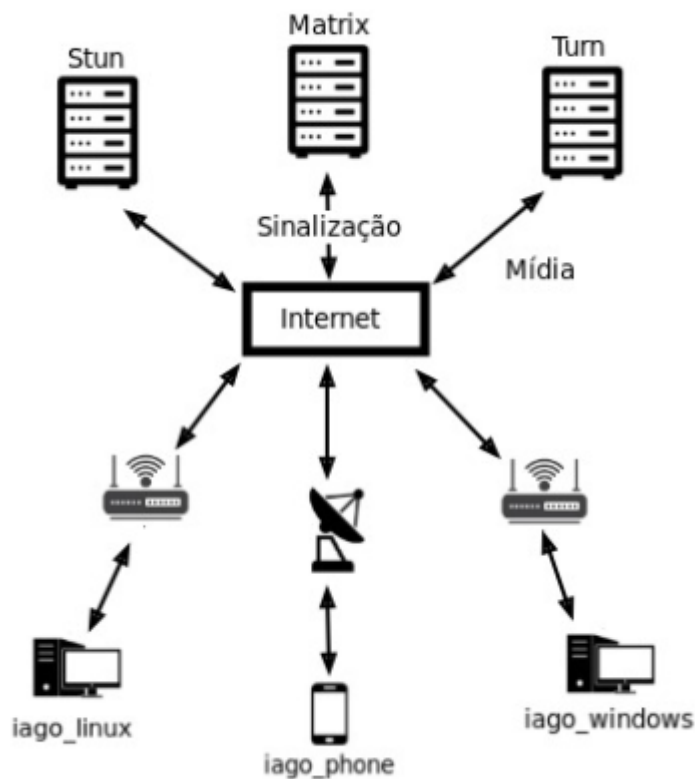
¹ <<https://caniuse.com/#search=webrtc>>.

² <<https://developers.google.com/web/tools/chrome-devtools/>>.

³ <<https://www.chromium.org/blink/>>.

⁴ <<https://developer.mozilla.org/pt-BR/docs/Glossary/Gecko>>.

Figura 16 – Exemplo do sistema proposto no trabalho.



Fonte: Autoria própria.

rede móvel no 4G e 5G.

Os testes a serem feitos irão observar estes pontos:

- Tempo de estabelecimento da conexão
- Troca de mídia, garantir que os dados estão chegando no outro *peer*
- RTT
- *Jitter*
- Perda de Pacotes
- Pacotes fora de ordem
- Oferta de caminhos
- Oferta de *codecs*

Alguns desafios que podem ser enfrentados são os problemas referente a rede dependendo do tipo de NAT que é usado em uma das partes da conexão.

A escolha de apenas 2 navegadores, 2 infraestruturas de rede e 3 sistemas operacionais, foi feita para que a quantidade de testes entre as possibilidades não fosse grande a ponto de tornar o projeto algo extenso, impossibilitando o término no período proposto. Quanto mais diferenças temos maior é probabilidade de problemas causados pela interoperabilidade, para restringir estas ocorrências foi que houve esta delimitação.

3.3 Requisitos

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

3.3.1 Requisitos funcionais

Como o objetivo do trabalho é verificar a compatibilidade de diferentes dispositivos em diferentes redes em comunicações que usam *WebRTC*, Esta seção apresenta os requisitos que foram definidos inicialmente no projeto.

- Possibilitar a comunicação de voz e vídeo dos usuários que estão logados no servidor.

3.3.2 Requisitos não funcionais

Esta seção está relacionada ao uso da aplicação em requisitos importantes mas não primordiais no projeto.

- Possibilidade de chamadas com mais de 2 usuários com boa qualidade.

3.4 Construção da infraestrutura

Tomando como base a [Figura 16](#), o sistema deve ter um servidor e para isto foi utilizada uma máquina cedida pela Coordenadoria de Tecnologia da Informação e Comunicação (CTIC), um Ubuntu, que já vem com as portas de número mais alto abertos para serem usadas pelo RTP e também endereço IPv6. Já que o WebRTC funciona sobre HTTPS e este protocolo necessita de um certificado, foi configurado um DNS que foi o `https://tcciago.sj.ifsc.edu.br/`, pois o certificado só pode ser criado para um DNS e não um IP.

3.4.1 Características da infraestrutura

- Máquina cedida pelo IFSC é um Ubuntu 22.04., kernel 6.2.16-10, possui endereços IPv4 e IPv6. Porém o TURN apenas IPv4.
- Dispositivo Linux é um Ubuntu 23.04, *kernel* 6.2.0-39, navegador Chrome versão 121, possui endereços IPv4 e IPv6.
- Dispositivo *Windows* é um 11 Pro, versão 23H2, possui endereços IPv4 e IPv6.
- Dispositivo móvel é um *Android* na versão 14, possui endereços IPv4 e IPv6.

Para melhorar a autenticidade dos testes e torná-los mais interessantes, um dos computadores utilizará um endereço remoto de um outro país, permitindo assim testar distâncias longas, ao invés do cenário real onde os mesmos estão numa rede igual, possibilitando analisar melhor os tempos de latência e *jitter*.

3.4.2 Instalação do TURN

Devido aos cenários propostos estarem inseridos em redes com NAT, foi escolhido o TURN para ser o resolutor dos possíveis problemas que podem haver neste tipo de ocasião, portanto no mesmo servidor onde está o Matrix foi instalado o servidor TURN, usado como redirecionador do conteúdo da comunicação.

O fator primordial da escolha por este método de resolução foi devido a sua simples instalação, pois há um pacote nos repositórios padrão do Ubuntu, sendo apenas necessário executar o comando 3.1. A instalação tomou cerca de poucas horas.

Código 3.1 – Comando de instalação TURN

```
1 apt install coturn
```

A configuração do servidor fica no arquivo localizado neste arquivo 3.2 e o conteúdo do arquivo é este 3.3. As configurações importantes estão listadas abaixo.

Código 3.2 – Arquivo de configuração TURN

```
2 /etc/turnserver.conf
```

Código 3.3 – Conteúdo do arquivo de configuração do TURN

```
3 turns.server.conf
4 cli-password=tcciago
5 # Rede
6 listening-ip=0.0.0.0
7 listening-ip=::
```

```
8 # Transporte
9 listening-port=3478
10 min-port=49152
11 max-port=65535
12 # Logging
13 syslog
14 verbose
15 # Identificação do servidor
16 server-name=tcciago.sj.ifsc.edu.br
17 realm=tcciago.sj.ifsc.edu.br
18 # TLS
19 fingerprint
20 cert=/etc/letsencrypt/live/tcciago.sj.ifsc.edu.br/fullchain.pem
21 pkey=/etc/letsencrypt/live/tcciago.sj.ifsc.edu.br/privkey.pem
22 no-tls1
23 no-tls1_1
24 # Autenticação
25 lt-cred-mech
26 user=matrix:matrix
```

- na linha 13 tem a senha.
- linhas 19 e 20 o *range* de portas que podem ser usadas para o recebimento e envio de mídia.
- linha 25, nome do servidor, neste caso *server_name*.
- linhas 29 e 30, localização dos certificados a serem usados.

