INSTITUTO FEDERAL DE SANTA CATARINA

JEFFERSON BOTITANO CALDERON ROMERO

Desenvolvimento e Análise Comparativa de CODECs em Aplicações WebRTC Utilizando o Protocolo Matrix e SFU

São José - SC

dezembro/2023

LISTA DE ILUSTRAÇÕES

Figura 1 – Fluxo de exemplo de utilização API 10
Figura 2 – Arquitetura cliente-servidor 11
Figura 3 – Arquitetura peer-to-peer 12
Figura 4 – Arquitetura Web 15
Figura 5 – Arquitetura protocolo matrix P2P 24
Figura 6 – Arquitetura protocolo matrix com SFU 25

SUMÁRIO

1 INTRODUÇÃO 5
1.1 Objetivo geral 6
1.1.1 Objetivos específicos 6
1.2 Justificativa 6
1.2.1 Metodologia de trabalho 7

2 FUNDAMENTAÇÃO TEÓRICA 9
2.1 Aplicações em rede 9
2.1.1 Arquitetura de sistemas distribuídos 9
2.1.2 APIs: Facilitando a Inovação na Integração de Aplicações 10
2.1.3 Arquitetura cliente-servidor 11
2.1.4 Arquitetura ponto-a-ponto 12
2.2 Redes sem fio 13
2.3 Arquitetura Web 14
2.3.1 HTTP e HTTPS: Protocolos de Comunicação Web 15
2.3.2 WebSocket 17
2.3.3 WebRTC 18
2.3.4 Cifragem de Ponta a Ponta e Protocolos de Segurança 20
2.3.5 WebRTC Data Channels 22
2.4 Protocolo Matrix 23
2.4.1 NAT: Tradução de Endereços de Rede 25
2.4.2 STUN, TURN e ICE: Protocolos para Comunicação em Tempo Real 26
2.4.3 RTP: Protocolo de Transporte em Tempo Real 27
2.4.4 Codecs 28

3 PROPOSTA 31
3.1 Mapeamento de padrões e tecnologias 31
3.2 Arquitetura do sistema e seu desenvolvimento 31
3.3 Testes e validação 32
3.4 Cronograma de atividades 32

REFERÊNCIAS 35

1 INTRODUÇÃO

A comunicação em tempo real na Web tem se tornado cada vez mais relevante em nossa sociedade conectada digitalmente. O avanço das tecnologias e a demanda por interações imediatas têm impulsionado o desenvolvimento de soluções inovadoras, e uma delas é o *Web Real-Time Communication* (WebRTC).

Esta tecnologia desenvolvida pela Google e lançada como código aberto em maio de 2011, possibilita a comunicação de navegador para navegador sem a necessidade de *plugins* internos ou externos. (JAKOBSSON, 2015).

Enquanto o Protocolo de Controle de Transmissão (TCP) fornece os mecanismos para a troca de informações na internet, o *Web Real-Time Communication* (WebRTC) representa um avanço significativo na maneira como os dados são compartilhados e como as pessoas se comunicam online.

O WebRTC difere dos métodos tradicionais de comunicação em navegador, pois permite que a voz, vídeo e compartilhamento de arquivos ocorram diretamente entre os navegadores, sem a necessidade de passar por um servidor central, quando se refere a comunicação entre aplicações. Isso não apenas melhora a eficiência da comunicação, mas também reduz a latência e os gargalos de rede.

Quando se trata da expansão das aplicações e serviços multimídia na Internet, o WebRTC emerge como uma tecnologia central. Este *framework* é projetado para possibilitar comunicações em tempo real entre navegadores da web e dispositivos móveis, sendo notável sua utilização em plataformas como por exemplo o Google *Hangouts* e sua crescente adoção em sistemas de conferência com múltiplos participantes.

Resultado de uma colaboração entre duas organizações de padronização, o WebRTC utiliza tecnologias como JavaScript *Application Programming Interface*, em português Interface de programação de aplicativos (API) e marcador *Hypertext Markup Language version 5* (HTML5) para possibilitar conexões diretas entre dispositivos na web, permitindo que eles se comuniquem sem a necessidade de servidores intermediários.

A qualidade de aplicativos baseados em WebRTC torna-se uma preocupação crítica, impulsionando a necessidade de estratégias eficientes de avaliação de qualidade de experiência para essas aplicações.

Explorar o desempenho de diferentes *codecs* de áudio e vídeo dentro desta arquitetura juntamente com a crescente demanda por serviços de comunicação em tempo real, como evidenciado pelo aumento do tráfego de vídeo, a análise detalhada do desempenho dos *codecs* se torna algo útil para experiência do usuário final.

Esta investigação visa não apenas aprimorar a eficiência e qualidade das comunicações WebRTC, mas também entender como estas tecnologias podem ser otimizadas em diferentes cenários de uso, este estudo não apenas contribui para a compreensão técnica da comunicação WebRTC, mas também oferece *insights* práticos para profissionais, estudantes e entusiastas da tecnologia que buscam entender e explorar as complexidades e potencialidades desta inovadora plataforma de comunicação.

A implementação de uma aplicação WebRTC com o protocolo Matrix e uma *Selective Forwarding Unit* (SFU), aliada à análise do desempenho dos *codecs*, representa um passo significativo na direção de uma comunicação em tempo real mais eficiente e de melhor experiência para o usuário final.

1.1 Objetivo geral

Este trabalho tem como objetivo principal desenvolver uma aplicação WebRTC que utilize o protocolo Matrix e uma arquitetura baseada em *Selective Forwarding Unit* (SFU), e conduzir uma análise comparativa do desempenho de diversos *codecs* de áudio e vídeo para determinar a configuração mais eficiente e eficaz em termos de qualidade de mídia e uso de recursos.

1.1.1 Objetivos específicos

- Definir uma arquitetura WebRTC para realizar aos teste.
- Implementar a arquitetura de mídia para aplicação WebRTC.
- Definir como enviar dados de estatísticas de mídia para visualização gráfica.

1.2 Justificativa

O *Web Real-Time Communication* (WebRTC) consolidou-se como uma tecnologia essencial no contexto contemporâneo, adquirindo particular relevância durante a pandemia do COVID-19 devido ao seu papel fundamental no ensino e aprendizagem online.

Essencial para a comunicação em tempo real em diversos setores, incluindo educação, saúde e negócios, o WebRTC é também um instrumento vital na elaboração de soluções inovadoras para aprimorar a qualidade educacional e a eficiência empresarial.

Observa-se uma lacuna no conhecimento aprofundado do funcionamento e da importância do WebRTC, particularmente quanto ao desempenho dos *codecs* em sua integração com protocolos de sinalização.

Essa lacuna evidencia a necessidade de educar tanto usuários leigos quanto profissionais de desenvolvimento web acerca dos aspectos técnicos e práticos desta tecnologia. A presente pesquisa é impulsionada pela crescente demanda por profissionais qualificados em WebRTC, com o mercado de tecnologia da informação e comunicação em busca de especialistas aptos a implementar e resolver problemas complexos associados ao WebRTC em diversos setores.

Este estudo contribui para o entendimento técnico da integração de *codecs* com o protocolo Matrix em aplicações WebRTC e aborda a necessidade de comunicações seguras e eficientes em ambientes corporativos e educacionais.

A integração do WebRTC com múltiplos protocolos, exemplificada na aplicação prática de *chat* de vídeo, oferece amplas oportunidades para desenvolvedores. A flexibilidade do WebRTC, demonstrada pela contribuição de plataformas como a Kurento, sublinha seu potencial para impulsionar os serviços de comunicação em tempo real (DESHPANDE, 2015).

A importância científica deste trabalho reside na necessidade de investigar e ampliar o conhecimento sobre o WebRTC, com foco específico na análise de desempenho dos *codecs* em cenários de grande fluxo de mídia e o quanto impacta a *Quality of Service* (QoS).

1.2.1 Metodologia de trabalho

Neste trabalho, o foco é desenvolver e analisar uma aplicação WebRTC, incorporando o protocolo Matrix e uma *Selective Forwarding Unit* (SFU), para avaliar o desempenho de diversos *codecs*. A aplicação, destinada a ambientes de rede cabeada e sem fio, será testada com múltiplos usuários, mas exclusivamente na arquitetura que envolve o uso da *Selective Forwarding Unit* (SFU).

A metodologia começa com o desenvolvimento da aplicação WebRTC, onde o protocolo Matrix é integrado e a arquitetura SFU é configurada para suportar comunicações em tempo real com múltiplos usuários. Ambientes de teste para redes cabeada e sem fio serão estabelecidos para avaliar a aplicação sob diferentes condições de conectividade.

O estudo então se concentra na seleção e integração de vários *codecs* de áudio e vídeo na aplicação. Serão realizados testes para medir métricas como latência, qualidade de transmissão, uso de banda e estabilidade, fornecendo uma comparação detalhada do desempenho dos *codecs* na arquitetura com SFU.

Os dados obtidos desses testes serão analisados para identificar os *codecs* que oferecem o melhor desempenho em termos de eficiência e qualidade em diferentes condições de rede.

O objetivo é fornecer *insights* sobre a otimização de aplicações WebRTC em cenários que exigem comunicação eficiente e de alta qualidade em tempo real. Este trabalho contribuirá para o campo das comunicações em tempo real, oferecendo uma análise valiosa sobre a eficácia dos *codecs* na arquitetura WebRTC com SFU, essencial para o desenvolvimento de aplicações WebRTC mais robustas e eficientes.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo fornece uma base teórica fundamental para compreender os principais conceitos abordados neste trabalho. Esses conceitos estão organizados em quatro seções principais: Aplicações em rede, Redes sem fio, Arquitetura Web e Protocolo Matrix. A seguir, apresentamos características distintivas e possíveis aplicações de cada um desses tópicos, oferecendo uma visão abrangente dos elementos-chave que sustentam o estudo deste trabalho.

