KEVIN OURY

Aplicação de Web Streaming Híbrido CDN-P2P usando WebRTC

São Paulo
2016

KEVIN OURY

Aplicação de Web Streaming Híbrido CDN-P2P usando WebRTC

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção da graduação

Área de concentração: Engenharia da Computação

Orientador: Profa. Regina Melo Silveira
Co-orientador: MSc. Samuel Kopp

São Paulo
2016

KEVIN OURY 9239677 – S15

Aplicação de Web Streaming Híbrido CDN-P2P usando WebRTC

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção da graduação

Área de concentração: Engenharia da Computação

Orientador: Profa. Regina Melo Silveira
Co-orientador: MSc. Samuel Kopp

São Paulo
2016

FICHA CATALOGRÁFICA

Oury, Kevin
Aplicação de Web Streaming Híbrido CDN-P2P usando WebRTC / Kevin Oury - São Paulo, 2016

Trabalho de Conclusão de Curso – Escola Politécnica da Universidade de São Paulo, Departamento de Engenharia Elétrica, Sistemas Digitais e Computação.

1. Engenharia 2. Engenharia da Computação 3. Redes de Computadores 4. Curso de Graduação I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Elétrica, Sistemas Digitais e Computação.

Agradecimentos

A professora Regina Melo Silveira e MSc. Samuel Kopp, pela orientação e pelo constante estímulo transmitido durante todo o trabalho.

A todos que colaboraram direta ou indiretamente na execução deste trabalho.

Resumo

O que se desejou investigar é uma alternativa aos modelos clássicos para entrega de vídeo no contexto de Web Streaming. Pretendeu-se descobrir um novo modelo que permitisse ter uma melhor escalabilidade que sistemas cliente-servidor baseados apenas em Content Delivery Networks. O presente trabalho visa propor uma arquitetura de distribuição de conteúdo híbrida usando Peer-to-Peer e demostrar sua viabilidade com um protótipo.

Numa primeira fase, foram estudados os diferentes elementos técnicos envolvidos em um Web Streaming, da parte de infraestrutura até a parte de aplicação. Em seguida, foi descrita a proposta de arquitetura híbrida. Enfim, foram apresentados os resultados do protótipo e medidos os impactos possíveis.

Palavras-chave: Engenharia da computação. Redes. Peer-to-Peer. WebRTC.

Abstract

The subject of the present work is to find an alternative to classic models for video delivery in the context of Web Streaming, to achieve better scalability than client-server architectures based on Content Delivery Networks. This work proposes a hybrid architecture for media delivery using Peer-to-Peer, illustrated with a prototype.

First, this document presents an overview of the components of a Web Streaming system, from infrastructure elements to application elements. Then, the formal proposal for the hybrid infrastructure is explained. Finally, results obtained with the prototype are analyzed.

Key words: Engineering. Networks. Peer-to-Peer. WebRTC.

Lista de Ilustrações

Lista de abreviaturas e siglas

API	Application Programming Interface
CDN	Content Delivery Network
DDoS	Distributed Denial of Service
DNS	Domain Name Service
HLS	HTTP Live Streaming
HTTP	HyperText Transfer Protocol
ICE	Interactive Connectivity Establishment
IPTV	Internet Protocol Television
NAT	Network Address Translator
P2P	Peer To Peer
SA	Sistema Autônomo
SDP	Session Description Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UHD	Ultra High Definition
VOD	Video On Demand
WebRTC	Web Real-Time Communication

Sumário

I. Introdução

Nos últimos anos o mercado de vídeo na Web vem experimentando um crescimento exponencial devido aos efeitos combinados da alta do consumo e da qualidade dos vídeos. O tráfego de vídeo na Internet vai ser multiplicado por 4 entre 2015 e 2020 pois, de acordo com as previsões da Cisco, o volume de fluxo de vídeo constituirá 79% do tráfego total em 2020, contra 63% em 2015 [1].