3.4.3 Instalação do Matrix

O Matrix não é tão simples como o TURN pois é preciso instalar outros pacotes como o servidor HTTP, porém também tem um pacote de instalação nos repositórios padrões do Ubuntu sendo possível instalar com o comando 3.4.

Código 3.4 – Comando de instalação do Matrix

```
27 apt install matrix-synapse-py3
```

Após a instalação do pacote, será criada uma pasta do aplicativo no */etc*, onde faremos algumas modificações de acordo com o nosso cenário, no arquivo */etc/matrix-synapse/homeserver.yaml*, onde a configuração ficou desta forma 3.5. A instalação teve duração de alguns dias devido a tentativas através de outros métodos como a partir de um pacote *Python* usando *pip*.

Código 3.5 – Conteúdo do arquivo de configuração do Matrix

```
28 server_name: "tcciago.sj.ifsc.edu.br"
29 pid_file: "/var/run/matrix-synapse.pid"
30 listeners:
31   - port: 8008
32     tls: false
33     type: http
34     x_forwarded: true
35     bind_addresses: [ '::1', '127.0.0.1' ]
36     resources:
37       - names: [ client, federation ]
38         compress: false
39 database:
40   name: sqlite3
41   args:
42     database: /var/lib/matrix-synapse/homeserver.db
43 log_config: "/etc/matrix-synapse/log.yaml"
44 enable_registration: true
45 enable_registration_without_verification: true
46 media_store_path: /var/lib/matrix-synapse/media
47 signing_key_path: "/etc/matrix-synapse/homeserver.signing.key"
48 trusted_key_servers:
49   - server_name: "matrix.org"
50
51 turn_uris: [ "turn:tcciago.sj.ifsc.edu.br?transport=udp", "turn:tcciago.sj.ifsc.edu.br?transport=tcp" ]
52 turn_user_lifetime: 86400000
53 turn_allow_guests: True
```

O arquivo é criado no instante da instalação e vem preenchido com valores padrão, foi atualizado o DNS criado para o trabalho, as portas utilizadas na linha, o endereço do servidor TURN na linha.

É necessário também a instalação de um servidor HTTP na máquina para que seja possível hospedar serviços e fazer comunicações usando este protocolo. Temos muitos servidores sem custo e que já tem seus aplicativos nos pacotes padrão do Ubuntu como Apache e Nginx. Foram feitas tentativas com os 2 e por questões de facilidade de configuração o Nginx foi o escolhido para ser o servidor da infraestrutura, o processo dura cerca de 1 hora.

Para instalar o servidor HTTP na máquina foi usado este comando [3.6](#)

Código 3.6 – Comando de instalação do Nginx

```
54 apt install nginx
```

Também serão criados os arquivos de configuração na pasta `/etc/nginx`, como o arquivo `/etc/nginx/conf.d/matrix.conf`. que tem seu conteúdo mostrado na [3.7](#).

Código 3.7 – Conteúdo do arquivo de configuração do Nginx

```
55 server {
56     listen 80;
57     listen [::]:80;
58     server_name tcciago.sj.ifsc.edu.br;
59     return 301 https://$host$request_uri;
60 }
61
62 server {
63     listen 443 ssl;
64     listen [::]:443 ssl;
65     server_name tcciago.sj.ifsc.edu.br;
66
67     ssl on;
68     ssl_certificate /etc/letsencrypt/live/tcciago.sj.ifsc.edu.br/fullchain.pem;
69     ssl_certificate_key /etc/letsencrypt/live/tcciago.sj.ifsc.edu.br/privkey.pem;
70
71     location / {
72         proxy_pass http://localhost:8008;
73         proxy_set_header X-Forwarded-For $remote_addr;
74     }
75 }
76
77 server {
78     listen 8448 ssl default_server;
79     listen [::]:8448 ssl default_server;
80     server_name tcciago.sj.ifsc.edu.br;
81
82     ssl on;
83     ssl_certificate /etc/letsencrypt/live/tcciago.sj.ifsc.edu.br/fullchain.pem;
84     ssl_certificate_key /etc/letsencrypt/live/tcciago.sj.ifsc.edu.br/privkey.pem;
85     location / {
86         proxy_pass http://localhost:8008;
87         proxy_set_header X-Forwarded-For $remote_addr;
88     }
89 }
```

As linhas importantes serão descritas abaixo.

- nas linhas 72, 73, 87 e 88, mostra as portas usadas pelo servidor, o tipo de criptografia do certificado.
- linhas 77, 78, 92 e 93, indica a localização dos certificados
- linhas 74 e 89, o DNS usado nas portas utilizadas.

Neste arquivo há a criação do serviço para cada uma das portas que será utilizada, que neste caso são 8448, usando HTTP, e 443, usando a porta padrão do HTTPS, indicado pela ação *listen*, ouvir, mais o nome do servido que é o endereço DNS criado para o projeto sendo referido na linha em *servername*.

Como a conexão WebRTC é feita por meios seguros, através do HTTPS, é preciso de um certificado para que a comunicação seja encriptada entre as partes envolvidas, portanto usamos o Let's Encrypt, um a organização que cria esses certificados sem custos, porém não são verificados pelos grandes autenticadoras e ao acessar o site irá mostrar este símbolo no seu browser.

Na linha é possível ver que usamos uma criptografia SSL, apesar de ser mais recomendado o TLS por ser mais atual e seguro, para este fim o SSL satisfaz os requisitos, nas linhas colocamos o caminho para a chave pública e privada que serão usadas na conexão entre os dispositivos e o servidor Matrix. Para gerar este certificado na máquina, mais uma vez podemos fazer isso através de um pacote padrão do Linux e já preparado para funcionar na infraestrutura do servidor HTTP, apenas executando este comando 3.8. Caso fosse usado o Apache também tem um pacote do certbot para este. Após o comando deve se executar este comando 3.9, será pedido apenas o e-mail, concordar com os Termos de Serviço e o domínio no que irá utilizar este certificado. O certbot já coloca dentro do Nginx a configuração do certificado criado.

Código 3.8 – Comando para instalar o gerador de certificados do *Let's Encrypt* no Ubuntu

```
90 apt-get install python3-certbot-nginx
```

Código 3.9 – Gerar de certificados do *Let's Encrypt* no Ubuntu

```
91 certbot --nginx
```

Foi preciso habilitar as portas 443 e 8080, que são as padrões para os respectivos protocolos HTTPS e HTTP, assim temos o serviço já estabelecido. Este processo levou aproximadamente 2 dias devido a quantidade de passos a se fazer até a certificação de funcionamento do Matrix através do navegador.

O Matrix sendo apenas um protocolo de comunicação, necessita de um cliente que será a interface que o usuário terá contato, neste ponto foi escolhido o cliente Element.