2.1 Aplicações em rede

Redes de computadores desempenham um papel fundamental na comunicação moderna, permitindo que os usuários acessem serviços por meio de várias aplicações de software. Cada aplicação é um programa aplicativo que se comunica através da rede com outros programas em computadores remotos.

Essas aplicações oferecem uma ampla gama de serviços e estão profundamente integradas em nossa vida cotidiana, servindo como a base de serviços como a Internet, a *World Wide Web* (W3), pesquisa na Web, jogos *online*, *e-mails*, redes sociais, *e-commerce*, entre outros (COULOURIS et al., 2013).

Embora cada aplicação possua sua própria interface específica, todas elas têm a capacidade de se comunicar entre si por meio de uma infraestrutura de rede compartilhada ou uma rede unificada. A existência de uma rede que suporta todas essas aplicações simplifica o trabalho do desenvolvedor de software, pois ele só precisa dominar a interface de rede e um conjunto básico de funções. Essas funções são utilizadas de forma consistente por todos os programas que se comunicam através da rede (COMER, 2016).

2.1.1 Arquitetura de sistemas distribuídos

A arquitetura de um sistema é essencialmente sua estrutura composta por componentes individuais especificamente definidos e suas inter-relações. As principais preocupações desta arquitetura é garantir a confiabilidade do sistema, sua capacidade de gerenciamento, adaptabilidade e, é claro, sua viabilidade econômica.

Em certo sentido, o processo de definição arquitetônica de um edifício compartilha similitudes com o desenvolvimento de sistemas, pois não apenas influencia sua aparência visual, mas também estabelece a estrutura geral e o estilo arquitetônico, fornecendo um padrão de referência sólido para sua construção (COMER, 2016).

A flexibilidade da ligação inter-redes é notável, pois não está limitada em termos de escala. Existem *internets* que abrangem apenas algumas redes, enquanto a Internet global engloba centenas de milhares de redes interconectadas. Além disso, o número de computadores conectados a cada rede individual dentro de uma *internet* pode variar amplamente, com algumas redes não possuindo computadores conectados, enquanto outras contam com centenas deles (COULOURIS et al., 2013).

2.1.2 APIs: Facilitando a Inovação na Integração de Aplicações

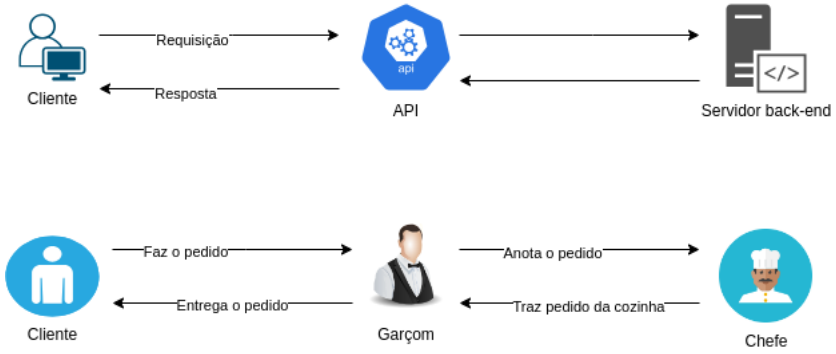
A *Application Programming Interface*, em português Interface de programação de aplicativos (API), é um conjunto de rotinas e padrões de programação projetado para facilitar o acesso a aplicativos de *software* e plataformas baseadas na Web. Ela engloba várias ferramentas, como métodos de desenvolvimento e protocolos, simplificando a comunicação entre programas e aplicações.

Em sua essência, a API possibilita a comunicação entre diferentes aplicações, programas e plataformas, eliminando a necessidade de intervenção direta do usuário. Isso permite que funcionalidades de um site ou serviço sejam disponibilizadas de forma eficiente e integrada em outras aplicações.

A seguir a Figura 1 apresenta dois tipos de fluxos, o superior mostra a interação entre um cliente e um servidor *Back-end* através de uma API. O cliente envia uma "Requisição"para a API, que por sua vez interage com o servidor *Back-end*. O servidor processa essa requisição e retorna uma "Resposta"para o cliente através da API.

O fluxo inferior da Figura 1 representa a interação em um restaurante, usando terminologia que espelha o fluxograma de Tecnologia da informação (TI) acima. Aqui, o “Client” faz um pedido, que é anotado pelo “Garçom”. O garçom então leva o pedido para a cozinha, onde o “Chefe” (chefe de cozinha) prepara o pedido. Após o pedido ser preparado, o garçom "Entrega o pedido"ao cliente.

Figura 1 – Fluxo de exemplo de utilização API



Fonte: Próprio Autor

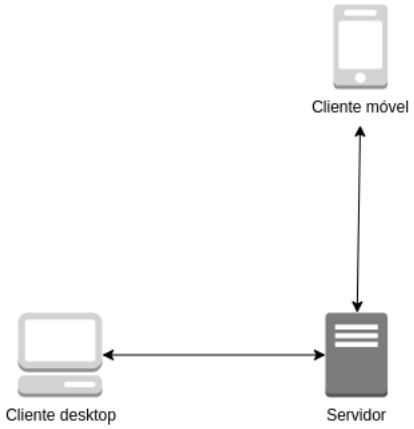
A principal vantagem de API reside na sua capacidade de permitir que soluções e

serviços se comuniquem com outros produtos e serviços sem a necessidade de conhecer detalhes sobre como eles foram implementados. Isso simplifica o processo de desenvolvimento de aplicações, economizando tempo e recursos financeiros. Ao criar novas ferramentas e soluções, ou ao gerenciar aquelas já existentes, as API oferecem a flexibilidade necessária para simplificar o *design*, administração e uso, além de proporcionar oportunidades de inovação (FERREIRA, 2021).

2.1.3 Arquitetura cliente-servidor

A principal ideia por trás do modelo cliente-servidor é permitir o acesso a recursos compartilhados disponibilizados pelo sistema. No entanto, uma característica notável desse modelo é a tendência à centralização de recursos, uma vez que as solicitações dos clientes são concentradas nos servidores, a Figura 2 ilustra esse tipo de arquitetura.

Figura 2 – Arquitetura cliente-servidor



Fonte: Próprio Autor

Ao considerar o desempenho de uma comunicação cliente-servidor, é evidente a importância de analisar o tempo necessário para a transmissão de mensagens de requisição e suas respostas em um contexto de rede local com baixa carga, incluindo as atividades de processamento. Nesse cenário, a latência, ou seja, o atraso na comunicação, é de aproximadamente de milissegundos na maior parte dos casos, o que pode parecer eficiente em muitos casos.

No entanto, ao compararmos esse valor com o tempo necessário para uma aplicação invocar uma operação em um objeto armazenado na memória local, que é inferior a um microssegundo, percebemos que o acesso a recursos compartilhados em uma rede local é aproximadamente mil vezes mais lento do que o acesso a recursos armazenados na memória local (COULOURIS et al., 2013). Essa diferença de desempenho destaca a necessidade de otimização e consideração cuidadosa ao projetar sistemas cliente-servidor. Apesar dos avanços no desempenho das redes, a latência ainda é um fator crítico a ser considerado.

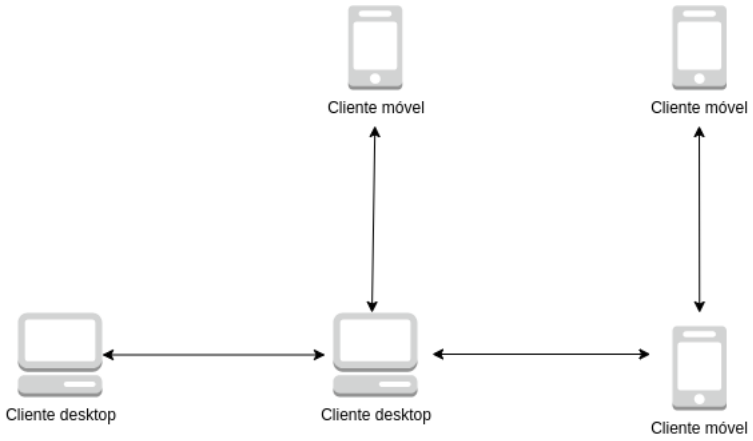
Por outro lado, é importante observar que a latência e a largura de banda da rede muitas vezes superam o desempenho dos discos rígidos. Isso significa que, em certos casos, o acesso a um servidor Web ou a um servidor de arquivos local que mantém em cache uma grande quantidade de arquivos frequentemente utilizados pode ser tão rápido ou até mais rápido do que o acesso a arquivos armazenados em um disco rígido local.

Nesse contexto, é fundamental compreender como a arquitetura cliente-servidor pode ser otimizada para lidar com questões de desempenho e latência, especialmente em ambientes de rede. A próxima seção abordará a arquitetura cliente-servidor com um foco específico em sua aplicação em ambientes Web, explorando as estratégias e desafios relacionados a esse modelo(COULOURIS et al., 2013).

2.1.4 Arquitetura ponto-a-ponto

A arquitetura Ponto-a-ponto (P2P) tem como característica central desse modelo a descentralização dos recursos, em que aplicativos são compostos por diversos processos, chamados de pares , geralmente executados em computadores distintos. Uma aplicação típica desse modelo é o compartilhamento de áudio, vídeo e programas (MONTEIRO et al., 2020), a Figura 3 faz a ilustração desse tipo de arquitetura.