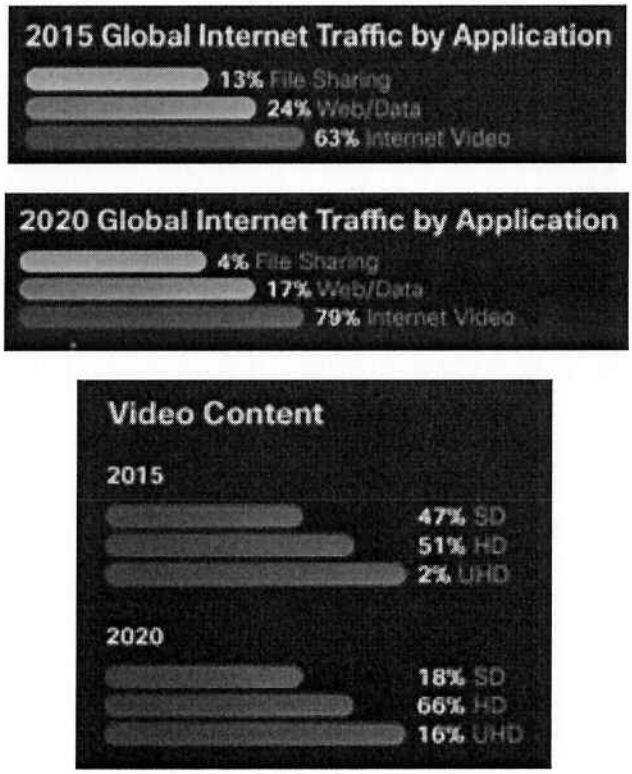

Fig. 1 - Evolução do tráfego Internet 2015-2020 [1]

Ao mesmo tempo, o tamanho dos arquivos de vídeo está aumentando também. Em um mercado onde qualidade é fundamental, *Ultra-High Definition* (UHD)[1] rapidamente virou necessária. Com usuários esperando melhores qualidades, distribuidores de vídeos precisarão de mais banda e de infraestruturas que possam receber arquivos maiores em picos de tráfego para atender as expectativas dos usuários finais. UHD representará 15,7% do tráfego total de vídeo na Internet, contra 2,3% em 2015 [1].

Os modelos de entrega de vídeo atuais usando protocolos unicast[2] já estão mostrando os seus limites. Conforme escreveu Kurt Michel, da Akamai: "O aumento da penetração da banda larga e os dispositivos mais rápidos fazem com que as experiências de visualização de alta qualidade sejam possíveis, mas problemas de latência de rede e

[1] Formato de vídeo digital de ultra-alta definição.

[2] Endereçamento de pacote IP para um único destino, ponto-a-ponto.

altas cargas de tráfego podem muitas vezes resultar em um desempenho de streaming de vídeo desapontador, com pausas frequentes para bufferização" [2].

Protocolos tradicionais baseados em relações um para um entre cliente e servidor têm provado serem insuficientes para atender a demanda atual, com sobrecarga e congestionamento de rede frequentes. As CDN (Content Delivery Networks), que serão detalhadas em sessão posterior, são infraestruturas criadas com a intensão de melhorar esta situação, multiplicando o número de servidores espalhados geograficamente. Porém, essa solução não é escalável e pode acabar falhando em prover confiabilidade a um custo aceitável [3].

Uma outra questão relevante é o uso dos recursos disponíveis. Admitindo que as infraestruturas de CDN possam aliviar as redes dos provedores e ajudar a entregar conteúdo de forma mais rápida, essas infraestruturas são compartilhadas entre vários clientes. Se um cliente precisar transmitir um evento que gere um grande consumo de recursos, será que os outros clientes dessa mesma CDN serão prejudicados no atendimento do serviço?