A única coisa que é preciso fazer é colocar o servidor criado, como o servidor a ser usado, e criar uma conta, para o projeto foram criadas 3 contas distintas para cada dispositivo sendo estes os nomes:

- iago_linux
- iago_windows

- iago_phone

O criador da sala sempre é o administrador, os convidados a entrar na sala recebem o cargo padrão que tem poder de apenas mandar mensagens, não podendo iniciar chamadas, com o usuário administrador atualizei os cargos dos usuários restantes para que os mesmos pudessem iniciar.

3.4.4 Ferramentas de medição

As ferramentas usadas para fazer a análise não têm custo financeiro e não precisam de licenças. Como o WebRTC já é nativo dos navegadores, os próprios já têm comandos internos para caso o usuário deseja ver com detalhamento a conexão. Nos navegadores baseados no Chromium temos o endereço abaixo que pode colocado no campo da URL.

```
chrome://webrtc-internals
```

No navegador Firefox o comando é

```
about:webrtc
```

Com este comando conseguimos ver as portas, codecs de áudio e vídeo utilizados pela conexões WebRTC ativas no navegador, Jitter, perda de pacotes, RTT, quantidade de dados mandados e recebidos, toda a negociação entre as partes que envolve a escolha de caminhos e *codecs*. A particularidade é que este endereço não funciona nos dispositivos móveis.

Os pacotes fora de ordem não podem ser obtida por esta ferramenta por ser uma ferramenta de análise da comunicação e não de tráfego de rede, restando assim a adição do uso do Wireshark, ferramenta bastante usada para este fim, que analisa todos os pacotes que são enviados e recebidos pela interface de rede do dispositivo. Para instalar no Ubuntu apenas precisa executar o comando [3.10](#)

Código 3.10 – Comando de instalação do *Wireshark* no Ubuntu

```
92 apt-get install wireshark
```

Para a instalação no Windows seguimos os passos mostrados abaixo:

- ir no site do *Wireshark*.
- em umas seções abaixo terá a opção de *Download* com o instalador do *Windows*.
- após o arquivo estar no computador, deve se ir a pasta de *Downloads*.
- clicar duas vezes no arquivo recém baixado.
- seguir as instruções padrões.

Para executar o *Wireshark* é preciso ter acesso de super usuário em qualquer sistema operacional, pois como a ferramenta lê todos os pacotes que passam pela placa de rede, é uma ação que o usuário padrão do sistema não tem acesso.

O *Wireshark* apenas funciona em computadores, no dispositivo móvel foi instalar um aplicativo chamado PCAPDroid, que está disponível na *Play Store*, loja de aplicativos do sistema *Android*.

Outra ótima ferramenta é o comando *tcpdump* que pode ser instalado a partir dos pacotes padrão do Ubuntu através do comando 3.11.

Código 3.11 – Comando de instalação do *tcpdump* no Ubuntu

```
93 apt-get install tcpdump
```

Esta ferramenta será usada no servidor, por ter apenas o acesso pelo terminal e não é possível ter o modo visual do *Wireshark* nesta máquina, é preciso uma ferramenta que dê seus resultados no próprio terminal. A usaremos pra analisar as mensagens entre o *Element* e o servidor.

4 Resultados e testes

Com o objetivo de apresentar o funcionamento do sistema desenvolvido e validá-lo serão apresentados nesse capítulo os testes e as análises dos resultados. O requisito básico de sucesso na comunicação tanto de áudio quanto de vídeo foi alcançado em todos os cenários testados.

O dispositivo iago_windows foi colocado numa VPN, que na mesma utiliza um IP de rede privada conforme [Rekhter et al. \(1996\)](#), como vemos na [Figura 17](#), respeitando a infraestrutura definida na [Figura 16](#).

Figura 17 – IP obtido na VPN utilizada pelo dispositivo iago_windows.

```
PS C:\Users\i.faria> ipconfig

Configuração de IP do Windows

Adaptador desconhecido ProtonVPN:

    Sufixo DNS específico de conexão. . . . . :
    Endereço IPv4. . . . . : 10.2.0.2
```

Fonte: Autoria própria.

Na negociação é possível comprovar o funcionamento do STUN quando este revela a um dos participantes, neste caso o iago_linux, seu IP externo como visto na [Figura 18](#), se mostrando correto ao fazer a mesma consulta em um site de descoberta de IP [Figura 19](#). Este IP externo será usado pelo ICE na tentativa de estabelecimento da comunicação, colocando este IP como destino para uso do outro dispositivo envolvido, ao invés do seu IP interno que não é usado na rede pública.

Figura 18 – Funcionamento do STUN.

```
199 10.473221582 3.124.186.214 192.168.2.105 STUN 140 Binding Success Response XOR-MAPPED-ADDRESS: 45.160.88.72:59014 user: 854101hm6vfnlt:DCF8
```

Fonte: Autoria própria.

A proposta do WebRTC em ter uma conexão segura é confirmada até no uso do SRTP em ambos os cenários como mostra a [Figura 20](#) ao final da figura onde mostra o campo Type: use strp, impossibilitando a leitura dos dados por dispositivos que não estiveram envolvidos na negociação da comunicação.

Figura 19 – IP externo do dispositivo Linux.

What is my IP .com.br

(vazio = seu IP)
 ex.: google.com.br ou 204.152.191.37

Endereço IP 45.160.88.72
Reverso Indisponível
Navegador Chrome
Plataforma Linux 

Fonte: Autoria própria.

Figura 20 – Requisição para uso do SRTP.

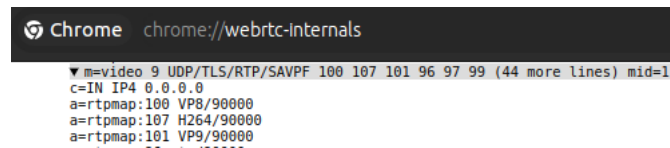
294	11.376425718	3.124.186.214	192.168.2.105	DTLSv1...	131	Server	Hello
Frame 294:	131 bytes on wire (1048 bits), 131 bytes captured (1048 bits) on interface						00
Ethernet II,	Src: zte_38:72:dc (b8:f0:b9:38:72:dc), Dst: Palladiu_72:2d:45 (5c:c9:d3)						00
Internet Protocol Version 4,	Src: 3.124.186.214, Dst: 192.168.2.105						00
User Datagram Protocol,	Src Port: 30300, Dst Port: 48520						00
Datagram Transport Layer Security							00
DTLSv1.2 Record Layer: Handshake Protocol:	Server Hello						00
Content Type:	Handshake (22)						00
Version:	DTLS 1.2 (0xfefd)						00
Epoch:	0						00
Sequence Number:	0						00
Length:	76						00
Handshake Protocol:	Server Hello						
Handshake Type:	Server Hello (2)						
Length:	64						
Message Sequence:	0						
Fragment Offset:	0						
Fragment Length:	64						
Version:	DTLS 1.2 (0xfefd)						
Random:	4a05481c2d482fb667c83842770e7f0f7e48987daca60f3aa42cfe0ea95981f						
Session ID Length:	0						
Cipher Suite:	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)						
Compression Method:	null (0)						
Extensions Length:	24						
Extension:	extended_master_secret (len=0)						
Extension:	renegotiation_info (len=1)						
Extension:	use_srtp (len=5)						
Type:	use_srtp (14)						
Length:	5						
SRTP Protection Profiles Length:	2						
SRTP Protection Profile:	SRTP_AEAD_AES_128_GCM (0x0007)						
MKI Length:	0						

Fonte: Autoria própria.