Figura 3 – Arquitetura peer-to-peer



Fonte: Próprio Autor

Ao observar a arquitetura P2P, notamos uma diferença fundamental em relação ao modelo cliente/servidor: a ausência de uma distinção clara entre processos clientes e servidores. Nesse contexto, todos os processos interagem como pares, e cada processo envolvido em uma atividade ou tarefa desempenha funções semelhantes. Isso significa que um mesmo processo pode atuar tanto como servidor quanto como cliente. Além disso, a arquitetura P2P opera por meio da replicação de objetos entre os nós, otimizando a distribuição de carga em comparação ao modelo cliente-servidor. Essa arquitetura também se destaca por sua resiliência, uma vez que a falha de um nó tende a não afetar o sistema como um todo, proporcionando maior disponibilidade (MONTEIRO et al., 2020).

Nesse modelo, todos os processos envolvidos desempenham funções semelhantes, interagindo como pares, sem distinção entre processos clientes e servidores. A escalabilidade é um desafio, pois a distribuição de recursos compartilhados deve ocorrer em grande escala.

Embora esta arquitetura ofereça maior disponibilidade de recursos à medida que mais usuários participam, ela introduz complexidades significativas de implementação devido à necessidade de replicar objetos, distribuir cargas de armazenamento, processamento e comunicação entre muitos computadores e garantir a recuperação em caso de desconexões. Portanto, o desempenho em arquiteturas P2P é influenciado pela capacidade de distribuir eficientemente recursos compartilhados em uma rede descentralizada (COULOURIS et al., 2013).

2.2 Redes sem fio

Redes sem fio referem-se a qualquer rede não conectada por cabos, proporcionando conveniência e mobilidade ao usuário. Elas são centrais na tendência de conectividade ubíqua, possibilitando acesso a serviços online em qualquer local e momento (GRIGORIK, 2013). A Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) estabeleceu diversos padrões para *Wireless Fidelity* (Wi-fi), cada um diferenciado por características técnicas específicas.

O padrão original 802.11 opera na frequência de 2,4 GHz com taxas de transmissão de 1 ou 2 Mb/s e usa as técnicas de modulação *Direct Sequence Spread Spectrum* (DSSS) e *Frequencyhopping spread spectrum* (FHSS). O padrão 802.11b, também em 2,4 GHz, oferece 5,5 a 11 Mb/s com DSSS. Já o 802.11g, na mesma frequência, aumenta a taxa para 22 a 54 Mb/s, aplicando OFDM e DSSS. Por fim, o 802.11n opera em 2,4 GHz com taxas de 54 a 600 Mb/s, utilizando Multiplexação por divisão de frequências ortogonal (OFDM). Estes padrões refletem a evolução na capacidade de transmissão de dados e eficiência da modulação ao longo do tempo (COMER, 2016).

As redes de dados celulares Terceira geração (3G) conectam dispositivos móveis à Internet através de uma infraestrutura que opera paralelamente à rede de voz Sistema Global para Comunicações Móveis (GSM) existente, sem alterá-la. O core da rede 3G é composto por Servidores de Nó de Suporte *General Packet Radio Service* (GPRS) que gerenciam a transmissão de dados dos dispositivos móveis e Roteadores de borda de suporte GPRS (*Gateway GPRS Support Node* (GGSN)) que atuam como *gateways* para a *Internet*.

A rede de acesso por rádio 3G, controlada pelo *Radio Network Controller* (RNC), utiliza a tecnologia Acesso Múltiplo por Divisão de Código de Sequência Direta (WS-CDMA) dentro de intervalos Acesso Múltiplo por Divisão de Tempo (TDMA). Este sis-

tema suporta o serviço de Acesso a Pacote em Alta Velocidade (HSPA), com velocidades de até 14 Mbit/s (KUROSE; ROSS, 2013).

A Quarta geração (4G) , iniciada em 2008, é centrada em fornecer suporte para conteúdo multimídia em tempo real, como transmissões de televisão e downloads de vídeos de alta velocidade. Os dispositivos 4G são equipados com várias tecnologias de conectividade, incluindo Wi-fi e satélite, selecionando automaticamente a melhor opção disponível.

A transição do 3G para o 4G representa uma mudança fundamental na tecnologia subjacente: enquanto o 3G foi desenvolvido com base em padrões de telefonia de voz herdados, o 4G é projetado com o protocolo *Internet Protocol* (IP) como alicerce para todas as comunicações, empregando a comutação de pacotes, tornando a transmissão de voz uma aplicação dentro do conjunto de serviços oferecidos pela tecnologia IP (COMER, 2016).

A tecnologia Quinta geração (5G) representa uma evolução significativa em termos de conectividade, oferecendo uma rede altamente flexível e escalável, capaz de conectar todos e tudo, em qualquer lugar. Com uma arquitetura resiliente baseada na nuvem, o 5G suporta o fatiamento de rede de ponta a ponta, permitindo a criação de novos valores através de serviços inovadores em três principais domínios de uso: Banda larga móvel aprimorada (emBB), Comunicações ultra-confiáveis e de baixa latência (URLLC) e Comunicações massivas do tipo máquina (mMTC) (GHOSH et al., 2019).

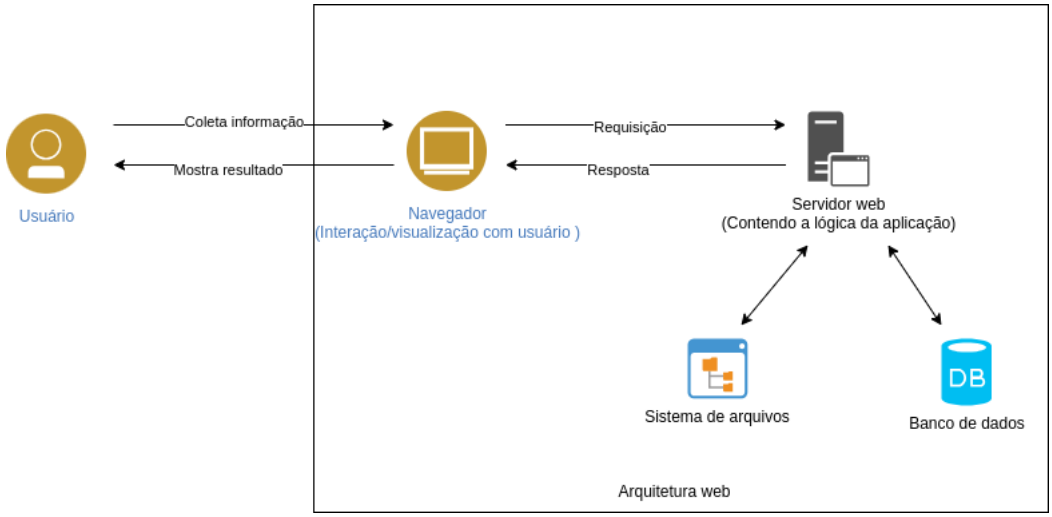
A crescente prevalência das redes sem fio impacta diretamente as aplicações WebRTC, que exigem conexões estáveis para a comunicação de áudio e vídeo em tempo real. Essa realidade demanda a otimização dessas aplicações para lidar com a variabilidade da largura de banda, latência e outras características inerentes às redes sem fio. Portanto, é essencial implementar soluções como adaptação de *bitrate* e estratégias para redução de latência, visando garantir uma experiência de usuário consistente e eficiente em aplicações WebRTC.

2.3 Arquitetura Web

No período anterior à década de 1990, a utilização da Internet estava majoritariamente circunscrita aos meios acadêmicos e de pesquisa, tendo como principais funcionalidades a transferência de arquivos, acesso remoto e comunicações eletrônicas, como correio eletrônico. Além disso, a facilidade de publicação na Web democratizou a distribuição de informações, permitindo que qualquer pessoa se tornasse um editor. Outras características, como *hiperlinks*, buscadores e recursos gráficos, aprimoraram a experiência do usuário, enquanto a evolução tecnológica resultou na criação de plataformas bem-sucedidas, como YouTube, Gmail e Facebook (KUROSE; ROSS, 2013).

Os *Web servers* atuam como facilitadores para o acesso via HTTP a um *"site"*, que é conceitualmente análogo a uma estrutura de árvore de documentos e informações, assemelhando-se a um sistema de arquivos de um computador. Enquanto os servidores Web inicialmente forneciam acesso a documentos estáticos, sua evolução permitiu a implementação de protocolos que direcionam solicitações a *softwares* específicos, possibilitando acesso a conteúdo dinâmico. Esse conteúdo pode originar-se de diversas fontes, como ferramentas de busca, bancos de dados, instrumentos de medição, entre outros (SHKLAR; ROSEN, 2003). A Figura 4 segue uma ilustração desse tipo de arquitetura.

Figura 4 – Arquitetura Web



Fonte: Próprio Autor

2.3.1 HTTP e HTTPS: Protocolos de Comunicação Web

O *Hypertext Transfer Protocol* (HTTP) opera no nível de aplicação no conjunto de protocolos Protocolo de Controle de Transmissão (TCP), utilizando o TCP como o protocolo de camada de transporte subjacente para transmitir mensagens (GRIGORIK, 2013). Pontos-chave para entender sobre o protocolo HTTP e a estrutura de mensagens do mesmo são:

- O HTTP opera em um modelo de requisição/resposta, onde um cliente HTTP envia uma mensagem de requisição para um servidor HTTP, e o servidor responde com uma mensagem de resposta HTTP. Alguns exemplos de acordo com a *Request for Comments* (RFC)

- A estrutura das mensagens de requisição e resposta HTTP se assemelha à de mensagens de e-mail, consistindo em cabeçalhos seguidos por uma linha em branco e um corpo de mensagem.

- O HTTP é um protocolo sem estado, o que significa que ele não gerencia ou mantém estados de sessão. Cada transação HTTP consiste em uma única requisição do cliente e uma única resposta do servidor.

O protocolo *Hypertext Transfer Protocol Secure* (HTTPS) é uma extensão do HTTP projetada para fornecer segurança na comunicação pela internet. Ele é utilizado quando há a necessidade de proteger a privacidade e integridade dos dados transmitidos. No HTTPS, as informações são criptografadas usando o protocolo *Transport Layer Security* (TLS), garantindo que apenas o destinatário correto possa decifrá-las.