A questão da largura de banda utilizada é um dos maiores desafios para as CDN e os ISP (Internet Service Provider - Fornecedor de Acesso à Internet). Desta maneira, os serviços de transmissão estão sempre submetidos ao poder do provedor da infraestrutura, como já observamos conflitos entre Netflix e Verizon por exemplo [4].

De forma mais ampla, a questão da alocação de banda entre diferentes serviços da Internet foi amplamente discutida na esfera política, quer seja nos Estados Unidos com as discussões sobre a *Net Neutrality* (ou *Neutralidade da Internet*), ou no próprio Brasil com o *Marco Civil da Internet*. Essas leis garantem a neutralidade da rede: *"O responsável pela transmissão, comutação ou roteamento tem o dever de tratar de forma isonômica quaisquer pacotes de dados, sem distinção por conteúdo, origem e destino, serviço, terminal ou aplicação"* [5].

II. Objetivo e Motivação

As questões mencionadas na introdução estão na origem da proposta deste Trabalho de Conclusão de Curso, que pretende oferecer pistas de pesquisa através da presente monografia e do protótipo técnico acompanhando ela. O objetivo a longo prazo deste tipo de projeto é de cortar os custos de infraestrutura e conseguir responder a uma demanda em crescimento exponencial, guardando em mente a ideia original da Internet que era uma ideia de uma rede decentralizada e aberta a todos, de forma igual.

De maneira mais concreta , a proposta deste trabalho de conclusão de curso é de desenvolver um protótipo de sistema de ajuda ao web streaming de vídeo via peer-to-peer (P2P), integrado a CDN como mecanismo complementar de entrega. O objetivo deste sistema é duplo:

- economizar recursos de rede (banda e processamento de servidores),
- melhorar a Qualidade de Experiência para o usuário final (maior qualidade de vídeo, menos rebufferização[3], melhor desempenho à altas cargas em grandes eventos, tolerância a falha de um servidor)

De um ponto de vista mais pessoal, foi escolhido este assunto por afinidade do autor pelos assuntos de vídeo na Internet, peer-to-peer e, de forma mais geral, a pesquisa sobre as novas organizações de rede que vão responder aos numerosos desafios presentes e dos próximos anos. Na mesma linha, o autor trabalhou no domínio de Web Streaming e peer-to-peer e desenvolveu uma base de conhecimento sobre esses assuntos. Este estágio criou uma vontade de continuar a trabalhar com esse assunto e o trabalho de conclusão de curso era uma ótima oportunidade para aproveitar os recursos tanto de conhecimento e experiência que materiais do Laboratório de Arquitetura e Redes de Computadores da USP (LARC-USP). O LARC participou do desenvolvimento da infraestrutura do IPTV-USP, e demostrou um interesse em integrar as tecnologias de peer-to-peer na sua arquitetura. Este assunto será aprofundado na parte dedicada do presente documento.

A escolha deste projeto como trabalho de conclusão de curso permitiu conduzir pesquisas aprofundadas no assunto e melhorar habilidades em desenvolvimento orientado para mídia, nas atividades de escolha das tecnologias adaptadas, reuso de blocos tecnológicos existentes e desenvolvimento de blocos próprios. O projeto usa tecnologias como Web Streaming e WebRTC, detalhadas na seção seguinte.

[3] O fenômeno de "rebufferização" corresponde a uma pausa na reprodução ("playback") por causa de falta de segmentos de vídeo dentro do buffer ("buffer underflow"). Precisa parar a reprodução para esperar o carregamento de mais segmentos de vídeo e poder voltar à leitura normal. Este fenômeno é uma das principais reclamações de usuários quando fizer streaming a partir de um servidor carregado.

III. Conceitos e definições

Esta seção discorre sobre os conceitos tecnológicos utilizados neste trabalho, assim como as definições utilizadas para a elaboração do mesmo

III.1. Web Streaming

a) Introdução

Podemos dividir as tecnologias de Web Streaming de vídeo (ou transmissão de vídeo via web) em duas categorias: a *Internet Video*, conhecida como *Over The Top* (OTT), e *Serviços Autônomos* como IPTV ou TV a cabo.