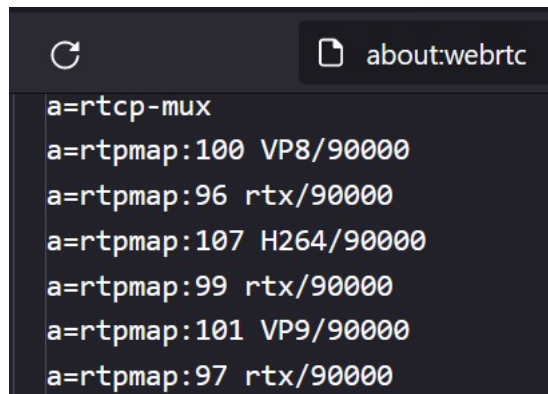
4.1 Codecs usados

Foi observado que todos os dispositivos, conforme [Valin e Bran \(2016\)](#), ofertam o *codec* Opus, apesar de serem oferecidos outros codecs de áudio.

O *codec* de vídeo utilizado foi o VP8 também em todas as ocasiões, apesar de ter nas ofertas outros codecs como H.264 e VP9 como visto em [Figura 21](#) e [Figura 22](#). As ofertas também seguem a regra de [Roach \(2016\)](#), que pede que as partes tenham ao menos os *codecs* VP8 e H.264.

Figura 21 – Oferta de *codecs* no dispositivo iago_linux, utilizando o navegador Chrome.

Fonte: Autoria própria.

Figura 22 – Oferta de *codecs* no dispositivo iago_windows, utilizando o navegador Firefox.

Fonte: Autoria própria.

4.1.1 Caminhos de mídia utilizados

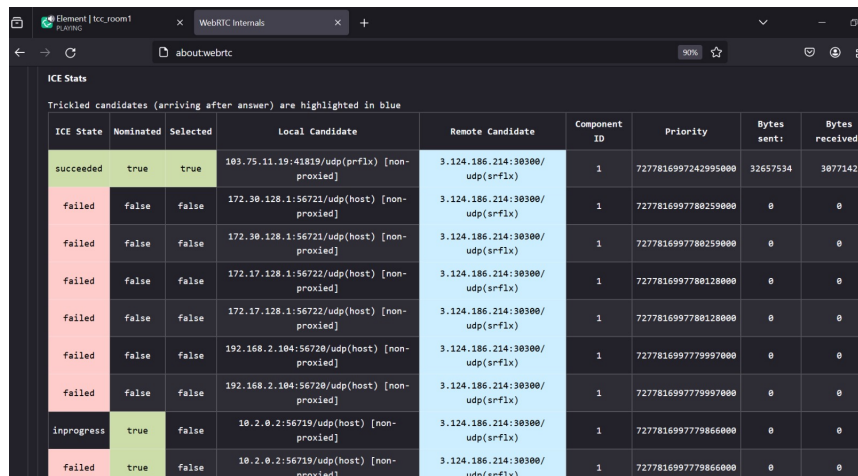
Devido ao uso do Element como cliente Matrix, este serviço já utiliza serviço de resolução de endereços NAT padrão com TURN e STUN, sendo prioritário ao configurado na máquina que utilizamos aqui. Podemos observar tais aplicações no decorrer da seção, como o STUN é mais recomendado, é o primeiro método a ser utilizado respondendo aos dispositivos seus respectivos IP externos.

Pelo caminho de mídia utilizado é suposto que o NAT onde os dispositivos se encontravam era do tipo *Symmetric*, ou seja, não passível de resolução através do método STUN, pois o tráfego da mídia passou inteiramente pelo servidor TURN indicado pelo Element ao invés do Matrix em todos os cenários, ficando para o servidor Matrix apenas o papel de gerenciamento da chamada. Na [Figura 23](#) mostra o funcionamento do método ICE, fazendo a negociação de estabelecimento da comunicação entre os dispositivos iago_windows e iago_linux, tentando achar um caminho para o transporte da mídia e falha em diversos caminhos propostos.

O método STUN falhou em todos os cenário e o TURN se tornou o método de resolução padrão, a negociação do estabelecimento da comunicação com o TURN foi analisada a partir do dispositivo iago_linux de IP [Figura 24](#), que realiza uma requisição de alocação de um endereço para o tráfego de mídia [Figura 25](#), com a [Figura 26](#) comprovando que este endereço IP se trata o servidor TURN utilizado pelo Element.

O servidor TURN oferece ao dispositivo um caminho alternativo para a mídia [Figura 27](#), sendo posteriormente utilizado no transporte da mídia como visto nas [Figura 28](#) e [Figura 29](#),

Figura 23 – Funcionamento do ICE.



ICE State	Nominated	Selected	Local Candidate	Remote Candidate	Component ID	Priority	Bytes sent	Bytes received
succeeded	true	true	193.75.11.19:41819/udp(prflx) [non-proxied]	3.124.186.214:30300/udp(srflx)	1	7277816997242995000	32657534	3077142
failed	false	false	172.30.128.1:56721/udp(host) [non-proxied]	3.124.186.214:30300/udp(srflx)	1	7277816997780259000	0	0
failed	false	false	172.30.128.1:56721/udp(host) [non-proxied]	3.124.186.214:30300/udp(srflx)	1	7277816997780259000	0	0
failed	false	false	172.17.128.1:56722/udp(host) [non-proxied]	3.124.186.214:30300/udp(srflx)	1	7277816997780128000	0	0
failed	false	false	172.17.128.1:56722/udp(host) [non-proxied]	3.124.186.214:30300/udp(srflx)	1	7277816997780128000	0	0
failed	false	false	192.168.2.104:56720/udp(host) [non-proxied]	3.124.186.214:30300/udp(srflx)	1	727781699779997000	0	0
failed	false	false	192.168.2.104:56720/udp(host) [non-proxied]	3.124.186.214:30300/udp(srflx)	1	727781699779997000	0	0
inprogress	true	false	10.2.0.2:56719/udp(host) [non-proxied]	3.124.186.214:30300/udp(srflx)	1	727781699779866000	0	0
failed	true	false	10.2.0.2:56719/udp(host) [non-proxied]	3.124.186.214:30300/udp(srflx)	1	727781699779866000	0	0

Fonte: Autoria própria.