Isso torna a comunicação mais segura, especialmente quando se trata de dados sensíveis, como informações pessoais e financeiras em transações online. O HTTPS é essencial para proteger os usuários contra ameaças e ataques, garantindo uma experiência segura na web (COULOURIS et al., 2013).

O HTTP/1.1 e o HTTP/2 são protocolos de aplicação que operam sobre o TCP e desempenham um papel fundamental na comunicação entre clientes e servidores na W3. A primeira padronização formal do HTTP foi o HTTP/1.1, lançado em janeiro de 1997 como RFC 2086 (MYERS, 1997). No entanto, o HTTP/1.1 apresentava algumas limitações, como a troca sequencial de requisições e respostas, que resultava em bloqueios de linha de cabeçalho no nível da aplicação, prejudicando os tempos de carregamento de páginas da Web.

Em meados de 2009, a empresa Google anunciou um novo protocolo experimental chamado *Speedy* (SPDY), projetado para resolver alguns dos problemas de desempenho conhecidos do HTTP/1.1 e melhorar a latência na entrega da Web. O HTTP/2 superou o bloqueio de linha de cabeçalho no nível da aplicação por meio da multiplexação dinâmica de requisições em uma única conexão TCP.

Além dos protocolos padronizados, o Google também desenvolveu um protocolo experimental chamado Conexões Rápidas de Internet via UDP (QUIC), baseado no *User Datagram Protocol* (UDP). Ele oferece várias novas funcionalidades, como estabelecimento de conexão mais rápido, migração de conexão, controle de congestionamento flexível e suporte a mecanismos criptográficos por *design* (ZINNER et al., 2017). Tanto o HTTP/2 quanto o QUIC aproveitam a multiplexação de fluxos sobre uma única conexão, permitindo que o navegador solicite vários arquivos simultaneamente sem a necessidade de conexões TCP adicionais. Isso ajuda a reduzir a latência da Web, pois reduz a necessidade de atrasos de Tempo de ida e volta (RRT).

Além disso, o HTTP/2 introduziu uma camada de estrutura binária para aproveitar melhor a conexão TCP, possibilitando a multiplexação de vários fluxos sobre o mesmo túnel TCP. O QUIC, apesar de ser construído sobre o UDP, também utiliza a multiplexação de fluxos por meio de uma única conexão, semelhante ao HTTP/2.

A evolução dos protocolos da Web trouxe o surgimento do HTTP/3. Diferentemente de suas versões anteriores que operavam sobre o TCP, o HTTP/3 é fundamentado no protocolo QUIC, que opera sobre o UDP. O QUIC proporciona uma série de melhorias em relação ao TCP, como o estabelecimento de conexão acelerado, redução da latência por meio da capacidade de estabelecimento de conexão de zero e a possibilidade de múltiplos fluxos ativos em uma única conexão. Ademais, o QUIC oferece uma criptografia de ponta a ponta, o que incrementa a segurança na comunicação Web. A integração desse protocolo ao HTTP/3 conferiu ao novo padrão melhorias significativas em termos de segurança e eficiência, tornando-o apto para atender às crescentes demandas da internet moderna(GUPTA; BARTOS, 2022).

2.3.2 WebSocket

O *WebSocket* é uma tecnologia que possibilita a transmissão bidirecional de dados, tanto em formato de texto quanto binário, de maneira orientada a mensagens entre o cliente e o servidor,ele acabando sendo um dos métodos de transporte mais versáteis e flexíveis disponíveis no navegador. Sua API simples permite a criação de protocolos de aplicação personalizados entre o cliente e o servidor.

Isso significa que é possível transmitir diversos tipos de dados, desde cargas *JavaScript Object Notation* (JSON) simples até formatos de mensagens binárias personalizados, de maneira contínua, onde ambos os lados podem enviar dados a qualquer momento (GRIGORIK, 2013). Esta tecnologia é uma API mais próxima de um *socket* de rede bruto no navegador e sua conexão vai além de um simples *socket* de rede, uma vez que o navegador abstrai toda a complexidade por trás de uma API simples e fornece diversos serviços adicionais:

- Negociação de conexão e aplicação da política de mesma origem.
- Interoperabilidade com a infraestrutura HTTP existente.
- Comunicação orientada a mensagens e eficiente estruturação de mensagens.
- Negociação de sub-protocolos e extensibilidade.

O início da conexão WebSocket se dá como uma simples solicitação HTTP, onde navegadores e clientes que suportam WebSocket enviam um pedido ao servidor com cabeçalhos específicos solicitando uma *"Connection: Upgrade"* para WebSocket.

Essa capacidade de "upgrade"foi introduzida no HTTP/1.1. Conforme especificado pelo WebSocket, a confirmação de que uma conexão foi estabelecida é dada pelo

campo de cabeçalho *"Sec-WebSocket-Accept"*, cujo valor é uma *hash* de um *Globally Unique Identifier* (GUID) predefinido e do cabeçalho HTTP do cliente *"Sec-WebSocket-Key"* (LOMBARDI, 2015).

A requisição Código 2.1 é apresentado um pedido HTTP inicial para estabelecer uma conexão WebSocket com o servidor local na porta 8181. Utilizando o protocolo HTTP/1.1, este pedido especifica, através do cabeçalho *"Upgrade: websocket"*, a intenção de mudar do protocolo HTTP para WebSocket. O cabeçalho *"Connection: Upgrade"* reforça essa intenção de alterar o protocolo.

O *"Sec-WebSocket-Key"* é um token base64 que o servidor codifica e retorna para confirmar a negociação. Por fim, *"Sec-WebSocket-Version: 13"* indica a versão do protocolo WebSocket que o cliente pretende usar.

Código 2.1 – Exemplo de HTTP GET via websocket

```
1 GET ws://localhost:8181/ HTTP/1.1
2 Origin: http://localhost:8181
3 Host: localhost:8181
4 Sec-WebSocket-Key: zy6Dy9mSAIM7GJZNf9rI1A==
5 Upgrade: websocket
6 Connection: Upgrade
7 Sec-WebSocket-Version: 13
```

A resposta apresentada em Código 2.2 é um indicativo do servidor de que aceitou o pedido para mudar do protocolo HTTP para o WebSocket. Utilizando o protocolo HTTP/1.1, o status *"101 Switching Protocols"* confirma a intenção do servidor de fazer a troca. O cabeçalho *"Connection: Upgrade"* e *"Upgrade: websocket"* reafirmam essa transição. O *"Sec-WebSocket-Accept"* é um *token* que o servidor gera com base no *token* *"Sec-WebSocket-Key"* enviado pelo cliente, validando assim a abertura bem-sucedida da conexão WebSocket.

Código 2.2 – Exemplo de Resposta HTTP via websocket

```
1 HTTP/1.1 101 Switching Protocols
2 Connection: Upgrade
3 Sec-WebSocket-Accept: EDJa7WCAQQzMCYNJM42Syuo9SqQ=
4 Upgrade: websocket
```

2.3.3 WebRTC

WebRTC é um conjunto de padrões, protocolos e APIs JavaScript que possibilitam a comunicação de áudio, vídeo e dados em tempo real entre navegadores sem a necessidade de *plugins* de terceiros ou software proprietário. Com o WebRTC, a comunicação em tempo real se torna um recurso padrão acessível através de uma simples API JavaScript.

Para possibilitar aplicações ricas e de alta qualidade, como teleconferência de áudio e vídeo e troca de dados ponto a ponto, os navegadores requerem novas funcionalidades, incluindo capacidades de processamento de áudio e vídeo, novas APIs de aplicação e suporte para diversos novos protocolos de rede.

A maioria desta complexidade é abstraída pelos navegadores através de três APIs principais: *MediaStream*, para aquisição de fluxos de áudio e vídeo; *RTCPeerConnection*, para comunicação de dados de áudio e vídeo; e *RTCDataChannel*, para comunicação de dados arbitrários de aplicação (GRIGORIK, 2013).

A arquitetura e os protocolos subjacentes ao WebRTC são determinantes para suas características de desempenho, como latência de configuração de conexão, sobrecarga de protocolo e semânticas de entrega. Distintamente de outras comunicações no navegador, o WebRTC transmite seus dados via UDP, embora muito mais seja necessário além do UDP bruto para viabilizar a comunicação em tempo real no navegador.

WebRTC rompe com o tradicional modelo de comunicação cliente-servidor, exigindo uma reengenharia completa da camada de rede no navegador, além de introduzir uma nova pilha de mídia necessária para o processamento eficiente em tempo real de áudio e vídeo. Em consequência, a estrutura do WebRTC incorpora uma ampla gama de padrões, abrangendo tanto as APIs de navegadores e aplicativos quanto uma variedade de protocolos e formatos de dados essenciais para seu funcionamento. O WebRTC não apenas facilita a interação em tempo real entre navegadores, mas também foi criado para se harmonizar com sistemas de comunicação pré-existentes, ampliando as funcionalidades da Web para adentrar no setor de telecomunicações, uma área de notável relevância econômica (GRIGORIK, 2013).

O *Session Description Protocol* (SDP) é um componente fundamental no estabelecimento de conexões WebRTC, funcionando como um formato para descrever os parâmetros de uma conexão ponto a ponto. SDP é empregado para detalhar o "perfil da sessão", que inclui as propriedades da conexão, como os tipos de mídia a serem trocados (áudio, vídeo e dados de aplicação), transportes de rede, *codecs* utilizados e suas configurações, informações de largura de banda, entre outras metadados.

