UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO POLITÉCNICO DA UFSM
CURSO DE SISTEMAS PARA INTERNET

Denilson dos Santos Ebling

DISTRIBUIÇÃO DE CONTEÚDO NA INTERNET COM UM MODELO HÍBRIDO CDN-P2P

Santa Maria, RS
2020

Denilson dos Santos Ebling

DISTRIBUIÇÃO DE CONTEÚDO NA INTERNET COM UM MODELO HÍBRIDO CDN-P2P

Trabalho de Conclusão de Curso apresentado ao Sistemas para Internet da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Tecnólogo em Sistemas Para Internet**

Orientador: Prof. Dr. Rafael Gressler Milbradt

Santa Maria, RS

2020

Denilson dos Santos Ebling

DISTRIBUIÇÃO DE CONTEÚDO NA INTERNET COM UM MODELO HÍBRIDO CDN-P2P

Trabalho de Conclusão de Curso apresentado ao Sistemas para Internet da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Tecnólogo em Sistemas Para Internet**

Aprovado em 12 de Fevereiro de 2020:

Rafael Gressler Milbradt, Dr. (UFSM)
(Presidente/Orientador)

Alencar Machado, Dr. (UFSM)

Thales Nicolai Tavares, Tecg. (UFSM)

Santa Maria, RS

2020

RESUMO

DISTRIBUIÇÃO DE CONTEÚDO NA INTERNET COM UM MODELO HÍBRIDO CDN-P2P

AUTOR: DENILSON DOS SANTOS EBLING
ORIENTADOR: RAFAEL GRESSLER MILBRADT

Atualmente, com a rápida expansão da Web e a quarentena causada pela pandemia da COVID-19, o consumo de serviços de streaming vídeo está no seu auge, com isso, é um grande desafio para CDNs distribuir o conteúdo de maneira robusta para que o usuário final tenha uma experiência fluida e, ao mesmo tempo, ter uma infra-estrutura altamente escalável. CDNs oferecem uma ótima experiência para o usuário com sua baixa latência, enquanto redes P2P oferecem baixo custo de manutenção e alta escalabilidade. Com isso, o intuito desse trabalho é implementar um cliente e servidor que utilizem uma arquitetura hibrida CDN-P2P e avaliar a desempenho do mesmos, com o intuito de demostrar os pontos fortes e fracos da arquitetura.

Palavras-chave: CDN. P2P. Sistemas Distribuídos.

ABSTRACT

CONTENT DISTRIBUTION ON INTERNET WITH A CDN-P2P HYBRID MODEL

AUTHOR: DENILSON DOS SANTOS EBLING
ADVISOR: RAFAEL GRESSLER MILBRADT

Currently, with a rapid expansion of the Web and the quarantine caused by the pandemic of COVID-19, the consumption of video streaming services is at its peak, with this, it is a great challenge for CDNs to distribute the content in a robust way so that the end user has a fluid experience while having a highly scalable infrastructure. CDNs offer a great user experience with their low latency, while P2P networks have low maintenance costs and high scalability. Thus, the purpose of this work is to implement a client and server that uses a hybrid CDN-P2P architecture and evaluate their performance, in order to demonstrate the architecture's strengths and weaknesses.

Keywords: CDN. P2P. Distributed Systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CDN	Content Distribution Network
DASH	Dynamic Adaptive Streaming over HTTP
DNS	Domain Name System
DTLS	Datagram Transport Layer Security
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
JVM	Java Virtual Machine
NAT	Network Address Translation
P2P	Peer-to-Peer
SCTP	Stream Control Transmission Protocol
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TURN	Traversal Using Relays around NAT
UDP	User Datagram Protocol
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium

SUMÁRIO

1 INTRODUÇÃO

O consumo de conteúdo na Internet, como vídeos, livestreams e download de arquivos, vem crescendo exponencialmente nos últimos anos graças a redução do custo dos dispositivos e serviços, aumentando significativamente a porcentagem da população com acesso à Internet. Esse problema cresceu ainda mais rápido devido ao período de isolamento causado pela pandemia da COVID-19, com isso o uso de aplicações como YouTube, NetFlix, Twitch aumentou significativamente, segundo Sharma (2020), o tempo gasto consumindo serviços de streaming de video cresceu 57%, tempo em serviços de streaming de música cresceu 39% e tempo ouvindo podcasts cresceu 14%.

Redes de Distribuição de Conteúdo (CDNs) são vastamente usadas para a distribuição de conteúdo na Internet, pois são rentáveis e apresentam baixa latência para o usuário final, porem, devido a seu modelo clássico de cliente-servidor, elas geram grandes custos em hardware e energia. Uma alternativa são Redes P2P, pois não oferecem custo de hardware sendo que é utilizado o dispositivo do próprio cliente para servir para os clientes que estão acessando o mesmo conteúdo, por causa disso, redes P2P, são altamente escaláveis, pois, quando maior o número de clientes acessando um conteúdo, maior será o número de servidores para atender os clientes. Redes P2P são baratas, mas altamente instáveis, pois dependem da quantia de clientes na mesma, e da banda de upload dos mesmo.

1.1 OBJETIVOS

O principal objetivo desse trabalho é desenvolver uma rede de distribuição de conteúdo híbrida (CDN-P2P), a fim de diminuir o custo de mantimento de uma CDN, aumentar a escalabilidade dos sistemas, utilizando recursos providos dos usuários finais.

1.1.1 Objetivos específicos

Para atingir a finalidade deste trabalho, os seguintes objetivos específicos foram determinados:

- Identificar os problemas na arquitetura atual mais usada (CDNs).
- Propor um modelo híbrido, que adiciona comunicação P2P opcional entre usuários finais de um CDN.

- Desenvolver um CDN que utilize o modelo proposto:
  - Apurar os benefícios e custos dessa solução.
- Incentivar a pesquisa e devolvimento sobre métodos de distribuição de conteúdo na web mais escaláveis.

1.2 ORGANIZAÇÃO DO TRABALHO

Esse trabalho está organizado em 3 capitulos, sendo estruturado da seguinte forma:

- O Capitulo 1 (Introdução) apresenta o problema a ser pesquisado, os objetivos desse trabalho e suas contribuições;
- O Capitulo 2 (Referencial Teórico) descreve conceitos importantes sobre distribuição de conteúdo na web e tecnologias necessárias para o entendimento desse trabalho.
- O Capitulo 3 (Trabalhos Relacionados) apresenta trabalhos relacionados comparando seus pontos fortes e fracos ao desenvolvido.

2 REFERENCIAL TEÓRICO

O Referencial Teórico apresenta conceitos e tenologias que são fundamentais para o entendimento desse trabalho. Sua estrutura apresenta as seguintes seções e subseções: A seção 2.1 (Distribuição de Conteúdo na Web) apresenta as duas soluções que o trabalho desenvolvido estará integrando. A qual desencadeia o estudo específico sobre os mesmos nas subseções 2.1.1 (CDN) e 2.1.3 (Comparação entre CDN e P2P). A seção 2.2 (WebRTC) apresenta a tecnologia utilizada que possibilita comunicação P2P entre navegadores de Internet. E por fim, 2.4 (TypeScript) e 2.5 (Clojure), apresentam as linguagens de programação utilizadas para o desenvolvimento no navegador e servidor, respectivamente.

2.1 DISTRIBUIÇÃO DE CONTEÚDO NA WEB

Para distribuição de conteúdo com tamanho grande, como vídeos, em larga escala, existem duas alternativas usadas atualmente: Redes de Distribuição de Conteúdo (CDNs) e Redes *Peer-to-Peer* (P2P). CDNs é o método mais usado atualmente, pois apresenta baixa latência e alta estabilidade para o usuário final, porém isso vem com um custo, manter uma rede de distribuição é caro e não escalável. Redes P2P são altamente escaláveis e bem baratas, pois utilizam o próprio dispositivo do usuário, porém apresentam alta latência e baixa estabilidade. A seguir serão apresentados os dois modelos citados, outros modelos híbridos como o proposto nesse trabalho serão mencionados no Capitulo 3.

2.1.1 CDN

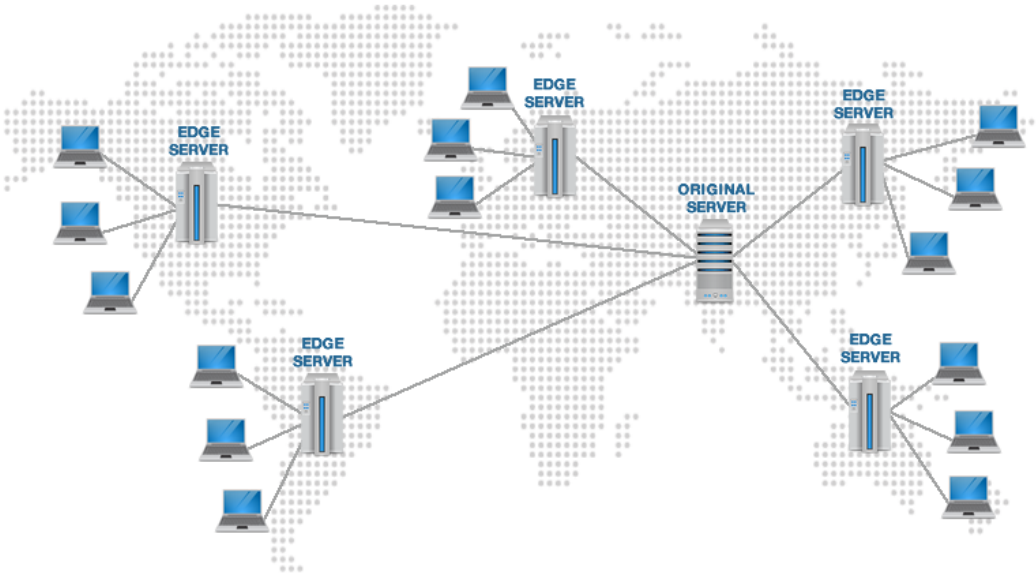
Content Distribution Network (CDN), segundo Tanenbaum (2010), é uma coleção de servidores distribuídos em diferentes localizações geográficas configurados por um provedor, onde são usados para servir conteúdo para clientes.

As CDNs, como menciona Tanenbaum (2010), foram popularizadas no final dos anos 90, onde Web estava nos primórdios do seu crescimento. Akamai foi o primeiro grande CDN e se tornou o líder da indústria. Empresas pagam Akamai para distribuir seu conteúdo para os clientes, para que eles tenham sites com baixo tempo de carregamento que clientes gostam de usar.