Figura 24 – IP interno do dispositivo iago_linux.

```
iago@iago-pc ~-> ip -4 a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    inet 192.168.2.105/24 brd 192.168.2.255 scope global dynamic noprefixroute wlp2s0
        valid_lft 78419sec preferred_lft 78419sec
```

Fonte: Autoria própria.

Figura 25 – Requisição de alocação para o servidor TURN.

215	10.578599198	192.168.2.105	3.74.3.2	TLSv1.2	406 Client Hello
216	10.578950631	192.168.2.105	3.74.3.2	STUN	94 Allocate Request UDP

Fonte: Autoria própria.

Figura 26 – Servidor TURN do Element.

```
iago@iago-pc ~-> ping -4 turn.tools.element.io
PING turn.tools.element.io (3.74.3.2) 56(84) bytes of data.
```

Fonte: Autoria própria.

é possível notar nas colunas 3 ou 4 o IP alocado pelo servidor TURN, e nas colunas 7 ou 9, é mostrada a porta alocada neste servidor, sendo neste caso a 30300, sendo exatamente os mesmo que foram negociados na Figura 27, estes endereço e porta também são os mesmos ofertados para o outro dispositivo envolvido na comunicação.

Figura 27 – Endereço IP alocado pelo TURN.

```
600 20.2555233600 192.168.2.105 3.124.186.214 STUN 106 Binding Success Response XOR-MAPPED-ADDRESS: 3.124.186.214:30300
```

Fonte: Autoria própria.

O mesmo procedimento, oferta de um endereço alternativo pelo servidor TURN, acontece em todos os cenários com os dispositivos iago_windows e iago_phone. O dispositivo

iago_phone, conectado pela rede móvel, tem um IP de rede privada (REKHTER et al., 1996) como mostrada na Figura 30, este IP é o mesmo tanto para a rede 4G como para rede 5G, respeitando a infraestrutura definida na Figura 16.

Figura 28 – Média trocada entre o servidor e dispositivo iago_windows.

10801	30.662358	10.2.0.2	3.124.186.214	UDP	897	52970 → 30300	Len=869
10802	30.662464	10.2.0.2	3.124.186.214	UDP	1221	52970 → 30300	Len=1193
10803	30.662489	10.2.0.2	3.124.186.214	UDP	153	52970 → 30300	Len=125
10804	30.674805	10.2.0.2	3.124.186.214	UDP	985	52970 → 30300	Len=957
10805	30.682424	3.124.186.214	10.2.0.2	UDP	160	30300 → 52970	Len=132
10806	30.682802	3.124.186.214	10.2.0.2	UDP	887	30300 → 52970	Len=859
10807	30.684699	10.2.0.2	3.124.186.214	UDP	158	52970 → 30300	Len=130
10808	30.684797	10.2.0.2	3.124.186.214	UDP	985	52970 → 30300	Len=957
10809	30.684913	3.124.186.214	10.2.0.2	UDP	887	30300 → 52970	Len=859
10810	30.684913	3.124.186.214	10.2.0.2	UDP	888	30300 → 52970	Len=860

Fonte: Autoria própria.

Figura 29 – Média trocada entre o servidor e dispositivo iago_linux.

784	21.612867186	192.168.2.105	3.124.186.214	UDP	772	46635 → 30300	Len=730
785	21.616136359	192.168.2.105	3.124.186.214	UDP	156	46635 → 30300	Len=114
786	21.625491369	3.124.186.214	192.168.2.105	UDP	90	30300 → 46635	Len=48
787	21.625571968	3.124.186.214	192.168.2.105	UDP	90	30300 → 46635	Len=48
788	21.625585703	3.124.186.214	192.168.2.105	UDP	90	30300 → 46635	Len=48
789	21.625595244	3.124.186.214	192.168.2.105	UDP	90	30300 → 46635	Len=48
790	21.632127868	192.168.2.105	3.124.186.214	UDP	154	46635 → 30300	Len=112
791	21.645769251	192.168.2.105	3.124.186.214	UDP	520	46635 → 30300	Len=478
792	21.653980203	192.168.2.105	3.124.186.214	UDP	152	46635 → 30300	Len=110
793	21.676648602	192.168.2.105	3.124.186.214	UDP	149	46635 → 30300	Len=107
794	21.677332789	192.168.2.105	3.124.186.214	UDP	758	46635 → 30300	Len=716
795	21.701150697	192.168.2.105	3.124.186.214	UDP	159	46635 → 30300	Len=117
796	21.718435632	192.168.2.105	3.124.186.214	UDP	150	46635 → 30300	Len=108
797	21.727294888	3.124.186.214	192.168.2.105	UDP	90	30300 → 46635	Len=48
798	21.727295348	3.124.186.214	192.168.2.105	UDP	86	30300 → 46635	Len=44
799	21.727295476	3.124.186.214	192.168.2.105	UDP	90	30300 → 46635	Len=48
800	21.736994338	192.168.2.105	3.124.186.214	UDP	153	46635 → 30300	Len=111
801	21.744733440	192.168.2.105	3.124.186.214	UDP	528	46635 → 30300	Len=486
802	21.758190436	192.168.2.105	3.124.186.214	UDP	151	46635 → 30300	Len=109
803	21.777271749	192.168.2.105	3.124.186.214	UDP	858	46635 → 30300	Len=816

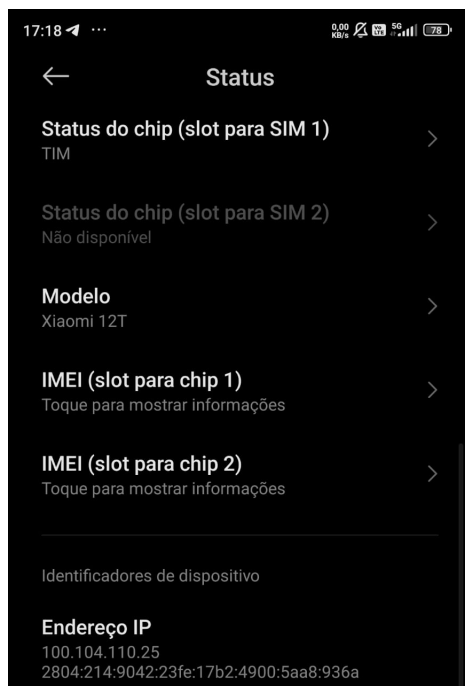
Fonte: Autoria própria.

4.1.2 Métricas de rede obtidas

Através dos comandos `chrome://webrtc-internals` e `about:webrtc`, nos seus respectivos navegadores, foi possível obter estas métricas de desempenho das conexões. As Figura 31 e Figura 32 mostram as métricas obtidas no navegador Chrome no dispositivo iago_linux. No dispositivo iago_windows foi usado o Firefox e as métricas são mostradas na Figura 33 e Figura 34.