O SDP em si não transmite mídia, seu papel é meramente descritivo. Em um fluxo de trabalho típico de WebRTC, uma vez que uma transmissão local é registrada com o objeto *RTCPeerConnection*, a função *createOffer()* é invocada para gerar a descrição SDP da sessão pretendida (GRIGORIK, 2013). Essa descrição SDP, que é uma representação textual simples , é então configurada como a descrição local da conexão no objeto *RTCPeerConnection* e, subsequentemente, enviada ao par remoto através de um canal de sinalização (ANDREASEN; BAUGHER; WING, 2006).

O SFU, é uma arquitetura de processamento de vídeo escalável que opera em um

ambiente de comunicação em rede. Sua função principal é codificar o vídeo de origem em conformidade com a taxa de *bits* fornecida pelo estimador de Rede e com resoluções espaciais e temporais determinadas pelo gerenciador de adaptação de resolução, que leva em consideração números pré-especificados de camadas espaciais e temporais. Embora a arquitetura SFU seja comumente adotada em aplicações WebRTC para gerenciamento eficiente de múltiplas streams de áudio e vídeo, sua utilização não é mandatória.

Alternativas como Multiponto (MCU) e abordagem P2P são igualmente viáveis, variando conforme requisitos específicos como número de participantes, qualidade de áudio/vídeo desejada e largura de banda, sendo assim o SFU desempenha um papel fundamental na adaptação e encaminhamento de fluxos de vídeo escaláveis em uma rede de comunicação, permitindo a otimização da qualidade do vídeo transmitido com base nas condições da rede e nas características do vídeo de origem (BAKAR; KIRMIZIOGLU; TEKALP, 2019). A arquitetura do SFU engloba diversos módulos, incluindo os seguintes exemplos de módulos:

- Gerenciador de Adaptação de Camadas : Este módulo recebe informações completas sobre as taxas de *bits* das camadas completas e medidas de atividade de movimento de todas as correntes de vídeo de entrada dos pares, juntamente com a estimativa da taxa de bits da rede do SFU para cada par de destino. Com base nessas informações, ele toma decisões sobre quais camadas espaciais e temporais serão encaminhadas para cada par de destino.
- Extrator e Empacotador de Fluxo de *Bits* : Este módulo recebe os fluxos de vídeo de todos os pares e as informações de seleção de camadas para cada par de destino, fornecidas pelo Gerenciador de Adaptação de Camadas. Ele extrai as camadas selecionadas para cada par de destino do fluxo de vídeo escalável, empacota essas camadas em pacotes e os encaminha para a rede.

2.3.4 Cifragem de Ponta a Ponta e Protocolos de Segurança

Na camada de transporte do modelo Interconexão de Sistemas Abertos (OSI), os protocolos TLS e *Datagram Transport Layer Security* (DTLS) são empregados para garantir a segurança nas comunicações. Esses protocolos fornecem mecanismos para a criptografia e a integridade de dados transmitidos sobre redes de computadores, assegurando comunicações protegidas entre entidades na rede. Esses protocolos atuam sobre um protocolo de camada de transporte confiável, como o TCP, assegurando serviços de segurança cruciais para transações online.

O TLS é uma evolução do *Secure Sockets Layer* (SSL), adotado como padrão pela *Internet Engineering Task Force* (IETF). Embora compartilhem muitas similaridades em sua estrutura e funcionalidade, existem distinções notáveis entre eles. Uma delas é a

questão da versão: enquanto o SSLv3.0 é compatível com o TLSv1.0, o TLS não suporta o uso do Fortezza no seu conjunto de cifras.

Outra diferença fundamental reside na geração de segredos criptográficos, com o TLS empregando uma função pseudoaleatória para a criação da chave mestra e dos dados das chaves, oferecendo uma camada adicional de segurança na geração de chaves (FOROUZAN, 2010).

Para resolver os desafios impostos pelo transporte baseado em datagramas do UDP, o DTLS introduz modificações específicas ao protocolo TLS. Isso inclui a implementação de um sistema semelhante ao TCP, mas apenas durante a sequência de *handshake*, assegurando a entrega ordenada e a integridade das mensagens.

O DTLS estende o protocolo de registro base do TLS, adicionando um deslocamento explícito de fragmento e número de sequência para cada registro de *handshake*. Isso permite a fragmentação de registros grandes em múltiplos pacotes e a reassemblagem pelo par, bem como a retransmissão de registros em caso de perda de pacotes, garantindo assim uma negociação segura do túnel de comunicação (GRIGORIK, 2013).

A *Multi-Level Security* (MLS) é um protocolo de segurança da informação que permite armazenar e processar dados de diferentes níveis de classificação — como não classificado, confidencial, secreto e ultra Secreto — em um único sistema. Ela garante que usuários com diferentes autorizações acessem apenas as informações para as quais têm permissão. O funcionamento do MLS é baseado em regras de acesso rigorosas. A primeira, “No-read-up”, impede que usuários de nível inferior acessem dados de nível superior.

A segunda, “*No-write-down*”, evita que informações de alto nível sejam escritas em níveis inferiores. Essas regras são parte do modelo Bell-LaPadula, amplamente reconhecido na implementação da MLS, que usa etiquetas de segurança em sujeitos (usuários ou processos) e objetos (arquivos, dispositivos) para definir as permissões de acesso.

Nas redes onde parte do sistema utiliza-se MLS, a classificação dos dados é incorporada nos pacotes de rede, assegurando que as informações mantenham seu nível de segurança durante a transmissão. Para proteger essas informações em trânsito, o protocolo IP *security* é utilizado, adicionando uma camada de segurança aos pacotes IP.

Sendo assim o MLS é uma abordagem que assegura o acesso a informações classificadas apenas por usuários autorizados, mantendo a integridade e confidencialidade dos dados sensíveis em sistemas e redes (MORSI; EL-FOULY; BADR, 2006) .

2.3.5 WebRTC Data Channels

A API DataChannel é projetada para fornecer um serviço de transporte genérico que permite a navegadores Web a troca de dados genéricos de forma bidirecional ponto-a-ponto. O trabalho de padronização dentro do IETF alcançou um consenso geral sobre o uso do *Stream Control Transmission Protocol* (SCTP) encapsulado em DTLS para lidar com tipos de dados não midiáticos.

A encapsulação de SCTP sobre DTLS sobre UDP, juntamente com *Interactive Connectivity Establishment* (ICE), oferece uma solução de transposição de *Network Address Translation* (NAT), bem como transferências protegidas por confidencialidade, autenticação de fonte e integridade (LORETO; ROMANO, 2014).

Além disso, essa solução permite que o transporte de dados trabalhe de forma harmoniosa com os transportes de mídia paralelos, e ambos podem também compartilhar um único número de porta da camada de transporte. O SCTP foi escolhido por suportar nativamente múltiplos fluxos com modos de entrega confiáveis ou parcialmente confiáveis. Ele permite a abertura de vários fluxos independentes dentro de uma associação SCTP em direção a um ponto final SCTP de pares.

Cada fluxo representa um canal lógico unidirecional, fornecendo a noção de entrega em sequência,sendo esta uma sequência de mensagens podendo ser enviada de forma ordenada ou não ordenada, sendo que a ordem de entrega é preservada apenas para todas as mensagens ordenadas enviadas no mesmo fluxo. Contudo, a API DataChannel foi projetada para ser bidirecional, o que significa que cada *DataChannel* é composto como um pacote de um fluxo SCTP de entrada e um de saída.

A interface *MediaStream* é utilizada para representar fluxos de dados de mídia, que podem ser de entrada ou saída, bem como locais ou remotos, como por exemplo, de uma webcam local ou de uma conexão remota. É importante destacar que um único *MediaStream* pode conter várias faixas (*tracks*), onde cada faixa possui um objeto *MediaStreamTrack* correspondente, representando uma fonte de mídia específica no agente do usuário.

Todas as faixas em um *MediaStream* são projetadas para serem sincronizadas quando renderizadas. Uma *MediaStreamTrack* representa conteúdo composto por um ou mais canais, onde os canais possuem uma relação bem definida e conhecida entre si.

Um canal é a unidade mais básica considerada nesta especificação da API (LORETO; ROMANO, 2014). Por exemplo, um *MediaStream* pode ser composto por uma única faixa de vídeo e duas faixas de áudio distintas, representando os canais esquerdo e direito.

2.4 Protocolo Matrix

A organização Matrix consiste em um protocolo aberto que visa promover comunicações seguras e descentralizadas. Conforme a missão estabelecida pela *Matrix.org Foundation*, busca-se oferecer um padrão único e não fragmentado, contribuindo para o benefício de todo o ecossistema, sem beneficiar ou privilegiar quaisquer participantes específicos (Matrix.org Foundation, 2023).