Os serviços como IPTV ou TV a cabo utilizam uma rede fechada (um "sistema autônomo") para transmitir porque usam transporte multicast[4] e precisam atender alguns requisitos de qualidade de serviço (QoS) [6].

Ao contrário, os serviços de OTT transmitem via Internet pública, com tecnologias tradicionais como Microsoft Windows Media, Apple Quicktime, Adobe Flash, e as mais atuais tecnologias de streaming adaptativo como *Apple HTTP Live Streaming* (HLS), *Microsoft Smooth Streaming* (HSS) e *Dynamic Adaptive Streaming over HTTP* (DASH). Essas tecnologias transmitem o conteúdo para o usuário através de uma conexão unicast (a partir de um servidor origem ou de uma CDN) ou de um protocolo proprietário em cima dos protocolos de transporte TCP e as vezes UDP (mais voltado para streaming ao vivo), ou de HTTP [7].

Tradicionalmente, a transmissão era feita por *Progressive Download*, que usa HTTP, por causa da simplicidade. O vídeo é transmitido por um servidor web HTTP básico, armazenado no disco do usuário, e reproduzido a partir do disco. Além da simplicidade, outra vantagem é que pode codificar o vídeo com taxas maiores que a banda efetivamente disponível para o usuário, porque o vídeo será reproduzido de maneira regular e sem pausas, já que é armazenado localmente.

Porém, uma vez que o vídeo está armazenado no disco do usuário, é muito mais fácil de ser copiado. Esse é um motivo pelo qual muitos produtores transmitem seus conteúdos com *Streaming*.

No caso de *Streaming*, é necessário usar um servidor de streaming, que é um tipo de servidor particular encarregado especialmente para fazer streaming de mídia, ao contrário de um servidor web básico. Mas neste caso, o vídeo tem que ser codificado em uma taxa bastante inferior à banda média disponível para o usuário alvo. Se não for, a reprodução vai ter paradas frequentes, diminuindo a qualidade de experiência do usuário.

Enfim, uma terceira opção que é mais usada hoje em dia é o *Adapative Streaming*, que codifica fluxos sob demanda ou ao vivo em várias taxas, e troca de taxa (ou qualidade) de maneira dinâmica e adaptativa em função da banda disponível para o

[4] Endereçamento que permite a entrega de uma mesma informação para vários destinatários, usando apenas um link comum até ele se dividir para chegar nos pontos desejados.

usuário e outras variáveis de estado da rede. Isso garante a melhor qualidade possível para quem tiver banda suficiente, e uma reprodução sem paradas com qualidade menor para quem estiver usando uma rede celular por exemplo [8]. Descrevemos alguns exemplos de protocolos de *Adaptive Streaming*.

b) Formatos de streaming

A transmissão de conteúdo entre nós de uma rede pode ser feita de várias maneiras, dependendo dos objetivos em termos de confiabilidade e atraso. Podemos dividir essas maneiras entre protocolos *push-based* e protocolos *pull-based*.

Com protocolos *push-based*, uma vez que o servidor e o cliente estabeleceram uma conexão, o servidor transmite um fluxo até o cliente até ele interromper a conexão, através do controle de sessão. O protocolo *Real-time Streaming Protocol* (RTSP, RFC 2326) é bastante usado para o controle de sessão, enquanto a transmissão de dados geralmente é feita com o *Real-time Transport Protocol* (RTP, RFC 3550), em cima do protocolo UDP.

Os protocolos *push-based* são adaptados para fazer multicast por exemplo.

Por outro lado, no caso dos protocolos *pull-based*, o cliente é a entidade ativa e é ele que pede o conteúdo para o servidor. Esses protocolos geralmente usam HTTP.