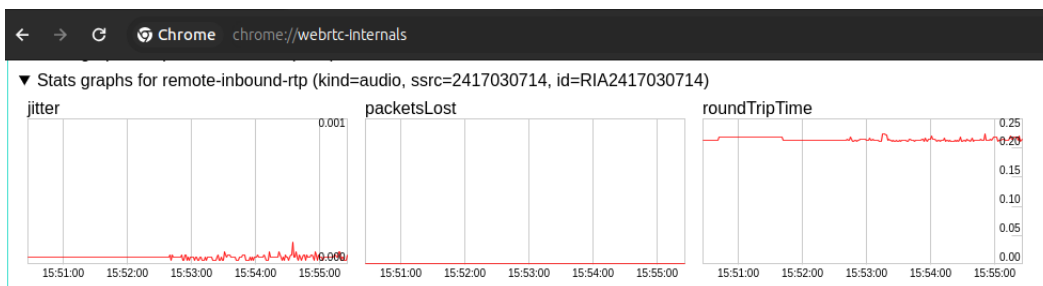
Nos testes com o dispositivo iago_phone, foi notado a possibilidade de escolha de qual rede preferencial a se conectar, podendo assim testar em 4G ou 5G, como visto na Figura 35. O dispositivo escolhido para fazer as chamadas com o celular(iago_phone), foi o Windows (iago_windows) por utilizar o navegador Firefox e este ter uma ferramenta melhor para análise da comunicação.

Figura 30 – IP do dispositivo iago_phone.



Fonte: Autoria própria.

Figura 31 – Métricas de rede da mídia de áudio no dispositivo iago_linux.



Fonte: Autoria própria.

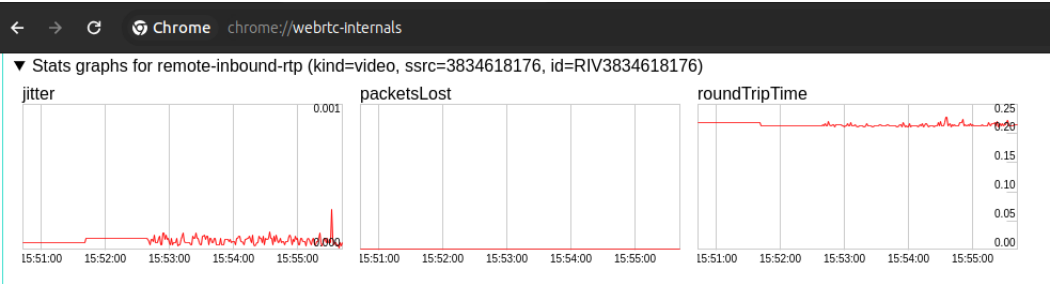
Tabela 1 – Média das métricas de rede dos cenários testados

Cenário	Jitter	RTT	Perda de pacotes
iago_linux e iago_windows	0.005 ms	690 ms	0%
iago_windows e iago_phone 4G	0.02 ms	695 ms	0%
iago_windows e iago_phone 5G	0.025 ms	693 ms	0%
iago_linux, iago_windows e iago_phone 5G	0.01 ms	704 ms	0%

A média das métricas em cada um dos cenários são mostradas em [Tabela 1](#).

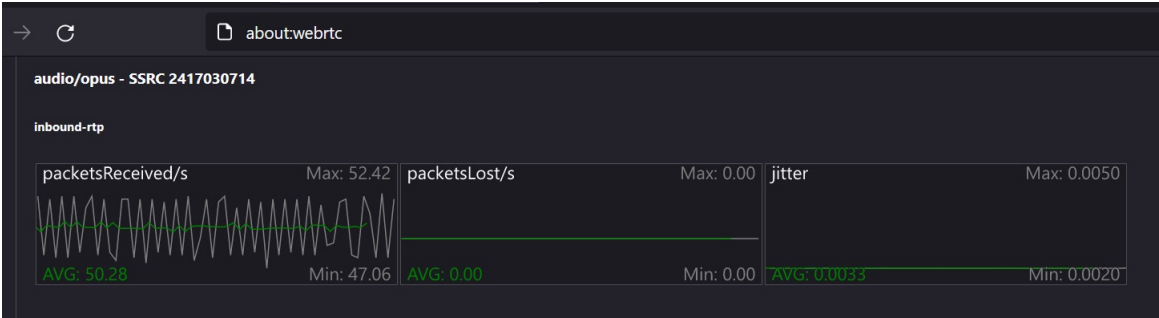
Cada chamada de teste teve duração média 5 minutos, todas foram feitas com áudio e vídeo, estas foram feitas em dias úteis no período da tarde ou noite. Em todos os testes de todos os cenários a perda de pacotes foi de 0 pacotes. Foram realizadas 10 chamadas em cada cenário, a partir destas obtidas uma média para cada uma das métricas de desempenho, os dados de cada uma delas se encontram em:

Figura 32 – Métricas de rede da mídia de vídeo no dispositivo iago_linux.



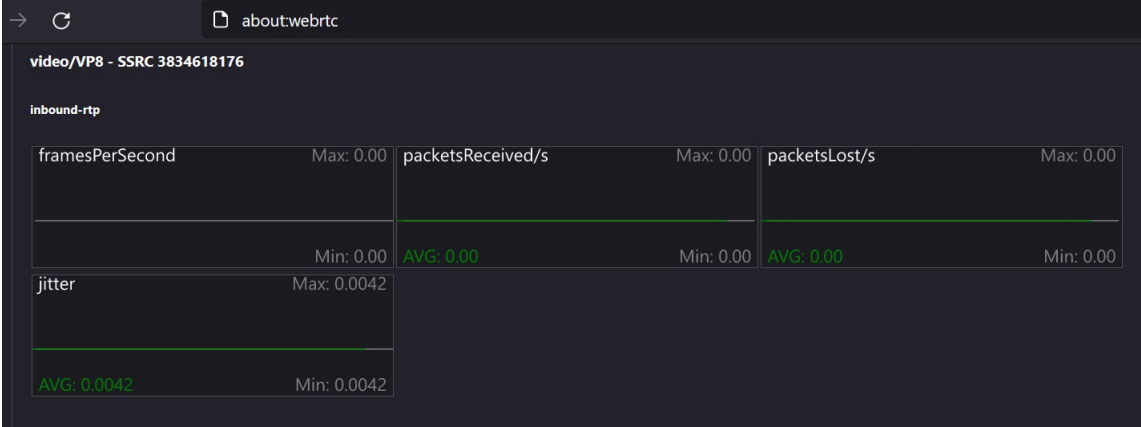
Fonte: Autoria própria.

Figura 33 – Métricas de rede da mídia de áudio no dispositivo iago_windows.



Fonte: Autoria própria.

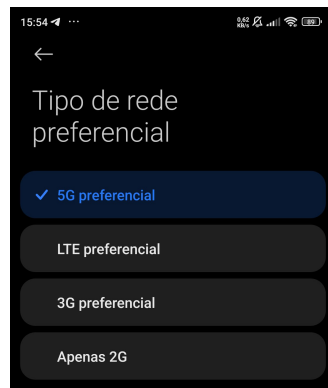
Figura 34 – Métricas de rede da mídia de vídeo no dispositivo iago_windows.