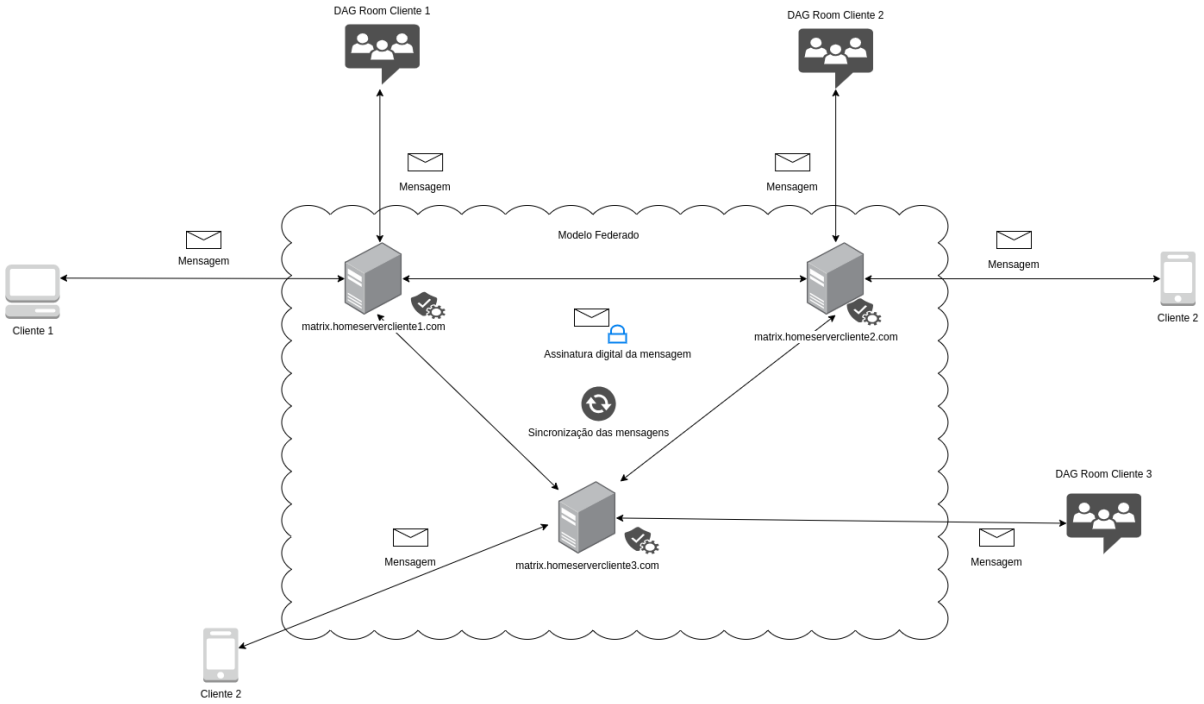
Na arquitetura do protocolo Matrix segunda sua especificação em (MATRIX. . . , 2023), os *"homeservers"* atuam como nós autônomos, encarregados de hospedar contas de usuários e gerenciar o armazenamento local do histórico de comunicações, estruturado como um *Directed Acyclic Graph* (DAG) dentro de *"rooms"*, espaços descentralizados de interação entre usuários.

As *"rooms"* funcionam como salas virtuais onde as mensagens são trocadas e registradas, permitindo a comunicação em tempo real. Os clientes, interfaces de usuário final, interagem com a rede Matrix por meio desses *"homeservers"*, que também se comunicam uns com os outros através de um mecanismo federado, assegurando a distribuição e sincronização das mensagens e a integridade do histórico de comunicações em toda a rede. No processamento de mensagens do Matrix, uma solicitação POST contendo um JSON inicia o envio de mensagens para o *"homeservers"* do emissor, que atribui um ID de evento à mensagem e a incorpora ao DAG da sala, assegurando sua conexão com o evento anterior não vinculado.

O *"homeserver"* responsável firma digitalmente a mensagem, incluindo as assinaturas de eventos antecedentes para preservar a integridade do histórico. O processo seguinte, a mensagem assinada é distribuída de forma segura via HTTPS aos demais *"homeservers"* participantes da sala, que realizam verificações de autenticidade e permissões para assegurar a imutabilidade e a autenticidade da comunicação. Inconsistências temporais, comuns em sistemas descentralizados, são geridas por meio da sincronização contínua do histórico de mensagens, permitindo a resolução de conflitos e a garantia da integridade histórica. A estrutura federativa do Matrix certifica a consistência eventual do histórico de mensagens em toda a rede, assegurando que todas as mensagens sejam acessíveis por todos os participantes, mantendo assim a coerência global da comunicação. A Figura 5 segue um diagrama para melhor entendimento da arquitetura.

O projeto *Waterfall* de acordo com (SIMONBRANDNER; CONTRIBUTORS, 2023), sua sinalização envolve a comunicação entre clientes e unidades de encaminhamento de fluxo (SFU) para gerenciar as sessões de comunicação em tempo real. A documentação de (SIMONBRANDNER; CONTRIBUTORS, 2023) aborda a sinalização para o uso de SFU em chamadas de grupo na Matrix sua arquitetura é ilustrativa na Figura 6. O objetivo é aprimorar chamadas em grande escala, utilizando SFU para retransmitir fluxos

Figura 5 – Arquitetura protocolo matrix P2P



Fonte: Próprio Autor

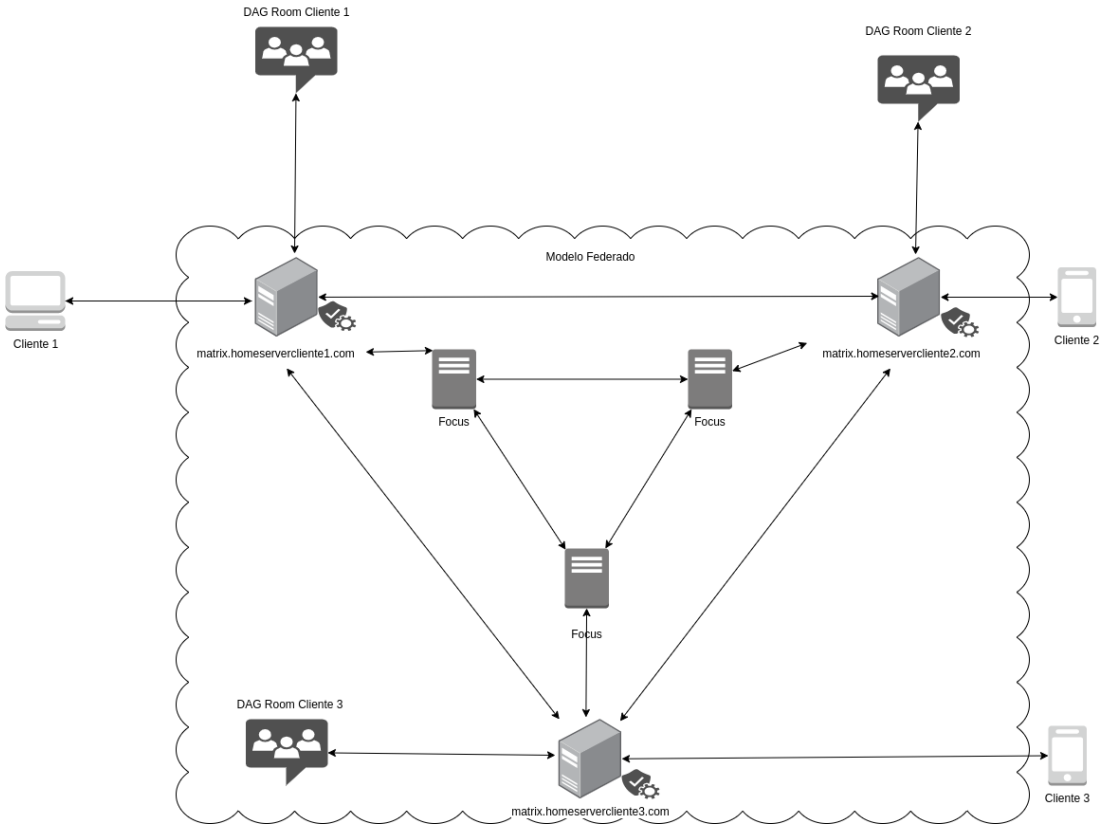
WebRTC entre pares. A sinalização permite aos usuários informar à SFU quais fluxos e resoluções desejam receber,a Figura 6 é possível notar a diferença da arquitetura do sistema ao adotar desejar uma melhor qualidade de mídia.

Cada SFU pode se conectar entre si, permitindo aos usuários direcionar a sinalização a uma SFU específica. O projeto *Waterfall* introduz campos opcionais no evento de estado *"m.call.member"* para sinalização ativa e preferencial de focos, a fim de otimizar a seleção do foco e gerenciar eficientemente os recursos de mídia. São feitos mecanismos para descobrir focos, determinar o melhor foco e gerenciar mudanças durante a chamada, além de abordar a segurança e o uso de canais de dados para sinalização de baixa latência.

O processo inicia com a verificação da existência de um *focus* preferencial, uma entidade responsável pela coordenação do fluxo de mídia. Se existir, o cliente tenta se conectar a esse *focus* sem enviar mídia para reservar um *slot*. Caso contrário, o cliente publica um evento *"m.foci.preferred"*. Se a chamada envolver mais de dois membros, incluindo o cliente, o processo segue para tentar conectar a um *focus* a partir dos eventos *"m.call.member"*".

Os *focus* são classificados do melhor para o pior, e o cliente passa pela lista ordenada para encontrar um que seja preferido por pelo menos outro membro. Se encontrado, o cliente se conecta ao *focus*,se não o cliente chama o *focus* ativo mais adequado na sala e publica um evento *"m.foci.active"*. Este mecanismo permite a gestão eficiente de recursos em chamadas com múltiplos participantes e a otimização da qualidade da chamada ao

Figura 6 – Arquitetura protocolo matrix com SFU



Fonte: Próprio Autor

selecionar o *focus* mais apropriado.Com os *homeservers* funcionando como nós autônomos e a estrutura de *rooms* descentralizada, a plataforma permite a criação e a gestão de aplicações de comunicação em tempo real. A especificação do Matrix, definindo uma API clara e um sistema federado robusto, permite que os desenvolvedores integrem e estendam funcionalidades como criptografia de ponta a ponta e sincronização de dados.

Essa abertura torna o Matrix uma escolha versátil para devs que desejam construir sistemas de comunicação customizados, desde mensagens instantâneas até soluções complexas de conferência de voz e vídeo, como exemplificado pelo projeto *Waterfall* para o tratamento de fluxo de mídia.

2.4.1 NAT: Tradução de Endereços de Rede

A medida que a Internet cresceu e os endereços IP se tornaram recursos escassos, foram desenvolvidos mecanismos, como sub-redes Protocolo de Internet versão 4 (IPv4) e endereçamento de rede e máscara, para otimizar o uso desses recursos.

Um terceiro mecanismo, conhecido como NAT, foi introduzido para permitir que vários computadores em uma rede interna compartilhem um único endereço IP globalmente válido (COMER, 2016).