Como é descrito por Peng (2004), um conteúdo em uma CDN é distribuído até os usuá-

rios finais, por uma arquitetura baseada em árvore. A raiz dessa árvore é chamada de nó origem, é o servidor do cliente que contratou o serviço, o conteúdo é distribuído pela árvore, onde é armazenado e feito cache, até chegar ao nó mais próximo do usuário, que é chamado de nó de borda. Esse nó é o servidor que atende o usuário final.

Figura 1 – Arquitetura de uma CDN



Fonte: (CASE STUDY: AMERICA MOVIL AND VIAVI, 2019)

Segundo Huang (2008), CDNs geralmente adotam duas diferentes filosofias para o posicionamento de seus servidores:

A primeira é entrar profundamente nos provedores de Internet, implantando servidores para a distribuição de conteúdo dentro do *Point of Presence* (PoP) do provedor (local onde o provedor mantém o equipamento necessário para permitir o acesso local dos seus usuários à Internet). Com isso os servidores ficam próximos dos usuários, aumentando o desempenho notado por eles em latência e taxa de transferência, porém, com esse design altamente distribuído, manter e gerenciar a rede é difícil. Esse design requer algoritmos mais sofisticados para embaralhar dados entre os servidores na Internet.

Para os provedores de Internet, é benéfico ter um nó CDN dentro de sua rede, pois diminui a quantia de banda *upstream* que eles precisam, reduzindo custos. Somando a isso, o nó CDN melhora a responsividade para os clientes do provedor, melhorando a imagem do

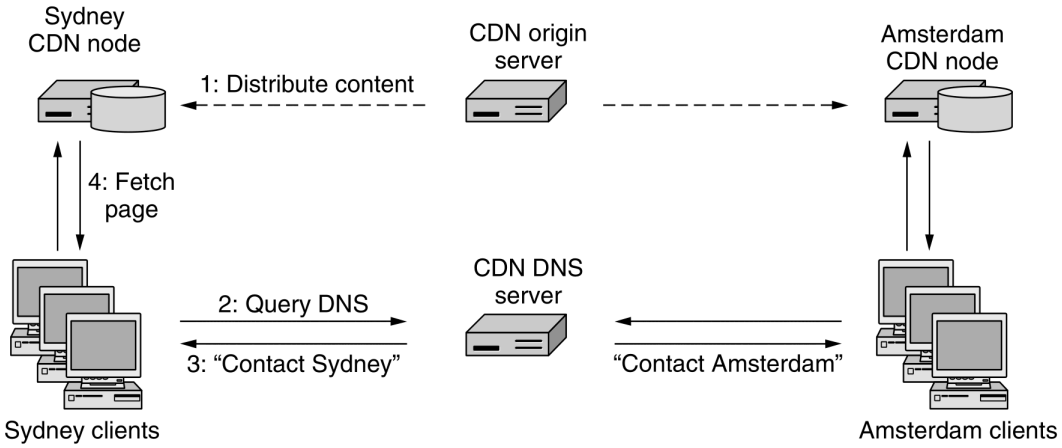
provedor e dando uma vantagem sobre outros provedores que não tem um nó CDN em sua rede.

A segunda filosofia é montar grandes *clusters* em um número menor de localizações chaves e conectar esses centros utilizando uma conexão de alta velocidade privada. Ao invés de estarem nos PoPs dos provedores, esses CDNs tipicamente colocam seus *clusters* em localizações perto dos PoPs de múltiplos grandes provedores. Comparado com a primeira, esse design resulta em custo de manutenção e gerência menor, porém apresenta um tempo de resposta maior.

Assim que os *clusters* são instalados o CDN replica o conteúdo entre eles. O conteúdo não é copiado por completo, é utilizado uma técnica de *pull* simples, se o cliente faz uma requisição pedindo um conteúdo que não está atualmente guardado no *cluster*, o *cluster* busca em um repositório central ou de outro *cluster* e guarda uma cópia, sendo assim, fazendo cache do conteúdo.

Após os *clusters* estarem posicionados, é necessário ter uma estratégia para interceptar a requisição do usuário e redirecionar para o *cluster* CDN mais próximo, fazendo com que ele obtenha a menor latência possível. Para isso, a maioria dos CDNs utilizam redirecionamento por meio de DNS, utilizando um servidor DNS da CDNs. Por Exemplo, um cliente quer buscar uma imagem com a URL *www.dominio.com/imagem.png*. Para buscar essa imagem, o navegador de Internet irá primeiramente fazer uma consulta DNS para resolver *dominio.com* para um endereço IP, nesse momento, ao invés de retornar o mesmo endereço IP para todas as requisições, o servidor DNS irá olhar para o endereço IP do cliente que está fazendo a consulta, e retornará o IP do nó da CDN mais prómixo ao cliente. Então se um cliente em Santa Maria faz um consulta para o servidor DNS, ele retornará o endereço do nó de Santa Maria, mas se um cliente em Porto Alegre faz a consulta, o servidor retornará o endereço do nó de Porto Alegre, exemplo ilustrado na figura 2.

Figura 2 – Redirecionamento por DNS



Fonte: (TANENBAUM; WETHERALL, 2010)

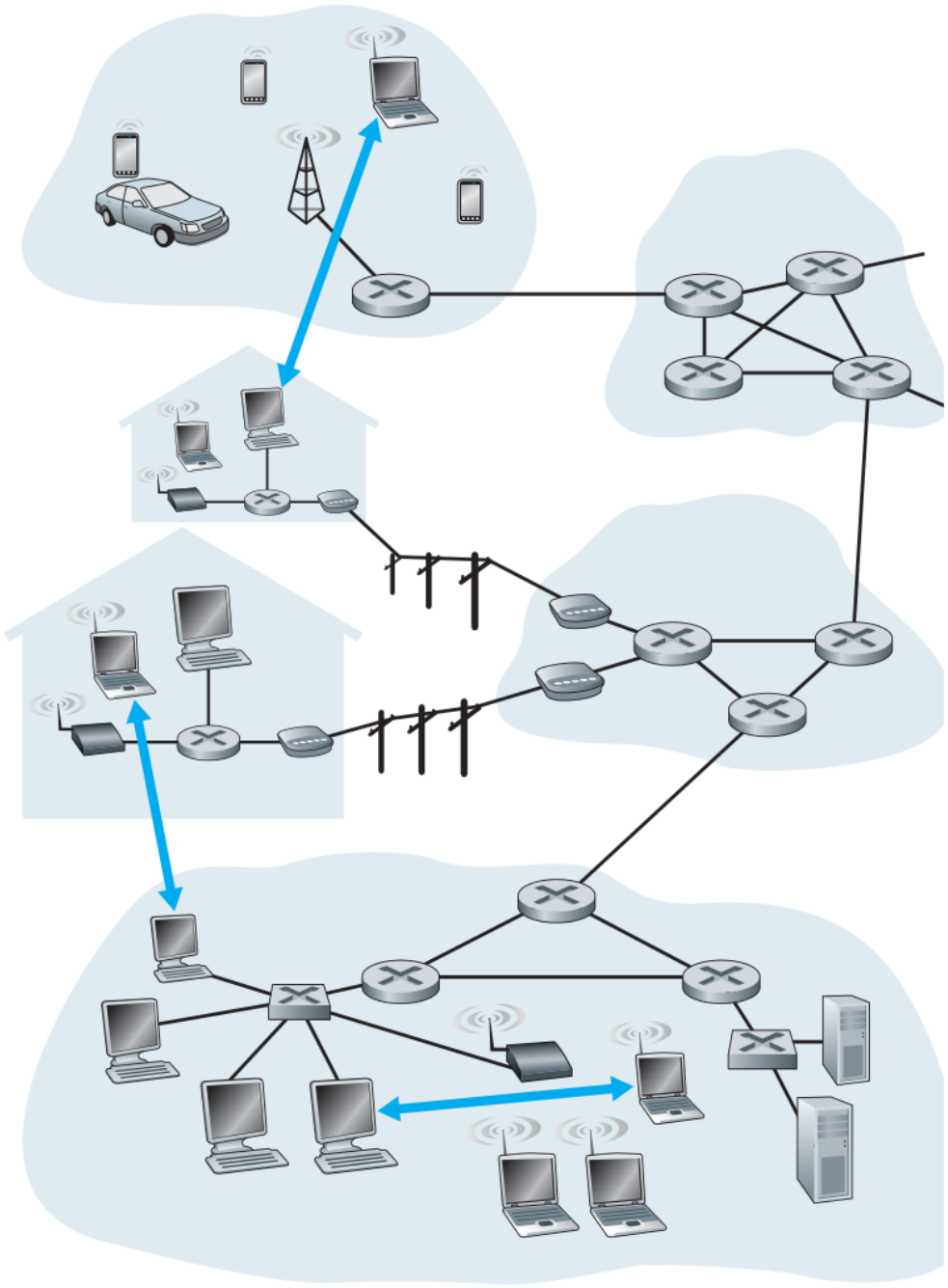
Para distribuir grandes cargas de dados de vídeo ou áudio, CDNs é o mais utilizado atualmente, graças a sua rentabilidade e estabilidade.

2.1.2 Redes Peer-to-Peer

Redes Peer-to-Peer (P2P), como menciona Tanenbaum (2010), começaram a surgir com força nos anos 90, a primeira aplicação que popularizou a tecnologia foi o Napster, onde os usuários compartilhavam arquivos de música, codificados no formato MP3, de maneira descentralizada, sendo que um computador conectado à sua rede, atuava ao mesmo tempo, como cliente e servidor. O sucesso de Napster fez dele o protogonista de uma grande luta jurídica entre a indústria fonográfica e as redes de compartilhamento de música na Internet, pois, era compartilhado música com direitos autorais, sem a permissão do autor da obra, consequentemente, o serviço foi fechado. Hoje, o BitTorrent é o protocolo P2P mais popular, ele é usado para compartilhamento de arquivos e é responsável por uma grande fração do tráfico de Internet.