Fonte: Autoria própria.

- [Tabela 2](#) para as métricas do cenário entre 2 computadores, iago_windows e iago_linux.
- [Tabela 3](#) para as métricas do cenário entre 1 computador(iago_windows) e um dispositivo móvel conectado a rede 4G(iago_phone).
- [Tabela 4](#) para as métricas do cenário entre 1 computador(iago_windows) e um dispositivo móvel conectado a rede 5G(iago_phone).
- [Tabela 5](#) para as métricas do cenário entre 2 computadores(iago_windows e iago_linux) e 1 dispositivo móvel conectado a rede 5G(iago_phone).

Figura 35 – Escolha da rede a ser usada no iago_phone.



Fonte: Autoria própria.

Devido ao uso do SRTP, que é encriptado e a dificuldade de identificação do número de sequência de tantos pacotes UDP, não foi possível analisar com precisão a métrica dos pacotes fora de ordem, ficando pendente.

5 Conclusão

A motivação deste trabalho foi construir uma aplicação de comunicação que utiliza WebRTC, sendo o objetivo analisar o seu funcionamento. Os objetivos gerais do projeto que eram de estabelecimento desta aplicação, criação de uma infraestrutura de resolução de endereços NAT, realizar comunicação entre os dispositivos e analisar as características do seu funcionamento, foram alcançados com sucesso, sendo até adicionado um cenário com os 3 dispositivos, que era um requisito adicional. O único porém foi a não possibilidade de usar o servidor TURN criado para este fim, devido a uma das ferramentas encaminhar para o seu servidor TURN prioritário, perdendo assim uma pequena parte do controle análise da aplicação.

A existência de servidores TURN para a resolução de conflitos gerados por NAT foi fundamental, sem essa ferramenta não poderia haver a comunicação entre nenhuma das partes envolvidas, dado que todas elas estavam em redes privadas, se mostrando essencial sua presença em qualquer serviço de chamadas na Internet.

A partir dos dados obtidos no [Capítulo 4](#), as chamadas feitas com o dispositivo móvel tiveram valores de *jitter* bem variáveis, apesar de pequenos, tanto para 4G e 5G, inclusive as 2 redes tiveram resultados bastante semelhantes em todas as métricas analisadas, como perda de pacotes, *jitter* e RTT. As métricas analisadas em todos os cenários tiveram seus valores esperados.

Sobre o restante das métricas analisadas, o RTT teve seu valor dentro do esperado, mantendo valores próximos em todos os cenários. Referente a métrica da perda de pacotes, apesar de não ter sido possível de analisar outra métrica relacionada que era a porcentagem de pacotes fora de ordem, era perceptível que a comunicação não fluía muito bem em alguns momentos dando a entender que era causada pela perda de pacotes, porém o valor desta métrica se manteve em 0, valor este que era absoluto e não em proporção, por exemplo 0 a cada 1 mil pacotes, ou porcentagem, dado que cada ligação teve duração média de 5 minutos, o resultado obtido da perda de pacotes foi surpreendente. Isto mostra a capacidade de controle de *buffer* da aplicação, ao guardar os pacotes que chegam fora de ordem e os reorganiza para mostrar ao usuário.

Devido a problemas no estabelecimento do servidor não foi possível o uso de endereços IPv6.

A compatibilidade entre os dispositivos foi alcançada, não havendo problemas de conexão, *codecs* e outros, com os resultados vistos no [Capítulo 4](#), concluo que o WebRTC é uma ferramenta confiável, segura e compatível com diferentes dispositivos. Posso afirmar o mesmo para o Matrix, protocolo que pouco usado mas bastante promissor. Confirmando também a aplicação das regras definidas nas RFCs citadas em relação a *codecs* de áudio ([VALIN](#);

BRAN, 2016) e vídeo (ROACH, 2016) são cumpridas garantindo assim a compatibilidade entre quaisquer dispositivos que queiram se conectar utilizando WebRTC, mantendo um requisito mínimo comum entre os mesmos.

5.1 Trabalhos futuros

Recomenda-se para trabalhos futuros a implementação de novas funcionalidades no sistema, como por exemplo:

- Implementar o STUN em uma máquina própria, ao invés de usar STUN de outra instância com Google, podendo assim editar as configurações.
- Encontrar um método da aplicação forçar o caminho dos dados para o TURN configurado no servidor Matrix.
- Forçar a escolha de outros codecs, como VP9, H.264
- Configurar outros tipos NAT onde estão os dispositivos e analisar o comportamento da comunicação.
- Testar a comunicação WebRTC em um cenário com endereços apenas IPv6.
- Testar um cenário de gargalo, onde a quantidade de participantes, portanto uma grande quantidade de mídia sendo trocada pelos participantes seja alta, aumentando assim o *jitter* dos pacotes.
- Criar um mecanismo no qual seja possível calcular a quantidade de pacotes que chegam fora de ordem.
- Analisar o *delay* de reprodução da mídia dentro da própria aplicação ocasionado pelo *buffer* dos dados devido a perda de pacotes ou muitos pacotes fora de ordem.

Referências

- 5G System Overview. [S.I.]. <<https://www.3gpp.org/technologies/5g-system-overview>>. Disponível em: <<https://www.3gpp.org/technologies/5g-system-overview>>. Acesso em: 15 jan 2024. Citado na página 37.
- A evolução das Telecomunicações. [S.I.], 2016. <<https://www.worknets.com.br/2016/06/04/a-evolucao-das-telecomunicacoes/>>. Disponível em: <<https://www.worknets.com.br/2016/06/04/a-evolucao-das-telecomunicacoes/>>. Acesso em: 22 mar 2023. Citado na página 15.
- BAUGHER, M. et al. *The Secure Real-time Transport Protocol (SRTP)*. [S.I.], 2004. <<http://www.rfc-editor.org/rfc/rfc3711.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc3711.txt>>. Acesso em: 7 jun 2024. Citado na página 35.
- BORGES, F. S. *WebRTC: Estudo e Analise do Projeto*. São José, 2013. 57 p. Acesso em: 29 out 2024. Citado na página 20.
- CIC, T. M. F. *Matrix Specification*. 2023. Disponível em: <<https://spec.matrix.org/latest/>>. Acesso em: 15 fev 2024. 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. Acesso em: 3 set 2024. Citado na página 29.
- FETTE, I.; MELNIKOV, A. *The WebSocket Protocol*. [S.I.], 2011. <<http://www.rfc-editor.org/rfc/rfc6455.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc6455.txt>>. Acesso em: 9 jun 2024. Citado na página 21.
- 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 20, 21 e 22.
- HABERMAN, B.; HINDEN, B. *Unique Local IPv6 Unicast Addresses*. RFC Editor, 2005. RFC 4193. (Request for Comments, 4193). Disponível em: <<https://www.rfc-editor.org/info/rfc4193>>. Citado na página 30.
- HANDLEY, M.; JACOBSON, V.; PERKINS, C. *SDP: Session Description Protocol*. [S.I.], 2006. <<http://www.rfc-editor.org/rfc/rfc4566.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc4566.txt>>. Citado na página 27.
- HIGH-LEVEL architecture of Mobile Cellular Networks from 2G to 5G. [S.I.]. <<https://www.3g4g.co.uk/Training/intermediate0001.html>>. Disponível em: <<https://www.3g4g.co.uk/Training/intermediate0001.html>>. Citado na página 36.
- HOLMBERG, C.; HAKANSSON, S.; ERIKSSON, G. *Web Real-Time Communication Use Cases and Requirements*. [S.I.], 2015. <<http://www.rfc-editor.org/rfc/rfc7478.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc7478.txt>>. Citado na página 21.