O protocolo Protocolo de Internet versão 6 (IPv6) foi introduzido como uma solução para o esgotamento de endereços IP, posicionando dispositivos NAT na borda da rede para mapear combinações locais de IP e porta para uma ou mais combinações de IP e porta globalmente únicas. Esse espaço de endereço IP local por trás do tradutor pode ser reutilizado em várias redes.

Embora originalmente concebido como uma solução temporária, os dispositivos NAT tornaram-se uma parte essencial da infraestrutura da *Internet*, presentes em muitos *proxies*, roteadores, *firewalls* e outros dispositivos.

No entanto, o NAT apresenta desafios na entrega de datagramas UDP devido à falta de estado de conexão no UDP, diferentemente do TCP, que possui uma máquina de estado de protocolo bem definida. Isso resulta em uma desconexão fundamental, pois os dispositivos NAT precisam manter um estado sobre cada fluxo UDP, que é inerentemente sem estado. A ausência de uma sequência de término de conexão no UDP também coloca um desafio adicional aos dispositivos NAT, levando ao uso de *timers* para expirar registros de roteamento (GRIGORIK, 2013).

A NAT apresenta desafios significativos para aplicações P2P, especialmente para compartilhamento de arquivos e serviços de voz sobre IP. Em um sistema P2P, é fundamental que qualquer nó, referido como par A, possa iniciar uma conexão TCP com outro nó, denominado par B. No entanto, quando o par B está situado atrás de uma NAT, ele não consegue operar como servidor e aceitar conexões TCP (KUROSE; ROSS, 2013). Neste contexto, surge o WebRTC, uma tecnologia de comunicação em tempo real para navegadores, oferece soluções para superar as limitações da NAT em cenários P2P. Através do uso de técnicas como *Traversal Using Relays around NAT* (TURN), o WebRTC facilita a estabelecimento de conexões diretas entre pares, mesmo quando estes estão atrás de NATs.

2.4.2 STUN, TURN e ICE: Protocolos para Comunicação em Tempo Real

O *Session Traversal Utilities for NAT* (STUN) é um protocolo que facilita a comunicação através de tradutores de endereços de rede (NAT) (ROSENBERG et al., 2008). Ele permite que uma aplicação hospedeira identifique a existência de um NAT e adquira o par público de IP e porta alocado para a conexão. Para isso, o STUN necessita da assistência de um servidor externo posicionado na rede pública.

A aplicação envia uma solicitação de associação ao servidor STUN, que responde com o IP público e a porta do cliente, visíveis na rede externa. Esse processo permite que a aplicação descubra seu par de IP e porta públicos, estabelecendo entradas de roteamento NAT para que pacotes recebidos possam ser direcionados de volta à aplicação na rede interna. Além disso, o STUN mantém as entradas de roteamento NAT ativas através de

pings periódicos. Entretanto, o STUN pode não ser suficiente para todas as topologias de NAT e configurações de rede, e em alguns casos, o tráfego UDP pode ser bloqueado por *firewalls* ou outros dispositivos de rede, uma ocorrência não rara em redes empresariais (GRIGORIK, 2013).

Nessas situações, o protocolo TURN atua como alternativa, podendo operar sobre UDP e recorrer ao TCP caso necessário (MAHY; MATTHEWS; ROSENBERG, 2010). O TURN funciona por meio de um servidor de retransmissão público para encaminhar dados entre os pares, garantindo a conectividade, mas com o custo de não manter uma comunicação direta ponto a ponto e exigindo capacidade substancial do servidor de retransmissão. Por isso, o TURN é considerado uma solução de último recurso quando a conectividade direta falha.

O protocolo ICE é uma componente fundamental no mecanismo de comunicação P2P, essencial para o estabelecimento de comunicações de mídia baseadas em UDP em redes complexas. O ICE opera em conjunto com servidores STUN e TURN, possibilitando a transposição de caixas NAT e *firewalls*. Sua função principal é permitir que os navegadores descubram informações suficientes sobre a topologia da rede onde estão implantados, identificando assim o melhor caminho de comunicação possível.

Adicionalmente, ICE atua como uma medida de segurança, impedindo que páginas e aplicações Web não confiáveis enviem dados para *hosts* que não estão preparados para recebê-los. Durante o processo de sinalização, cada mensagem é inserida na *PeerConnection* receptora assim que chega. As APIs enviam mensagens de sinalização que, embora muitas aplicações tratem como blocos de "caixa preta", devem ser transferidas de maneira segura e eficiente para o outro par pela aplicação Web, por meio do servidor Web (LORETO; ROMANO, 2014).

2.4.3 RTP: Protocolo de Transporte em Tempo Real

O *Real-time Transport Protocol* (RTP), conforme definido na RFC 3550, representa um padrão essencial para a transmissão de dados em tempo real. Sua versatilidade permite o transporte de formatos comuns, como *Pulse-Code Modulation* (PCM), *Advanced Audio Coding* (ACC) e *MPEG Audio Layer III* (MP3) para áudio, bem como *Moving Picture Experts Group* (MPEG) e Padrão de Compressão de Vídeo (H.263) para vídeo (SCHULZRINNE et al., 2003). Além disso, o protocolo é adaptável para suportar formatos proprietários de áudio e vídeo.

Este protocolo na maioria das vezes é implementado sobre o protocolo UDP, no lado do transmissor seu processo envolve a encapsulação de uma parte de mídia em um pacote RTP, seguido pela encapsulação desse pacote em um segmento UDP, que é então encaminhado para a camada IP.

No lado do receptor, o pacote RTP é extraído do segmento UDP e a parte de mídia é desencapsulada para posterior decodificação e apresentação através de um transdutor. Um exemplo prático é o transporte de voz codificada em PCM a 64 kbits/s, onde o RTP é empregado para adicionar um cabeçalho que contém informações de codificação, número de sequência e marca de tempo, facilitando a decodificação e reprodução da parte de áudio.

A incorporação do RTP por diferentes aplicações de VoIP permite a interoperabilidade, favorecendo a comunicação entre produtos distintos. É importante destacar que o RTP não assegura a entrega pontual dos dados ou oferece garantias de QoS, sendo que seu encapsulamento é apenas relevante nos sistemas finais, não sendo distinguido por roteadores. Além disso, o RTP possibilita a criação de múltiplos fluxos independentes para origens distintas, sendo útil em cenários de videoconferência. Pacotes RTP também podem ser enviados em grupos, sendo que todos os remetentes e origens da sessão geralmente compartilham o mesmo grupo para transmitir seus fluxos RTP, formando uma sessão RTP (KUROSE; ROSS, 2013).

2.4.4 Codecs

Codificador (CODEC) é definido como um dispositivo ou aplicativo responsável pela compressão e descompressão de dados digitais, particularmente sinais de áudio e vídeo. Esses sistemas são essenciais em redes distribuídas para a eficiência na transmissão de dados de mídia, visando a redução do tamanho dos arquivos com a mínima perda de qualidade. Os *codecs* são amplamente empregados em aplicações como *streaming* de mídia, videoconferência e telefonia pela *internet*, desempenhando grande impacto na otimização do uso da largura de banda e na preservação da qualidade do conteúdo transmitido (COULOURIS et al., 2013).

O *codec* Opus, projetado para atender às necessidades de uma ampla gama de aplicações, opera com base em dois *codecs* distintos: SILK e CELT, além de um modo híbrido. Esse modo de operação é definido conforme a banda de frequência utilizada e a natureza do sinal de entrada, seja ele de fala ou música.

O CODEC trabalha com taxas de bits variando de 6 kb/s a 510 kb/s, com tamanhos de quadro suportados de 2,5 ms a 60 ms. No modo SILK e híbrido, utiliza um filtro passa-alta com frequência de corte adaptativa, variando conforme a relação sinal/ruído e a frequência do tom do falante (PCJIć; STANIć; PLETL, 2014).

O Opus possui um algoritmo de ocultação de perda de pacotes integrado, visando aumentar a robustez em canais não confiáveis ou congestionados. Sua implementação permite ajustes customizados entre a complexidade da codificação e a qualidade do áudio. A técnica de ocultação de perda de pacotes difere para os tipos de quadro no Opus:

enquanto o SILK utiliza a extrapolação, o CELT emprega a repetição de forma de onda com janela baseada no deslocamento do tom.

O VP8 é um CODEC de vídeo focado em aplicações baseadas na Web, concebido para operar eficientemente mesmo em condições de largura de banda limitada. Projetado para fornecer desde qualidade "assistível" até "visualmente sem perdas", ele é otimizado para uma variedade de dispositivos, desde móveis de baixa potência até *desktops* avançados.

O formato de vídeo da Web suportado pelo VP8 abrange amostragem de cor 4:2:0 e profundidade de cor de 8 bits por canal, com dimensões de imagem de até 16383x16383 pixels. As características técnicas incluem transformada híbrida com quantização adaptativa, referências de quadros flexíveis, predição intra e inter eficiente, interpolação sub-pixel de alta performance e filtragem adaptativa de *deblocking* em *loop*. Além disso, o VP8 incorpora codificação de entropia adaptável ao nível do quadro e particionamento de dados amigável ao processamento paralelo, beneficiando-se particularmente em processadores multi-core (BANKOSKI; WILKINS; XU, 2011).

3 PROPOSTA

Este capítulo explica a proposta de desenvolvimento de uma aplicação WebRTC, integrando o protocolo Matrix e adotando uma arquitetura baseada em *Selective Forwarding Unit* (SFU). O foco é analisar comparativamente o desempenho de diversos *codecs* de áudio e vídeo, visando aprimorar a qualidade de mídia e a eficiência no uso de recursos e na qualidade de serviço para usuário final.