Diferente de uma CDN, como explica Kurose (2012), uma rede P2P (*peer-to-peer*) tem pouca ou nenhuma dependência em servidores dedicados, ao invés disso, é explorada a comunicação direta entre hosts autônomos na rede, onde estes podem atuar, ao mesmo tempo, como cliente e servidor, esses nós são chamados de *peers*. Eles não são controlados por um provedor de serviço, são *desktops*, *notebooks* e celulares controlados pelos usuários. A arquitetura é chamada de *peer-to-peer*, pois, os *peers* não se comunicam através de um servidor dedicado, como mostra figura 3.

Figura 3 – Comunicação entre hosts em uma rede P2P



Fonte: (KUROSE; ROSS, 2012).

Como menciona Kurose (2012), uma das maiores vantagens das arquiteturas P2P, é que são auto escaláveis, mesmo que cada *peer* gere carga na rede por requisitar um recurso, cada um também contribui servindo para outros *peers*. Considerando uma rede P2P com n usuários, cada uma conexão com 1Mbps de banda, a capacidade de upload da rede P2P é n Mbps.

2.1.3 Comparação entre CDN e P2P

CDNs e redes P2P apresentam modelos de distribuição de conteúdo opostos, onde CDNs oferem alta rentabilidade, redes P2P oferecem alta tolerância a falha e baixo custo, como mostra a tabela 1.

Tabela 1 – Comparação entre as arquiteturas CDN e P2P

Comparação	CDN	P2P
Capacidade e Escalabilidade	Capacidade é limitada e custo de expansão é alto	Capacidade aumenta com o aumento de peers e custo de expanção é baixo
Rentabilidade	Alta rentabilidade	Baixa rentabilidade e estabilidade
Estabilidade	Alta estabilidade	Baixa estabilidade
Monitoramento da fonte de conteudo	Pode ser monitorado	Difícil de ser monitorado
Garantia de Qualidade de Serviço	Pode ser garantido dentro da capacidade máxima do serviço	Melhor esforço. Não pode ser controlado
Nó de serviço	Um nó CDN é um servidor, e clientes simplesmente acessão os nós servidores que são heterogêneos	Nós clientes podem prover conteúdo para outros nós clientes, portando nós de serviço são heterogêneos

Fonte: Adaptado de (LU et al., 2012).

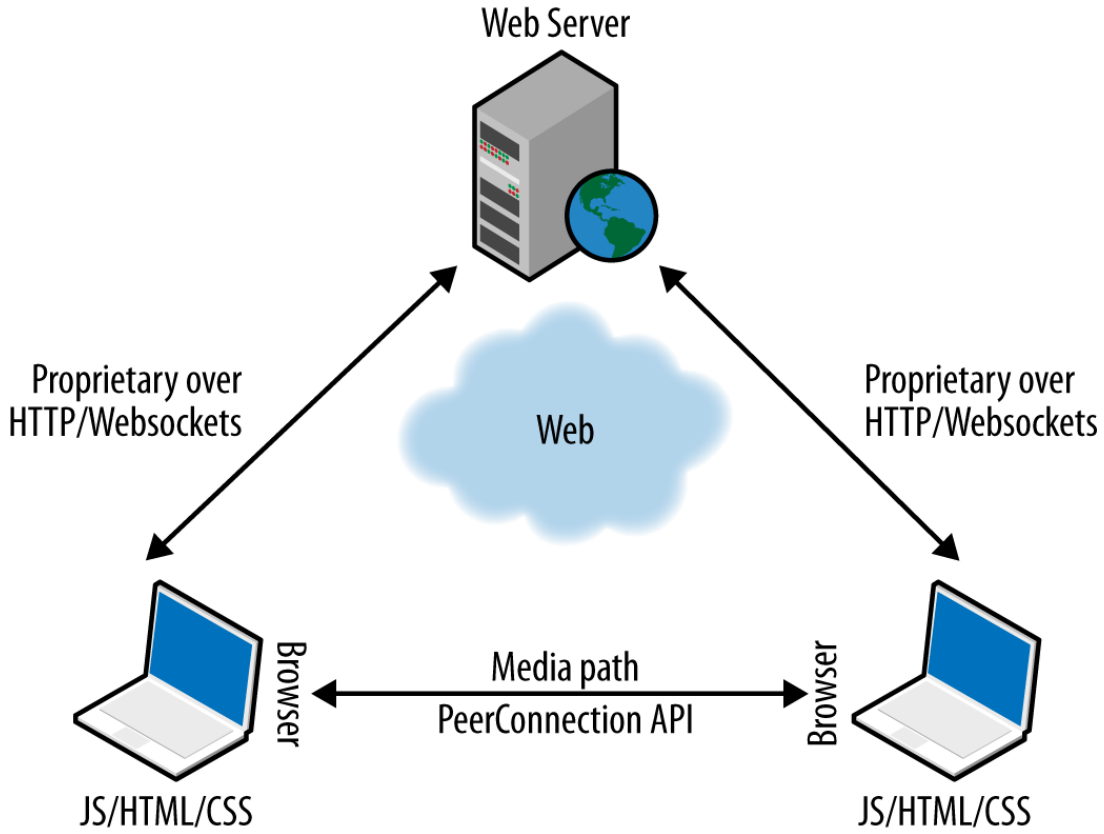
2.2 WEBRTC

Com o objetivo de suprir a necessidade da existência de método padronizado de comunicação *peer-to-peer* entre navegadores, *Web Real-Time Comunication (WebRTC)* (BERGKVIST et al., 2012), como expõe Loreto (2014), é uma tecnologia proposta pela *World Wide Web Consortium (W3C)* e *Internet Engineering Task Force (IETF)*, que é formado por um conjunto de protocolos e APIs JavaScript, que juntas, provêm funcionalidades necessárias para estabelecer canais de áudio, vídeo e dados genéricos, permitindo a comunicação *peer-to-peer* entre navegadores tanto em desktops, quanto em dispositivos móveis, onde todos os dados são transmitidos usando DTLS (RFC6347, 2012) (protocolo da camada de aplicação baseado no TLS, que provém garantias de segurança similares, porém, sobre UDP ao invés de TCP). As três principais APIs providas são:

- *RTCPeerConnection*: permite criar e gerenciar conexões;
- *MediaStream*: permite que a aplicação envie e receba audio e vídeo;
- *RTCDataChannel*: permite envio e recebimento de dados arbitrarios, imitando o comportamento de WebSockets.

Entre as APIs disponibilizadas, *RTCPeerConnection* é a que expõe uma interface para criar e gerenciar conexões *peer-to-peer*, ela representa uma associação com um *peer* remoto. A negociação para estabelecer uma conexão entre dois *peers* é coordenadas pela troca de mensagens de controle (chamada de protocolo de sinalização) sobre um canal fornecido por meios não especificados, mas geralmente por um código na página provida pelo servidor web, utilizando *XMLHttpRequest* ou *WebSocket*. Estabelecer comunicação entre *peers*, tipicamente envolve a troca de IPs e porta entre os mesmos, isso gera uma série de problemas quando os dispositivos estão atrás de roteadores ou firewalls que utilizam a técnica Network Address Translation (NAT), para que seja estabelecida uma conexão, mesmo nessas condições. Loreto (2014), descreve que PeerConnection utiliza o protocolo *Interactive Connectivity Establishment (ICE)* (RFC8445, 2018) que define uma técnica para travessia de NAT utilizando STUN (RFC5389, 2008) e TURN (RFC5766, 2010). Em um cenário típico, a comunicação pode ser representada por um triangulo, onde no topo se encontra um servidor que se encarrega de distribuir a aplicação Web via HTTP e da sinalização entre os dois navegadores para que a conexão P2P seja estabelecida, como mostra a figura 4.

Figura 4 – Diagrama mostrando a comunicação P2P com WebRTC



Fonte: Loreto (2014).

MediaStream, como é descrito por Bergkvist (2012), é a representação abstrata de uma transmissão de áudio e/ou vídeo, ela permite que a aplicação solicite áudio e video da plataforma, assim como manipule e processe o dado. Uma transmissão pode ser entrada ou saída, pode ser local, como uma webcam ou microfone, ou remota, e pode conter várias faixas de áudio ou video, onde cada faixa é uma instância de *MediaStreamTrack*, que representa uma fonte de conteúdo. Todas as faixas dentro de uma *MediaStream* são sincronizadas ao serem renderizadas.

RTCDataChannel, como é descrito Bergkvist (2012) permite o envio e recebimento de dados arbitrários entre *peers*, onde a API provida modela o comportamento de *Web Sockets*, fornecendo um canal bi-direcional. RTCDataChannel permite a customização da maneira de entrega dos dados, onde cada canal pode ser configurado para optar ou não por envio de mensagens confiáveis com garantia de ordem. Essa configuração é possível graças ao uso de SCTP que suporta múltiplas transmissões com configurações diferentes nativamente. A partir de um RTCPeerConnection, podem ser criados múltiplos *RTCDataChannel*.

2.3 WEBSOCKET

WebSocket (RFC6455, 2011) permite uma comunicação de duas vias entre um cliente e um servidores com baixa latência que funciona em cima do protocolo TCP. *WebSocket* é composto pela *WebSocket* API, padronizada pela W3C e pelo Protocolo *WebSocket*, padronizado pela IETF, e foram desenvolvidos fazendo parte do padrão HTML 5. Ao contrário do HTTP, WebSocket implementa uma comunicação *full-duplex* assíncrona. Seu objetivo é tornar eficiente comunicação de baixa latência, o que até então, era feita usando *pooling* com HTTP.

2.4 TYPESCRIPT

TypeScript é uma linguagem de programação multiparadigma que tranpila para JavaScript, segundo Bierman (2014), TypeScript é um superconjunto de JavaScript, ou seja, todo programa JavaScript é um programa TypeScript. Apesar de seu sucesso, JavaScript continua uma linguagem problemática para o desenvolvimento e manutenção de grandes aplicações, TypeScript tenta mitigar esse problema adicionando uma série de características à JavaScript, entre elas estão:

- Sistema de tipagem estática gradual.
- Tipos de dados algébricos, TypeScript permite produto e soma de tipos.
- Equivalência de tipos baseada em estrutura (A é equivalente a B se A é composto pelas mesmas propriedades de B).
- Fácil interoperabilidade com JavaScript, pois, é um superconjunto dá linguagem.
- Inferência de tipos, TypeScript utiliza inferência de tipos para minimizar o número de anotações de tipos que precisam ser providas explicitamente.
- Apagamento completo de tipos, os tipos da linguagem são eliminados em tempo de compilação, portanto, não tem verificação de tipos em tempo de execução.
- Possibilidade de escolher qualquer versão do EcmaScript para ter como alvo de compilação, facilitando suporte para navegadores mais antigos.