- 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.
- MéSZÁROS, M. *WebRTC*. 2014. <<https://www.slideshare.net/rootkiskacs/webrtc-educonf-porto>>. Citado na página 32.
- 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 na página 24.
- PAGELS, S. *Will You Stay or Will You Go?* [S.l.], 2019. <<https://lup.lub.lu.se/luur/download?func=downloadFile&recordId=8998380&fileId=8998381>>. Disponível em: <<https://lup.lub.lu.se/luur/download?func=downloadFile&recordId=8998380&fileId=8998381>>. Citado na página 16.
- PROUST, S. *Additional WebRTC Audio Codecs for Interoperability*. [S.l.], 2016. Citado na página 22.
- REKHTER, Y. et al. *Address Allocation for Private Internets*. [S.l.], 1996. <<http://www.rfc-editor.org/rfc/rfc1918.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc1918.txt>>. Citado 3 vezes nas páginas 30, 49 e 53.
- ROACH, A. *WebRTC Video Processing and Codec Requirements*. RFC Editor, 2016. RFC 7742. (Request for Comments, 7742). Disponível em: <<https://www.rfc-editor.org/info/rfc7742>>. Citado 3 vezes nas páginas 23, 50 e 58.
- 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. *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 2 vezes nas páginas 33 e 34.
- SCHULZRINNE, H. et al. *RTP: A Transport Protocol for Real-Time Applications*. [S.l.], 2003. <<http://www.rfc-editor.org/rfc/rfc3550.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc3550.txt>>. Citado 2 vezes nas páginas 33 e 35.
- SOUTO, R. *WebSockets*. 2013. Acesso em 08/12/2017. Disponível em: <<https://prezi.com/hyv8u1sl6qjx/rtfmwebsockets/>>. Citado na página 28.
- SRISURESH, P.; HOLDREGE, M. *IP Network Address Translator (NAT) Terminology and Considerations*. [S.l.], 1999. <<http://www.rfc-editor.org/rfc/rfc2663.txt>>. Disponível em: <<http://www.rfc-editor.org/rfc/rfc2663.txt>>. Citado na página 29.
- STANDARDIZATION. [S.l.], 2023. <<https://webRTC.org/support/standardization?hl=pt-br>>. Disponível em: <<https://webRTC.org/support/standardization?hl=pt-br>>. Citado na página 19.

TELEGEOGRAPHY International Voice Report. [S.l.], 2022. Disponível em: <<https://cdn2.hubspot.net/hubfs/594292/assets/product-tear-sheets/product-page-content-samples/telegeography-report-and-database/telegeography-report-executive-summary.pdf>>. Citado na página 17.

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 21, 22, 28, 29 e 31.

VALIN, J.; VOS, K.; TERRIBERRY, T. *Definition of the Opus Audio Codec*. [S.l.], 2012. Citado na página 23.

VALIN, J.-M.; BRAN, C. *WebRTC Audio Codec and Processing Requirements*. RFC Editor, 2016. RFC 7874. (Request for Comments, 7874). Disponível em: <<https://www.rfc-editor.org/info/rfc7874>>. Citado 4 vezes nas páginas 22, 23, 50 e 58.

VARANDA, J. H. de O. *ICE: Uma solução geral para travessia de NAT*. Distrito Federal, 2008. 98 p. Citado na página 33.

W3C. *WebRTC: Real-Time Communication in Browsers*. [S.l.], 2023. <<https://w3c.github.io/webrtc-pc/>>. Disponível em: <<https://w3c.github.io/webrtc-pc/>>. Citado na página 19.

WebRTC. *Architecture*. 2018. <<https://webrtc.org/architecture/>>. Citado na página 20.

Apêndices

Tabela 2 – Média das medições do cenário entre 2 computadores obtidas em cada chamada

Número de tentativas	<i>Jitter</i>	RTT	Perda de pacotes
1	0.004 ms	687 ms	0%
2	0.005 ms	688 ms	0%
3	0.006 ms	685 ms	0%
4	0.004 ms	694 ms	0%
5	0.003 ms	690 ms	0%
6	0.005 ms	689 ms	0%
7	0.006 ms	686 ms	0%
8	0.008 ms	692 ms	0%
9	0.005 ms	695 ms	0%
10	0.004 ms	694 ms	0%

Tabela 3 – Média das medições do cenário entre 1 computador(iago_windows) e 1 dispositivo móvel conectado na rede 4G obtidas em cada chamada

Número de tentativas	<i>Jitter</i>	RTT	Perda de pacotes
1	0.025 ms	697 ms	0%
2	0.013 ms	699 ms	0%
3	0.020 ms	695 ms	0%
4	0.021 ms	691 ms	0%
5	0.017 ms	697 ms	0%
6	0.022 ms	693 ms	0%
7	0.021 ms	702 ms	0%
8	0.019 ms	696 ms	0%
9	0.024 ms	688 ms	0%
10	0.018 ms	692 ms	0%

Tabela 4 – Média das medições do cenário entre 1 computador(iago_windows) e 1 dispositivo móvel conectado na rede 5G obtidas em cada chamada

Número de tentativas	<i>Jitter</i>	RTT	Perda de pacotes
1	0.028 ms	700 ms	0%
2	0.030 ms	703 ms	0%
3	0.023 ms	685 ms	0%
4	0.022 ms	691 ms	0%
5	0.025 ms	701 ms	0%
6	0.024 ms	689 ms	0%
7	0.035 ms	687 ms	0%
8	0.018 ms	685 ms	0%
9	0.017 ms	695 ms	0%
10	0.028 ms	694 ms	0%

Tabela 5 – Média das medições do cenário entre 2 computadores e o dispositivo móvel conectado a rede 5G obtidas em cada chamada

Número de tentativas	<i>Jitter</i>	RTT	Perda de pacotes
1	0.012 ms	710 ms	0%
2	0.007 ms	703 ms	0%
3	0.011 ms	705 ms	0%
4	0.010 ms	708 ms	0%
5	0.008 ms	699 ms	0%
6	0.009 ms	700 ms	0%
7	0.015 ms	704 ms	0%
8	0.011 ms	708 ms	0%
9	0.008 ms	701 ms	0%
10	0.009 ms	702 ms	0%