Os protocolos de *Adaptive Streaming* são *pull-based*. Os mais usados hoje em dia são o *HTTP Live Streaming Protocol* (HLS) da Apple, o *Dynamic Adaptive Streaming over HTTP* (MPEG-DASH) e o *Smooth Streaming da Microsoft*. HLS e Smooth são protocolos proprietários, enquanto DASH foi elaborado pelo MPEG e padronizado pela ISO [9].

i) HLS

HLS é o protocolo de streaming adaptativo para vídeo sob demanda e ao vivo da Apple, e é o único compatível com as aplicações de streaming da plataforma iOS (iPhone, iPad). De fato, ele deve ser obrigatoriamente considerado para quem quiser atingir usuários desta plataforma.

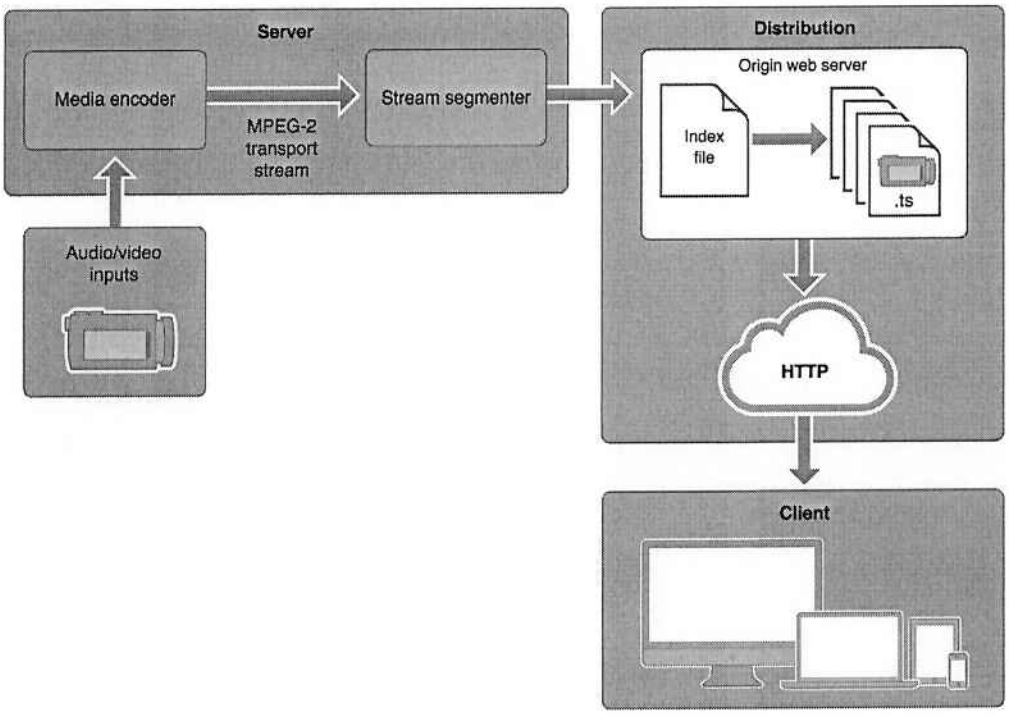

Fig. 2 – Visão geral do streaming de um fluxo HLS [10]

O seu funcionamento, descrito na figura Fig. 2, é similar aos outros formatos de streaming adaptativo. É preciso codificar o conteúdo em várias qualidades diferentes (várias taxas) que ficarão em arquivos separados, entre as quais o *player* poderá alternar dinamicamente para otimizar a experiência do lado cliente. Cada qualidade é dividida em segmentos que representam entre 5 e 10 segundos de vídeo. A entrega é feita a partir de um servidor HTTP (e não um servidor de streaming) que serve os segmentos e os arquivos de texto que são chamados "manifests" ou playlists. A primeira playlist, a *Master* Playlist, (extensão .m3u8) contem a lista de todas as qualidades disponíveis para aquele fluxo (Fig. 3). A playlist de segundo nível é dedicada para cada qualidade diferente, e contem a lista de segmentos com os caminhos relativos onde eles podem ser encontrados no formato MPEG-2 TS (.ts).