Essas características fazem com que o TypeScript seja uma otina linguagem para escrever código cliente que utilize tecnologias como WebRTC devido sua capacidade de interoperabilidade com JavaScript e fácil capacidade de depuração, pois o tranpilador gera código JavaScript legível, além disso, TypeScript não adiciona nenhum custo em tempo de execução.

2.5 CLOJURE

Clojure, como coloca Halloway (2009), é uma linguagem de programação funcional impura dinâmica para a maquina virtual Java (JVM). É um dialeto LISP que apresenta as seguintes principais caraterísticas:

- Fundações em programação funcional provendo um conjunto de estruturas de dados imutáveis e persistentes.
- Concorrência simplificada, o modelo de concorrência de Clojure não é baseado em travas, ao invés disso, ela prove alternativas como memoria transacional em software, *agents* e *atoms*.
- Provêm interoperabilidade com Java, pois, chamar código Java a partir de Clojure é rápido e direto, pois não passa por uma camada de tradução, permitindo o fácil uso das bibliotecas do grande ecossistema Java.
- Por ser um dialeto LISP, possui um poderoso sistema de macros que facilita metaprogramação.

Dialetos LISP, como Clojure, diferem de linguagens de programação convencionais, no sentido que o código é escrito utilizando as estruturas de dados da linguagem, essa propriedade presente em algumas linguagens de programação é chamada de Homoiconicidade, termo que foi popularizado por Kay (1969). Tendo a figura 5 como exemplo, os parenteses são a definição de uma lista na linguagem, + é uma função e os números 1 e 2 são os argumentos dessa função, dado isso, quando essa expressão for avaliada pelo intérprete, resultará na aplicação da função + com os argumentos 1 e 2, que resulta em 3 onde nesse exemplo, “=> ” é utilizado para indicar o resultado da avaliação da expressão.

Figura 5 – Expressão clojure

```
(+ 1 2)
=> 3
```

Fonte: acervo pessoal.

Devidoa à Clojure ser uma linguagem com foco em concorrência e que prioriza imutabilidade, esta torna-se Clojure uma ótima linguagem para implementações de servidores altamente concorrentes para CDNs. Além disso, sua interoperabilidade com Java faz com que Clojure herde o ecossistema de bibliotecas do mesmo.

3 TRABALHOS RELACIONADOS

Dado o problema de distribuição de conteúdo de mídia de maneira escalável, usando uma arquitetura P2P que complementa a tradicional cliente-servidor adotada pela maioria dos CDNs atualmente, foi feita uma análise de três sistemas, entre eles, dois que utilizam uma rede P2P estruturada com múltiplas árvores Cooperative Networking (CoopNet) e SplitStream, e outra que utiliza uma híbrida, de múltiplas árvores e malha, chamado LiveSky.

3.1 COOPNET

CoopNet (Padmanabhan, (PADMANABHAN; SRIPANIDKULCHAI, 2002; PADMANABHAN et al., 2002)) é um sistema com o objetivo tratar o problema de um tráfego *flash Crowd* (surto rápido no volume de requisições que chegam ao servidor) implementando uma rede P2P baseada em múltiplas árvores que complementa o modelo cliente-servidor. Ele utiliza a técnica de *Multiple description coding (MDC)* onde uma *stream* de mídia é fragmentada em $n > 1$ *substreams* para transmití-las em múltiplos caminhos em uma árvore multicast onde um servidor central fica responsável por construir e gerenciar as árvores de distribuição. O servidor tem conhecimento completo de todas as árvores e quando um cliente deseja entrar na rede, ele informa o servidor que responde com uma lista de nós pais, sendo um por árvore. A procura por um nó pai para o novo cliente envolve uma travessia na árvore, iniciado pela raíz, até encontrar um nível onde existe um nó com uma largura de banda livre suficiente para servir o novo cliente. Como o estado não é distribuido e, portanto, fica contido em um servidor, isso o torna um grande ponto de falha e faz com que não seja muito flexível em casos que são necessários mais servidors para atender uma região, pois o conhecimento da rede P2P não é compartilhado entre eles. Outro problema, que é inerente ao dá arquiterura baseada em árvore, é o aproveitamento da capacidade de upload, pois, em árvores onde o grau é maior do que 1, a maioria dos nós serão folhas.

3.2 SPLITSTREAM

Esse problema é abordado em SplitStream (Castro, 2003), um sistema de streaming proposto pelo *Microsoft Research Center*, onde é possível criar redes P2P, nas quais, cada *peer* contribui com a mesma quantia de banda que recebe. Para atender esse requisito, a mídia

é dividida em n faixas, onde cada uma é transmitida em uma árvore, de maneira similar ao CoopNet, porém, tais árvores são construídas de maneira que um *peer* é um nó interno em pelo menos uma árvore e, é uma folha nas restantes.

3.3 LIVESKY

LiveSky (Yin, 2009), um sistema de *live streaming* híbrido CDN-P2P, usado comercialmente, que foi desenvolvido e implantado por ChinaCache. LiveSky tem a mesma proposta de complementar a arquitetura tradicional de um CDN com uma rede P2P, com o objetivo de reduzir custos. A arquitetura típica de um CDN é dividida em várias camadas C, onde C_0 é a mais próxima da fonte do conteúdo e $Cn - 1$ é a mais próxima do usuário final, na qual um nó nessa camada é chamado de "servidor de borda". Em LiveSky, além de servir os usuários finais, os servidores de borda atuam como um *tracker* para iniciar a rede P2P, assim, a rede *overlay* funciona como servidor de borda, ou seja, os *peers* que se comunicam na mesma rede estão atribuídos ao mesmo servidor. Os servidores de borda decidem se um cliente vai ser atendido em modo CDN ou se será redirecionado para ser atendido pela rede P2P, com base em sua capacidade total de banda, carga atual e número de *peers* para determinada mídia. LiveSky támbem adere o problema do tempo de inicialização quando um novo peer se junta a rede, em típicos sitemas P2P isso implica em um atraso significante no início da reprodução do video, nesta solução, quando um novo cliente envia sua primeira requisição ao servidor de borda é enviado uma resposta com um numero pré-especificado de segmentos do video e enquando o buffer de video do cliente é preenchido, o *playback* do video é iniciado, ao mesmo em que o processo para entrar na rede P2P é inicado.

3.4 BEMTV

E, por fim, BemTV (Barbosa, 2014), é um sistema que, assim como o proposto neste trablalho, utiliza WebRTC para a comunicação peer-to-peer. Em BemTV, rede peer-to-peer atua como uma camada de caching para um CDN tradicional, onde é utilizado o protocolo HLS (RFC8216, 2017), onde é possivel buscar um segmento da stream tanto pela rede peer-to-peer, quando pelo CDN.

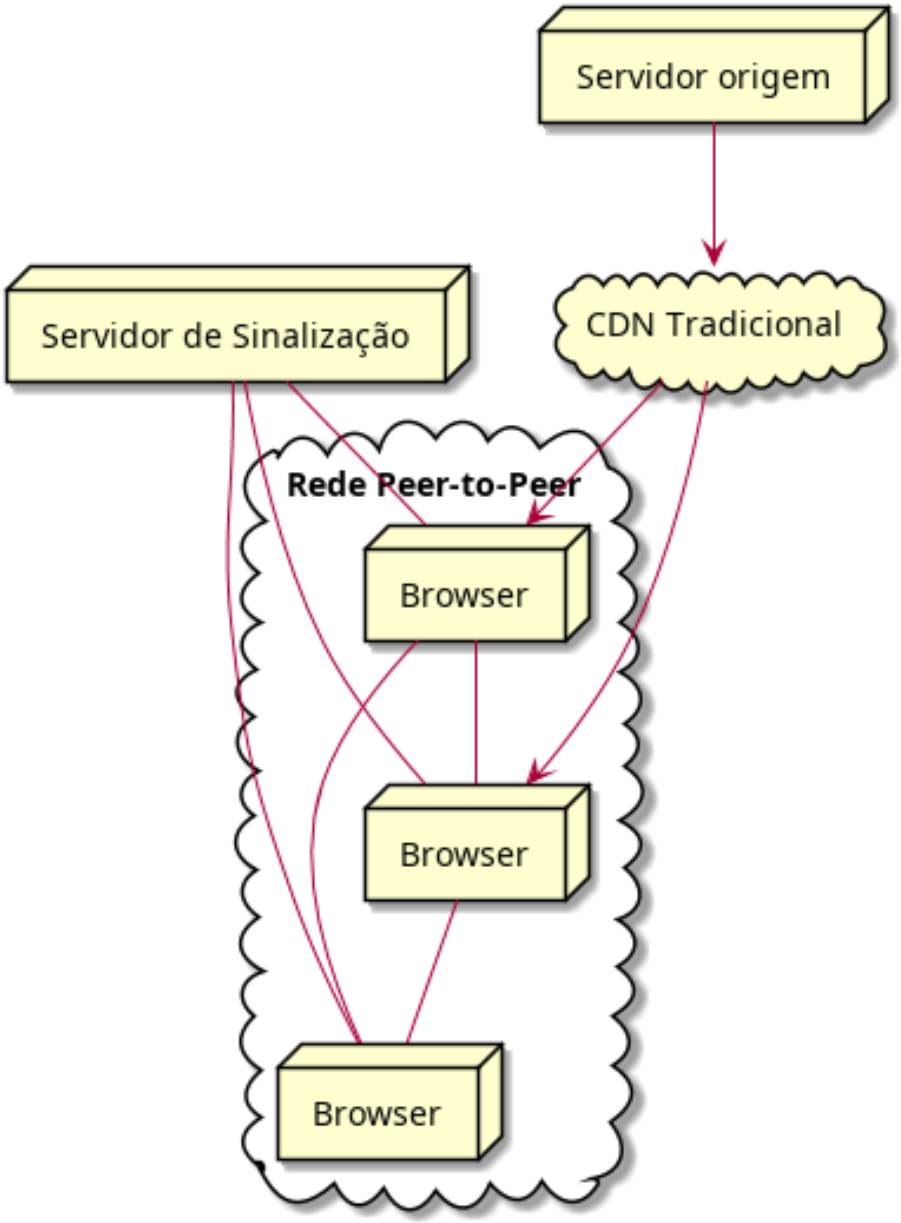
4 ARQUITETURA DO SISTEMA PROPOSTO

Neste capitulo é apresentado o sistema proposto, bem como sua arquitetura e a interação entre seus componentes, destacando partes relevantes para seu entendimento.