3.1 Mapeamento de padrões e tecnologias

No desenvolvimento da arquitetura de sistema para a aplicação WebRTC com integração do protocolo Matrix e da arquitetura SFU, foi realizado um mapeamento criterioso de padrões e tecnologias pertinentes. Este processo envolveu a análise de padrões de comunicação em tempo real, arquiteturas de rede e protocolos de transmissão de dados, com o objetivo de otimizar a qualidade de transmissão e a eficiência de recursos.

O WebRTC, uma tecnologia chave neste estudo, foi escolhido devido à sua capacidade de permitir comunicação direta entre navegadores, sem a necessidade de *plugins* adicionais. A integração do protocolo Matrix e a utilização de sua sinalização, por outro lado, foi essencial para proporcionar uma estrutura de comunicação descentralizada e segura, facilitando a gestão de mensagens e chamadas de voz e vídeo em uma variedade de dispositivos.

A arquitetura baseada em *Selective Forwarding Unit* (SFU) foi adotada para melhorar a eficiência na transmissão de dados multimídia em cenários com múltiplos participantes. Esta escolha foi baseada na capacidade da SFUs de minimizar a largura de banda necessária por meio da seleção inteligente de fluxos de mídia a serem transmitidos entre os usuários, sendo assim será utilizado o projeto *Waterfall* como SFU.

3.2 Arquitetura do sistema e seu desenvolvimento

A arquitetura do sistema, que combinará o protocolo Matrix com a SFU *Waterfall*, visando otimizar a eficiência e escalabilidade da comunicação em tempo real. Esta seção abordará a inclusão de intermediários na distribuição de fluxos de mídia, bem como a integração de informações de protocolos como RTP e SCTP.

3.3 Testes e validação

Os testes da aplicação serão realizados em ambientes de rede cabeada e sem fio. Esta seção abordará os métodos e critérios de teste, como latência, qualidade de transmissão, uso de banda e estabilidade, visando identificar os *codecs* mais eficientes e de melhor qualidade em diferentes condições de rede.

3.4 Cronograma de atividades

Este cronograma está organizado em distintas etapas, cada uma contendo tarefas detalhadas e um tempo estimado para sua execução. Esta estruturação leva em conta tanto a complexidade inerente ao projeto quanto as demandas específicas relacionadas à arquitetura do sistema proposto no trabalho.

- **Fase 1: Planejamento e Definição da Arquitetura (4 semanas)**
  - Semanas 1-2: Revisão de documentação de APIs sobre WebRTC, Protocolo Matrix e arquitetura SFU. Definição inicial da arquitetura do sistema.
  - Semana 3: Elaboração de um esboço detalhado da arquitetura. Identificação de requisitos técnicos e de negócios.
  - Semana 4: Finalização do plano de arquitetura. Preparação de documentos de especificação técnica.
- **Fase 2: Desenvolvimento e Implementação (8 semanas)**
  - Semanas 5-6: Configuração do ambiente de desenvolvimento. Início da implementação da arquitetura base WebRTC.
  - Semanas 7-8: Integração do Protocolo Matrix. Testes iniciais de conectividade e comunicação.
  - Semanas 9-10: Implementação da arquitetura SFU. Testes de performance em comunicação de áudio e vídeo.
  - Semanas 11-12: Otimização de *codecs* de áudio e vídeo. Avaliação de desempenho e ajustes.
- **Fase 3: Desenvolvimento da Aplicação de Jogo (4 semanas)**
  - Semanas 13-14: Desenvolvimento do protótipo inicial do jogo. Integração com a arquitetura WebRTC.
  - Semanas 15-16: Implementação das funcionalidades do jogo. Testes de integração e desempenho.

- **Fase 4: Testes e Validação (6 semanas)**
  - Semanas 17-18: Testes em ambientes de rede cabeada e sem fio. Coleta de métricas para a arquitetura e o jogo.
  - Semanas 19-20: Análise detalhada dos resultados dos testes. Ajustes na arquitetura e na aplicação de jogo.
  - Semanas 21-22: Testes finais de validação do sistema e do jogo. Preparação para a implementação e lançamento.
- **Fase 5: Documentação e Conclusão (2 semanas)**
  - Semana 23: Documentação técnica detalhada da arquitetura, do processo de desenvolvimento e do jogo.
  - Semana 24: Preparação do relatório final do trabalho. Revisão e ajustes finais.

REFERÊNCIAS

ANDREASEN, F.; BAUGHER, M.; WING, D. *Session Description Protocol (SDP) Security Descriptions for Media Streams*. [S.l.], 2006. 19

BAKAR, G.; KIRMIZIOGLU, R. A.; TEKALP, A. M. Motion-based rate adaptation in webrtc videoconferencing using scalable video coding. *IEEE Transactions on Multimedia*, v. 21, n. 2, p. 429–441, 2019. 20

BANKOSKI, J.; WILKINS, P.; XU, Y. Technical overview of vp8, an open source video codec for the web. In: *2011 IEEE International Conference on Multimedia and Expo*. [S.l.: s.n.], 2011. p. 1–6. 29

COMER, D. E. *Redes de Computadores e Internet*. [s.n.], 2016. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788582603734/. E-book. ISBN 9788582603734. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788582603734/. 9, 13, 14, 25

COULOURIS, G. et al. *Sistemas Distribuídos*. Grupo A, 2013. E-book. ISBN 9788582600542. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788582600542/. 9, 10, 11, 12, 13, 16, 28

DESHPANDE, M. Integration of webrtc with sip – current trends. In: . [s.n.], 2015. Disponível em: https://api.semanticscholar.org/CorpusID:28845530. 7

FERREIRA, A. G. *Interface de Programação de Aplicações (API) e Web Services*. Editora Saraiva, 2021. E-book. ISBN 9786553560338. Disponível em: https://app.minhabiblioteca.com.br/#/books/9786553560338/. 11

FOROUZAN, B. A. *Comunicação de dados e redes de computadores*. Grupo A, 2010. ISBN 9788563308474. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788563308474/. 21

GHOSH, A. et al. 5g evolution: A view on 5g cellular technology beyond 3gpp release 15. *IEEE Access*, v. 7, p. 127639–127651, 2019. 14

GRIGORIK, I. *High-Performance Browser Networking*. First. Sebastopol, CA: O'Reilly Media, Inc., 2013. First release on 2013-09-09. See http://oreilly.com/catalog/errata.csp?isbn=9781449344764 for release details. ISBN 978-1449344764. Disponível em: http://oreilly.com/catalog/9781449344764. 13, 15, 17, 19, 21, 26, 27

GUPTA, A.; BARTOS, R. User experience evaluation of http/3 in real-world deployment scenarios. In: *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. [S.l.: s.n.], 2022. p. 17–23. 17

JAKOBSSON, C. *Peer-to-peer communication in web browsers using WebRTC*. Dissertação (Mestrado) — University of Umeå, Sweden, Spring 2015. 5

KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet: Uma Abordagem Top-Down*. 6th. ed. São Paulo: Pearson, 2013. 634 p. ISBN 9788581436777. 14, 26, 28

LOMBARDI, A. *WebSocket*. First edition. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2015. Editors: Simon St. Laurent and Brian MacDonald; Production Editor: Colleen Lobner; Copyeditor: Kim Cofer; Proofreader: Sharon Wilkey; Indexer: Wendy Catalano; Interior Designer: David Futato; Cover Designer: Karen Montgomery; Illustrator: Rebecca Demarest; Revision History for the First Edition: 2015-09-04: First Release. Disponível em: http://safaribooksonline.com. 18

LORETO, S.; ROMANO, S. P. *Real-Time Communication with WebRTC*. 1. ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2014. First release 2014-04-15. ISBN 9781449371876. Disponível em: http://oreilly.com/catalog/errata.csp?isbn=9781449371876. 22, 27

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. 27

MATRIX Specification. 2023. Acessado em 21 de novembro de 2023. Disponível em: https://spec.matrix.org/latest/. 23

Matrix.org Foundation. *About Matrix*. 2023. https://matrix.org/about/. Acessado em 23 de novembro de 2023. 23

MONTEIRO, E. R. et al. *Sistemas Distribuídos*. Grupo A, 2020. E-book. ISBN 9786556901978. Disponível em: https://app.minhabiblioteca.com.br/#/books/9786556901978/. 12

MORSI, W.; EL-FOULY, T.; BADR, A. Using ipsec to secure multi-level data classification in mls networks. In: *2006 6th International Conference on ITS Telecommunications*. [S.l.: s.n.], 2006. p. 817–821. 21

MYERS, J. *IMAP4 ACL extension*. [S.l.], 1997. 16

PCJIć, A.; STANIć, P. M.; PLETL, S. Analysis of packet loss prediction effects on the objective quality measures of opus codec. In: *2014 IEEE 12th International Symposium on Intelligent Systems and Informatics (SISY)*. [S.l.: s.n.], 2014. p. 33–37. 28

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. 26

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. 27

SHKLAR, L.; ROSEN, R. *Web Application Architecture: Principles, Protocols and Practices*. The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England: John Wiley & Sons Ltd, 2003. Library of Congress Cataloging-in-Publication Data: 2003011759. British Library Cataloguing in Publication Data available. Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire. This book is printed on acid-free paper responsibly manufactured from sustainable forestry. ISBN 0-471-48656-6. Disponível em: www.wiley.com. 15

SIMONBRANDNER; CONTRIBUTORS other. *MSC3898: SFU*. 2023. GitHub repository: matrix-org/matrix-spec-proposals. Acessado em 21 de novembro de 2023. Disponível em: https://github.com/matrix-org/matrix-spec-proposals/blob/SimonBrandner/msc/sfu/proposals/3898-sfu.md. 23

ZINNER, T. et al. Comparison of the initial delay for video playout start for different http-based transport protocols. In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. [S.l.: s.n.], 2017. p. 1027–1030. 16