O sistema é dividido em duas partes pricipais, um servidor de sinalização, que tem como função organizar e gerenciar a rede *Peer-to-Peer*, e uma biblioteca cliente, que vai suprir os dados que permitem o *playback* da stream de vídeo. Em uma visão macro, o sistema funciona como uma camada de caching em cima de um CDN tradicional, onde o cliente, recebe a stream de um CDN e distribui partes da mesma para outros *peers*, essa arquitetura é ilustrada pela figura 6.

Essa arquitera visa suprir o objetivo de fácil integração com sistemas já existentes, junto a isso, já existe excelentes CDNs no mercado, que possuem alta rentabilidade, segurança e performance, tornando desejavel a utilização dos mesmos.

Figura 6 – Arquitetura do sistema proposto



Fonte: Acervo pessoal

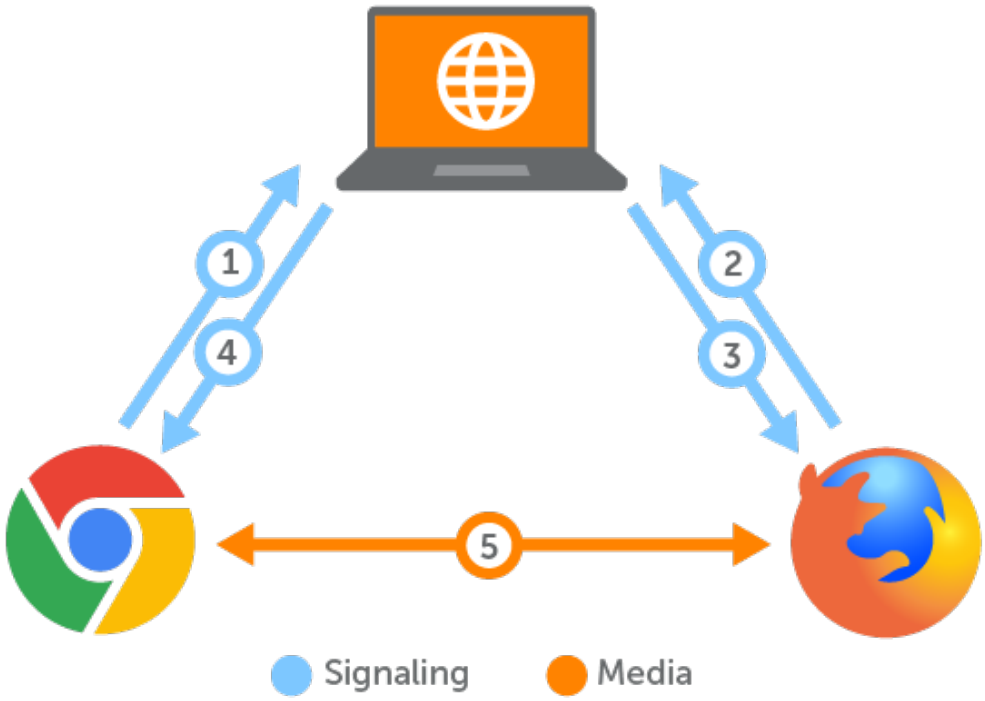
4.1 SERVIDOR DE SINALIZAÇÃO

O servidor de sinalização tem como papel principal organizar e gerenciar a rede P2P. A biblioteca cliente utiliza WebRTC, que é uma tecnologia *Peer-to-Peer* completa para a troca de dados em tempo real entre navegadores de Internet, porem, é necessária uma maneira de descoberta e negociação para que dois dispositivos, em redes diferentes, possam localizar um ao outro e estabelecer conexão. O servidor não lida com trafego de vídeo em si, mas sim com o

descobrimento de clientes.

O servidor de sinalização atua como intermediário para que dois clientes WebRTC possam se conhecer e negociar uma conexão, para assim então os clientes começarem a troca de mensagens Peer-to-Peer, como é ilustrado na figura 7, onde as setas em azul indicam as mensagens de sinalização entre o servidor e o cliente, e em laranja a troca de dados via Peer-to-Peer.

Figura 7 – Servidor de Sinalização

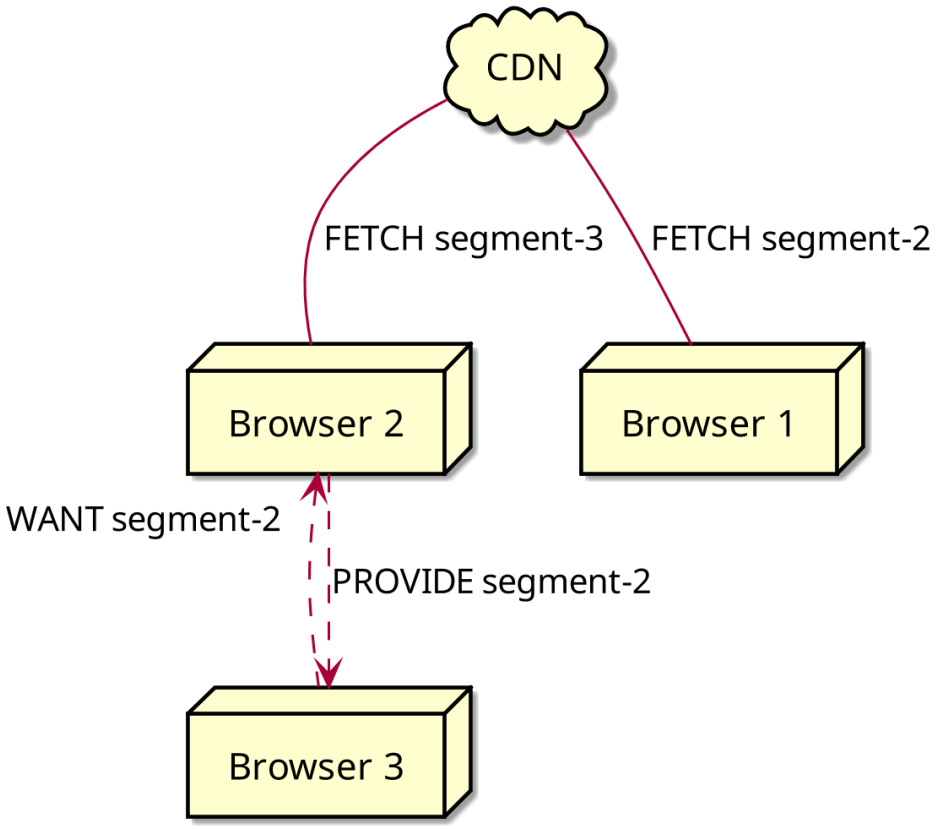


Fonte: (WEBRTC SIGNALING SERVERS: EVERYTHING YOU NEED TO KNOW | WOWZA, 2020)

4.2 BIBLIOTECA CLIENTE

A biblioteca cliente tem como função permitir o *playback* de *streams* que utilizam o protocolo DASH (STOCKHAMMER, 2011), onde os segmentos do vídeo são inicialmente buscados do CDN que os distribui e armazenados temporariamente, até que seja estabelecido a conexão na de rede P2P. Enquanto a reprodução do vídeo distribuído pelo CDN acontece, o cliente estabelece conexão com o servidor de sinalização, onde o mesmo irá organizar sua entrada na rede *Peer-to-Peer*, após se juntar a rede, a aplicação começa a buscar segmentos do vídeo a partir de outros *peers*, essa comunicação é demostrada na figura 8.

Figura 8 – Busca de segmentos pela biblioteca cliente



Fonte: Acervo pessoal

4.3 DESENVOLVIMENTO

Neste capitulo é apresentado o desenvolvimento dos componentes do sistema proposto, junto a isso, exemplos de códigos simplificados com o objetivo de facilitar o entendimento.

4.3.1 Reprodução inicial do vídeo

Para o desenvolvimento do sistema foi necessário coordenar múltiplas tecnologias Web de maneira que uma complementa a outra. Um dos objetivos principais do sistema proposto é a fácil integração com sistemas já existentes, tendo isso em mente, o principal componente do sistema é a biblioteca cliente, que provem os dados necessários para reproduzir um vídeo solicitado através da interface MediaSource, sendo possível reproduzir o vídeo com o *player* HTML5 de um browser moderno utilizando a tag <`video`>, como é demostrado na figura9, ou com qualquer player que suporte a interface MediaSource.

Figura 9 – Interface pública da bilioteca

```
const mediaSrcFactory = new MediaSourceFactory(SIGNALING_ADDR);

const video = document.createElement('video');
video.src = mediaSrcFactory.createMediaSourceUrl(VIDEO_URL);
video.autoplay = true;
```

Fonte: Acervo pessoal.

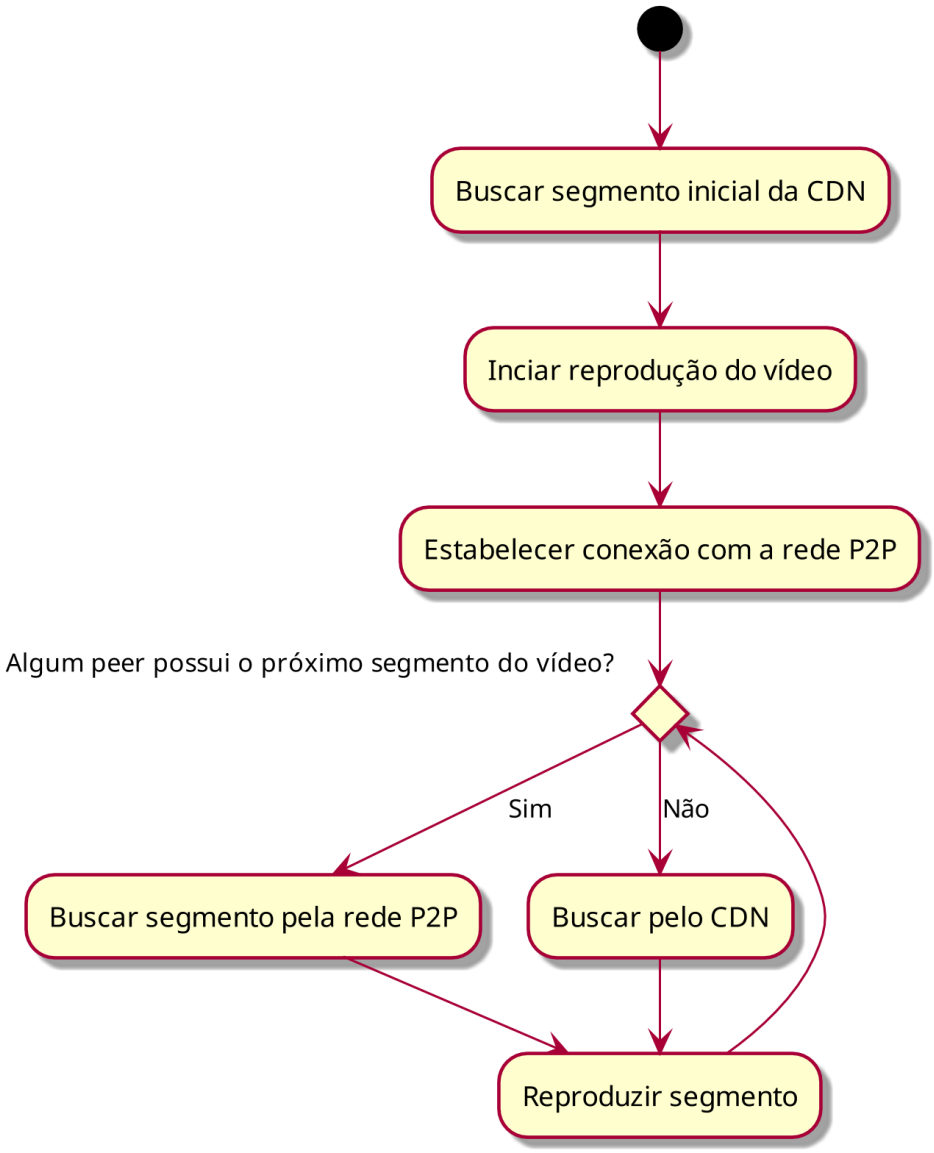
O ponto de entrada para a biblioteca é a classe *MediaSourceFactory*, que possui somente um método público chamado *createMediaSourceUrl*, como mostra a figura 10, onde o *MediaSource* é instanciado, após isso, é utilizado a *callback onsourceopen* para fazer a configuração inicial do objeto, essa *callback* é chamada após o *player* de vídeo concluir seu carregamento inicial. O papel do MediaSourceFactory é construir um objeto *MediaSource* com as *callbacks* necessárias para buscar o vídeo via CDN, enquanto estabelece conexão com a rede P2P em segundo plano, essa processo é enfatizado na figura 11, isso é feito utilizando o evento *onupdateend* do source *buffer*, que chama a função para buscar o próximo segmento. Essa função é criada pelo metodo *makeNextSegmentFn*, que recebe um SourceBuffer como parametro, e retorna uma função que ao ser chamada busca o proximo segmento e adiciona ele ao buffer, essá função utiliza o conexão Peer-to-Peer da classe *MediaSourceFactory* e decide se o próximo segmento deve ser buscado da CDN ou de outro *peer*.

Figura 10 – Interface pública da bilioteca

```
class MediaSourceFactory {
  private readonly conn: P2PConnection;

  constructor(signalingServerAddr: string);

  public createMediaSourceUrl(videoUrl: string): string {
    const ms = new MediaSource();

    ms.onsourceopen = () => {
      const sb = ms.addSourceBuffer('video/mp4; codecs="avc1.4d401f');

      const nextSegmentFn = this.makeNextSegmentFn(sb);
      sb.onupdateend = nextSegmentFn;

      // Fetch init segment
      this.fetchChunk(sb, videoUrl);
    }

    return URL.createObjectURL(ms);
  }
}
```

Fonte: Acervo pessoal.

Figura 11 – Lógica para reprodução do vídeo



Fonte: Acervo pessoal

4.3.2 Ingresso na rede *Peer-to-Peer*

Para que os segmentos possam ser recuperados de outros peers é necessário, primeiramente, entrar na rede *Peer-to-Peer* do sistema. Esse processo envolve entrar em contato com o servidor de sinalização para que a conexão com outros peers seja negociada.

A comunicação com servidor de sinalização é feita via *WebSocket*, utilizando mensagems em *JSON*, permitindo comunicação bidirecional de baixa latência entre o servidor e os clientes, outra propriedade importante do *WebSocket* é que ele permite ao servidor fazer *broad-*

cast de uma mensagem para todos os clientes conectados a ele. O uso de mensagens em *JSON* permite uma comunicação mais pratica entre o cliente e o servidor, além disso, é possivel tirar proveito do sistema de tipagem do *TypeScript*, a figura 12 ilustra a definição dos tipos de mensagems no cliente, para cada mensagem é definido uma interface especifica com os atributos da mesma, após isso é definido um tipo chamado *WSMessage* que é a união (denotada por |) de todos os tipos de mensagem.

Figura 12 – Tipagem das mensagems WebSocket no cliente

```ts
export interface WSSuccessMessage {
  kind: 'connect';
  clientId: string; // uuid
};

export interface WSOfferMessage {
  kind: 'offer';
  offer: RTCSessionDescriptionInit;
};

export interface WSAnswerMessage {
  kind: 'answer';
  answer: RTCSessionDescriptionInit;
};

export interface WSUserConnectedMessage {
  kind: 'user-connected';
  clientId: string;
};

export interface WSErrorMessage {
  kind: 'error';
  message: string;
};


export type WSMessage =
  WSSuccessMessage
  | WSOfferMessage
  | WSAnswerMessage
  | WSUserConnectedMessage
  | WSErrorMessage;
```

Fonte: Acervo pessoal.

O processo de ingresso inicia com a conexão ao servidor de sinalização, onde ao esta-

belecer uma conexão o cliente recebe uma identificação em forma de *UUID* e é feito broadcast para todos os clientes conectados, notificando os memsos que um novo peer surgiu, como é ilustrado na figura 13.

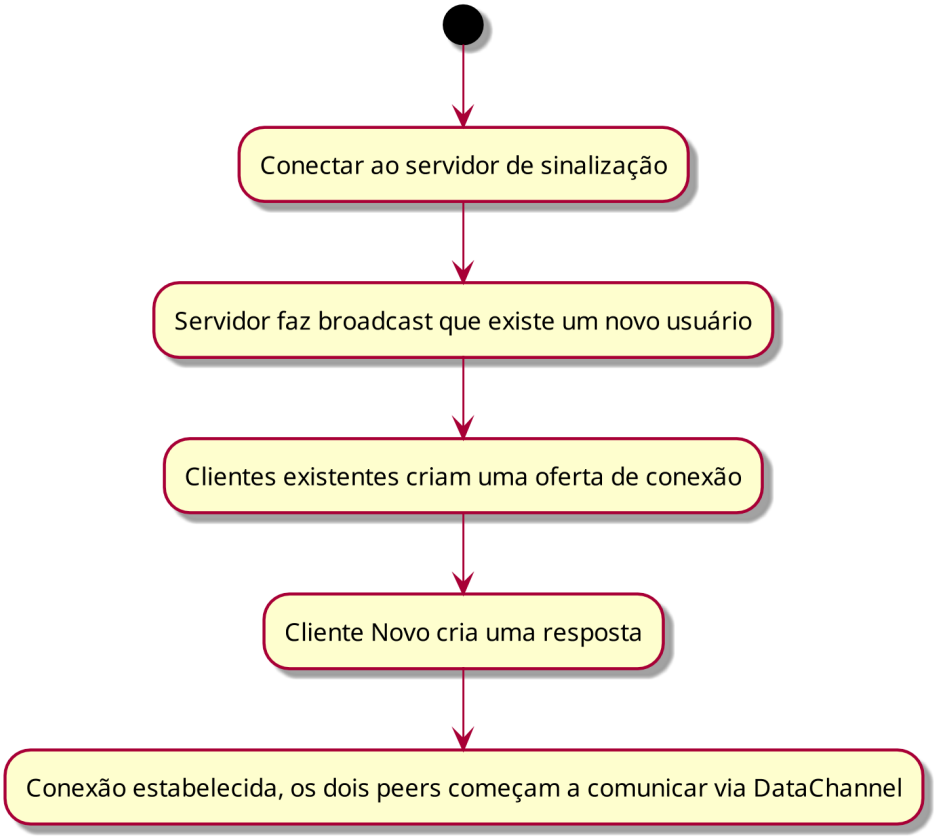
Figura 13 – Tratamento do evento de conexão no servidor

```
(defn on-connect!
  "Handler para o evento de conecção, adicionar peer ao conjunto
  global de peers"
  [ch]
  (let [client-id (java.util.UUID/randomUUID)]
    (log/info "New peer connected, assigned id " client-id)
    ;; Adiciona o peer a um registro global
    (swap! peers assoc ch client-id)
    ;; Envia o id do cliente para o Peer que se conectou
    (s/send! ch (json/write-str {:kind :connect
                                 :clientId (str client-id)}))
    ;; Faz broadcast para todos os outros cliente, notificando-os
    ;; sobre o surgimento do novo cliente
    (broadcast {:kind :user-connected
                :clientId (str client-id)}
               ch)))
```

Fonte: Acervo pessoal.

Ao ser notificado sobre o ingresso do novo cliente, os outros peers começão o processo de negociação de uma conexão com o mesmo, formando uma malha de peers, onde todos os peers estão interconectados. O processo de negociação inicia com os clientes existentes criando uma oferta de negociação, utilizando o método *createOffer* esse disponibilizado pela tecnologia WebRTC, essa oferta é enviado para o servidor de sinalização, que atuando como um intermediário, envia a oferta para o cliente novo, que ao recebe-lá, cria uma resposta utilizando *createAnswer*, que também é enviado pelo servidor de sinalização, ao receber a resposta, os clientes concluem o processo de negociação estabelecendo uma conexão, a partir desse ponto, o servidor de sinalização não é mais utilizado, e a comunicação entre *peers* é feita utilizando um *DataChannel* criado com a conexão WebRTC, este processo é ilustrado pela figura 14.

Figura 14 – Processo de negociação de conexão



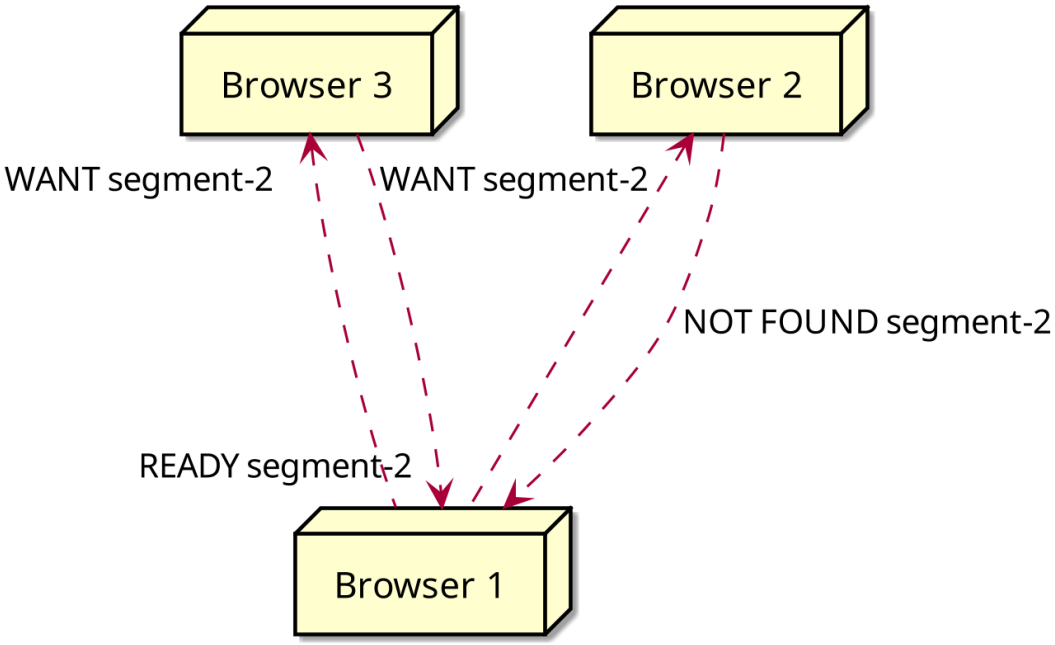
Fonte: Acervo pessoal

4.3.3 Compartilhamento de Segmentos na Rede *Peer-to-Peer*

Após o ingresso na rede *Peer-to-Peer*, o cliente pode requisitar e enviar segmentos na mesma. O processo de recuperação de um segmento na rede *Peer-to-Peer* inicia com o cliente fazendo *broadcast* que deseja recuperar o segmento, como é ilustrado na figura 15, após enviar a mensagem, os *peers* podem responder com um dos códigos de *status* abaixo:

- *READY*: O *peer* possui o segmento e pode atender a requisição;
- *NOT FOUND*: O *peer* não possui o segmento;

Figura 15 – Recuperação de um segmento via *Peer-to-Peer*



Fonte: Acervo pessoal

Após enviar a requisição, é esperado até que um *peer* responda com o código de *READY*, assim que o primeiro *peer* responder, é enviado uma requisição de recuperação do segmento desejado para o mesmo, e após ele receber o segmento, os dados são adicionado ao *buffer* para reprodução do vídeo.

Dado esse processo mais complexo e considerando que a largura de banda de um cliente é geralmente menor que a de um nó CDN, recuperação de um segmento pela rede *Peer-to-Peer* é mais lenta que buscar de uma CDN, porém, apesar de ser mais lenta, ela deve ser maximizada para diminuir a carga no servidor. Para isso, algumas condições referentes ao estado do cliente precisam ser satisfeitas para que uma recuperação via *Peer-to-Peer* não resulte em *buferring* para o cliente (parada na reprodução, pois o *buffer* foi esgotado), as condições são as seguintes:

- Estar conectado a rede *Peer-to-Peer*;
- O segmento a ser recuperação não deve ser o segmento inicial para reprodução do vídeo;
- O tamanho do *buffer* de reprodução não deve ser muito pequeno, pois recuperar da rede *Peer-to-Peer* consome um tempo maior;

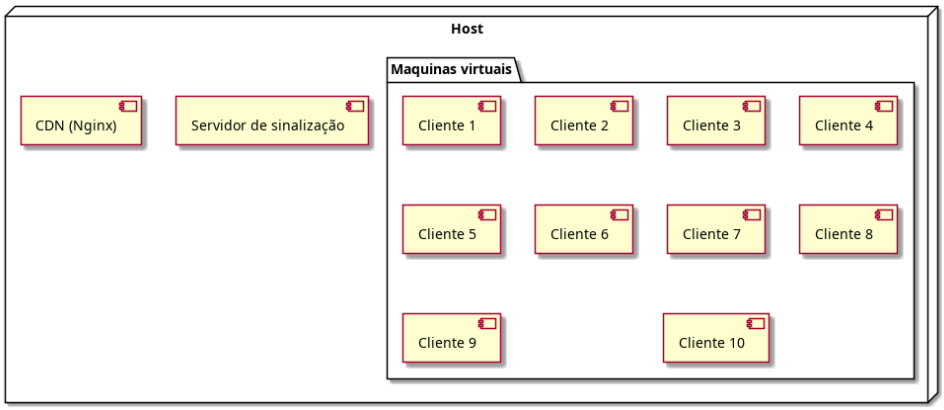
Além disso, é necessário que seja estabelecido um tempo máximo de espera ao disparar uma recuperação na rede *Peer-to-Peer*, pois é possível que não tenha nenhum *peer* com o

segmento e é possível que o *peer*, mesmo possuindo o segmento, tenha uma largura de banda muito baixa para *upload*, fazendo com que a transferência do dado seja muito lenta. Nos casos que o tempo limite é atingido, é disparado a recuperação pela CDN para o segmento que falhou, sendo assim, aumentando a fluidez do sistema para o usuário final.

4.4 AVALIAÇÃO

Para avaliar o sistema proposto foi conduzido um teste com o objetivo de monitorar o número de requisições feitas ao CDN pelo sistema. O cenário de teste é constituido por 10 clientes, onde cada cliente é representado por uma maquina virtual, com um *thread* e 1024Mb de ram, rodando o sistema operacional NixOs e utilizando o navegador Google Chrome versão V88, o CDN foi representado por um servidor Web Nginx que ficou encarregado de distribuir a stream utilizando o protocolo DASH (Dynamic Adaptive Streaming over HTTP), e a página Web que será acessada pelo cliente para reproduzir o vídeo, esse cenário é ilustrado na figura 16.

Figura 16 – Cenário de teste



Fonte: Acervo pessoal

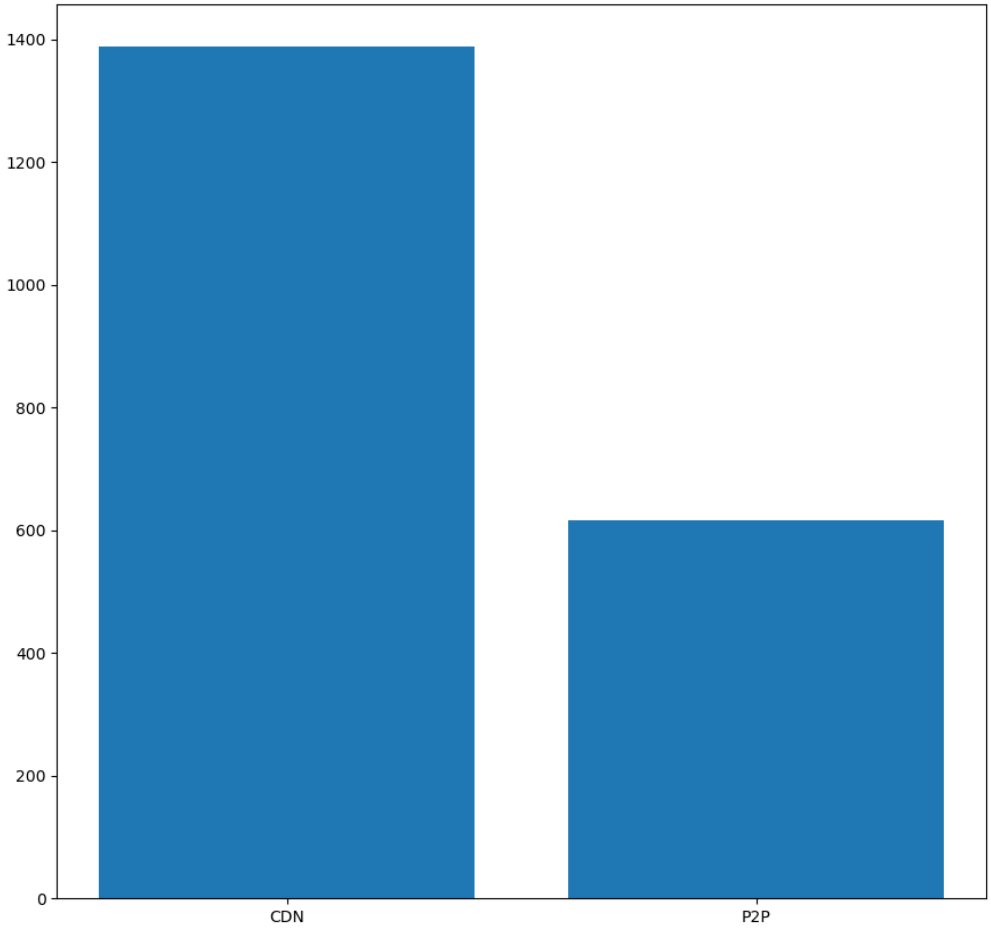
Para a apuração dos resultados, foi coletado metricas nos clientes após cada requisição, onde essa metrica é enviada para o servidor de sinalização, para que o mesmo armazene em um banco relacional, assim, facilitando a consulta e avaliação dos dados coletados durante o teste.

4.4.1 Resultados

O teste foi executado por 10 minutos, onde durante esse periodo, 5 dos cliente foram adicionados em um intervalo de 1 segundo, e outros 5 foram adicionados no segundo subse-

quente. Nesse intervalo, foi recuperado um total de 2189 segmentos, como ilustra a figura 17, 1481 recuperações pelo feita pelo CDN, ou seja, 67.65% do total, e 32.35% foram feitas pela rede P2P.

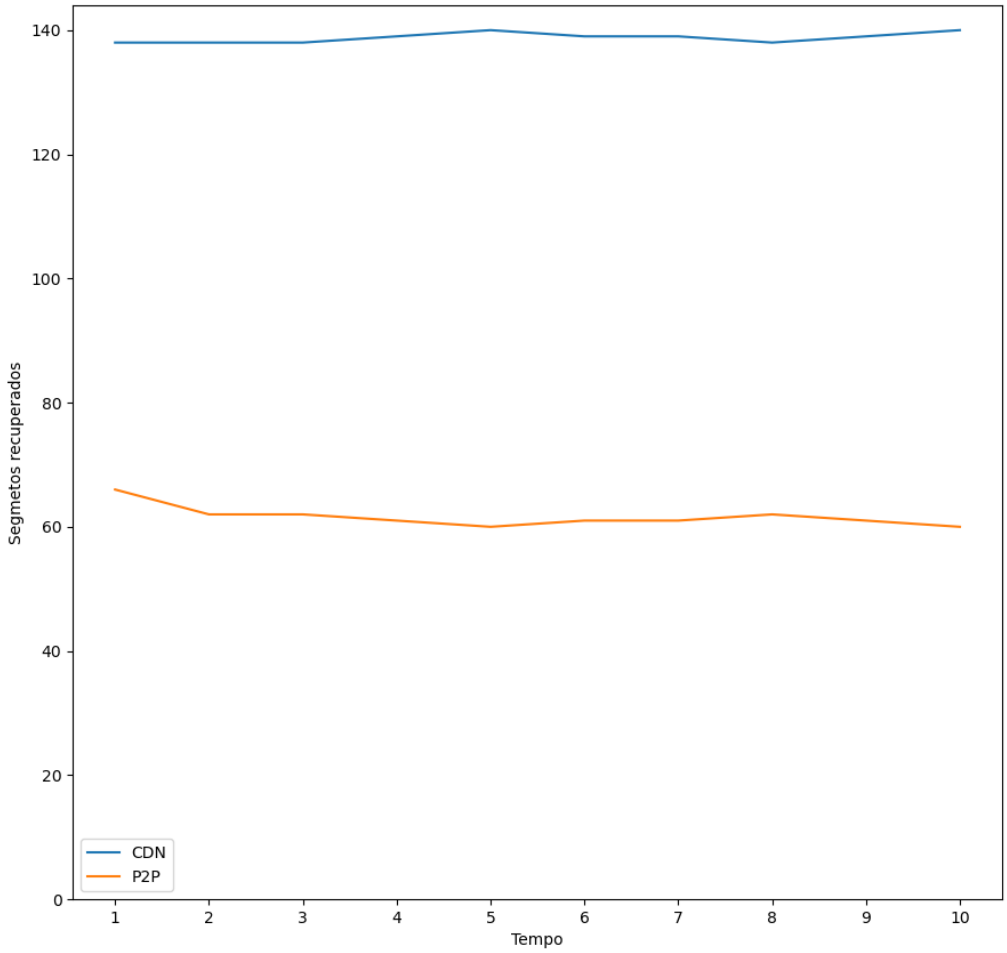
Figura 17 – Recuperações efetuas durante o teste



Fonte: Acervo pessoal

A figura 18 mostra o número de recuperações feitas pelo CDN e P2P dado intervalos de 1 minuto, onde pode-se observar o aumento de recuperações pela rede *peer-to-peer* no primeiro minuto devido à discrepância entre o intervalo de tempo onde o segundo lote de clientes foi adicionado.

Figura 18 – Recuperações efetuas durante o teste por tempo



Fonte: Acervo pessoal

Os resultados mostram que a adição da camada *peer-to-peer* resultou de 32% do total de requisições feitas ao CDN, reduzindo significativamente a carga imposta nos mesmas, garantindo assim, um aumento da escalabilidade utilizando os recursos providos pelos usuários finais.

5 CONSIDERAÇÕES FINAIS

A partir dos testes e resultados apresentados neste trabalho e em relacionados, pode-se observar que houve reduções significativas no número de requisições feitas para a CDN, mostrando é viável servir *streams* de vídeo via um modelo híbrido CDN-P2P.

Com a redução da quantidade de requsições feitas a CDN, faz com que o número necessário sevidores em um CDN para atender um numero x de clientes seja reduzido, assim, diminuindo o custo de mantimento de uma CDN. Junto a isso, a adicação de uma camada peer-to-peer garante um aumento da escalibilidade, pois, com ela, quanto maior o número de clientes a ser atendidos pela CDN, maior será o número de servidores que a mesma terá, pois cada cliente atuará, ao mesmo tempo, como cliente e sevidor.

O sistema proposto mostra que servir vídeo de uma maneira híbrida CDN-P2P pode ser feito sem muita complicação, tanto para quem está implementando, granças a WebRTC, quanto a quem usará o sistema em sua aplicação, graças ao uso da API *SourceMedia*, qualquer *stream* DASH pode ser servida por uma rede *peer-to-peer* e ser reproduzida pelo *player* nativo HTML5 ou qualquer *player* que suporte a API *SourceMedia*, simplesmente com uma chamada de método da biblioteca.

O sistema proposto é uma implementação de prova de conceito e não está pronto para uso comercial, pois não trata de questões de segurança e monitoramento. O código do sistema proposto pode ser acessado nos seguintes links: Servidor de Sinalização e Biblioteca Cliente

5.1 TRABALHOS FUTUROS

Para trabalhos futuros é desejável expandir a aplicação do sistema proposto, de maneira que seja extraído o sistema de compartilhamento de dados na rede P2P em uma biblioteca separada, fazendo que outros sistemas com usos diferentes, como compartilhar arquivos possam ser construídos em cima da mesma.

REFERÊNCIAS

BARBOSA, F. R. N.; SOARES, L. F. G. Towards the application of WebRTC peer-to-peer to scale live video streaming over the internet. **Simpósio Brasileiro de Redes de Computadores (SBRC), no. Figure**, [S.l.], v.1, p.119–124, 2014.

BERGKVIST, A. et al. Webrtc 1.0: real-time communication between browsers. **Working draft, W3C**, [S.l.], v.91, 2012.

BIERMAN, G.; ABADI, M.; TORGERSEN, M. Understanding typescript. In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING. **Anais...** [S.l.: s.n.], 2014. p.257–281.

CASE Study: america movil and viavi. Acesso em: 12-07-2020, https://www.gsma.com/futurenetworks/wiki/case-study-america-movil-and-viavi.

CASTRO, M. et al. SplitStream: high-bandwidth multicast in cooperative environments. **SIGOPS Oper. Syst. Rev.**, New York, NY, USA, v.37, n.5, p.298–313, Oct. 2003.

FETTE, I.; MELNIKOV, A. **The WebSocket Protocol**. [S.l.]: RFC Editor, 2011. RFC, http://www.rfc-editor.org/rfc/rfc6455.txt. (6455).

HALLOWAY, S. **Programming Clojure**. [S.l.]: Pragmatic Bookshelf, 2009.

HUANG, C. et al. Understanding Hybrid CDN-P2P: why limelight needs its own red swoosh. In: INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, 18., New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2008. p.75–80. (NOSSDAV '08).

KAY, A. C. **The Reactive Engine**. 1969. Tese (Doutorado em Ciência da Computação) — . AAI7003806.

KERANEN, A.; HOLMBERG, C.; ROSENBERG, J. **Interactive Connectivity Establishment (ICE)**: a protocol for network address translator (nat) traversal. [S.l.]: RFC Editor, 2018. RFC. (8445).

KUROSE, J. F.; ROSS, K. W. **Computer Networking**: a top-down approach (6th edition). 6th.ed. [S.l.]: Pearson, 2012.

LORETO, S. **Real-Time Communication with WebRtc**: peer-to-peer in the browser. [S.l.]: O'Reilly Media, 2014.

LORETO, S.; ROMANO, S. P. **Real-time communication with WebRTC**: peer-to-peer in the browser. [S.l.]: "O'Reilly Media, Inc.", 2014.

LU, Z. et al. An Analysis and Comparison of CDN-P2P-hybrid Content Delivery System and Model. **JCM**, [S.l.], v.7, n.3, p.232–245, 2012.

MAHY, R.; MATTHEWS, P.; ROSENBERG, J. **Traversal Using Relays around NAT (TURN)**: relay extensions to session traversal utilities for nat (stun). [S.l.]: RFC Editor, 2010. RFC. (5766).

PADMANABHAN, V. N. et al. Distributing Streaming Media Content Using Cooperative Networking. In: INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEMS SUPPORT FOR DIGITAL AUDIO AND VIDEO, 12., New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2002. p.177–186. (NOSSDAV '02).

PADMANABHAN, V. N.; SRIPANIDKULCHAI, K. The Case for Cooperative Networking. In: REVISED PAPERS FROM THE FIRST INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS, Berlin, Heidelberg. **Anais...** Springer-Verlag, 2002. p.178–190. (IPTPS '01).

PANTOS, R.; MAY, W. **HTTP Live Streaming**. [S.l.]: RFC Editor, 2017. n.8216. (Request for Comments).

PENG, G. CDN: content distribution network. **CoRR**, [S.l.], v.cs.NI/0411069, 2004.

RESCORLA, E.; MODADUGU, N. **Datagram Transport Layer Security Version 1.2**. [S.l.]: RFC Editor, 2012. RFC. (6347).

ROSENBERG, J. et al. **Session Traversal Utilities for NAT (STUN)**. [S.l.]: RFC Editor, 2008. RFC. (5389).

SHARMA, M. et al. Digital Burnout: covid-19 lockdown mediates excessive technology use stress. **World Social Psychiatry**, [S.l.], v.2, n.2, p.171–172, 2020.

STOCKHAMMER, T. Dynamic Adaptive Streaming over HTTP –: standards and design principles. In: SECOND ANNUAL ACM CONFERENCE ON MULTIMEDIA SYSTEMS,

New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2011. p.133–144. (MMSys '11).

TANENBAUM, A. S.; WETHERALL, D. J. **Computer Networks**. 5th.ed. USA: Prentice Hall Press, 2010.

WEBRTC Signaling Servers: everything you need to know | wowza. Acesso em: 12-01-2021, https://www.wowza.com/blog/webrtc-signaling-servers.

YIN, H. et al. Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: experiences with livesky. In: ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA, 17., New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2009. p.25–34. (MM '09).