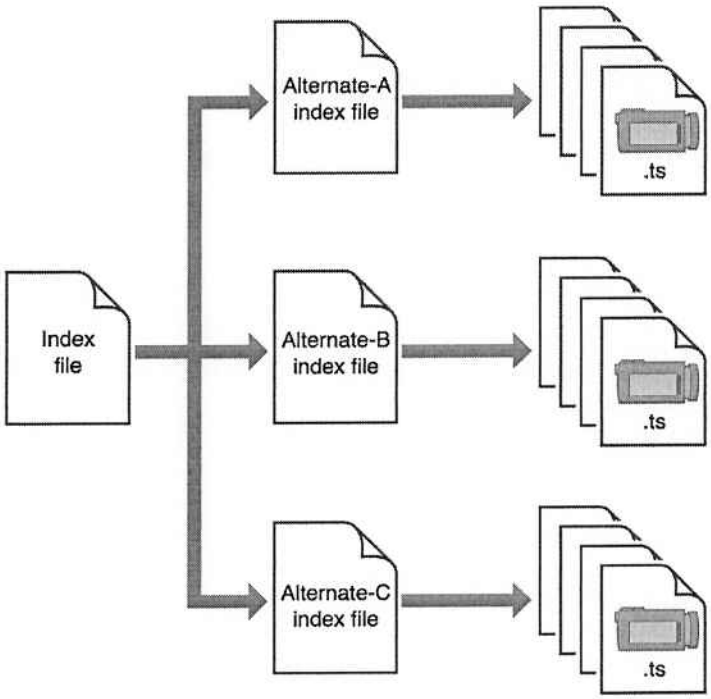

Fig. 3 – Hierarquia de playlists .m3u8 [10]

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=150000,RESOLUTION=416x234, \
CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/low/index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=240000,RESOLUTION=416x234, \
CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/lo_mid/index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=440000,RESOLUTION=416x234, \
CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/hi_mid/index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=640000,RESOLUTION=640x360, \
CODECS="avc1.42e00a,mp4a.40.2"
http://example.com/high/index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=64000,CODECS="mp4a.40.5"
http://example.com/audio/index.m3u8
```

Fig. 4 – Exemplo de playlist de primeiro nível [11]

A figura Fig. 4 representa um exemplo de playlist de primeiro nível. Cada linha representa uma qualidade diferente do mesmo conteúdo, com o link para a playlist de segundo nível correspondente.

A figura Fig. 5 representa uma playlist de segundo nível. Cada linha representa um segmento no formato MPEG-2 TS, com a sua duração em segundos e o link para ser baixado.

```
#EXTM3U
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
http://example.com/movie1/fileSequenceA.ts
#EXTINF:10.0,
http://example.com/movie1/fileSequenceB.ts
#EXTINF:10.0,
http://example.com/movie1/fileSequenceC.ts
#EXTINF:9.0,
http://example.com/movie1/fileSequenceD.ts
#EXT-X-ENDLIST
```

Fig. 5 – Exemplo de playlist de segundo nível [11]

O vídeo é codificado com o codec H.264 e o áudio com AAC. Os segmentos individuais estão no formato MPEG-2 Transport Stream. Deve-se notar que vídeo e áudio são multiplexados, ao contrário do caso do MPEG-DASH por exemplo.

ii) DASH

MPEG-DASH é o padrão ISO (ISO/IEC 23009-1) de *Streaming Adaptativo* que tem potencial para substituir padrões proprietários como HLS ou Smooth Streaming, com o suporte de empresas como Apple, Netflix, Microsoft, etc. A vantagem de ter um único padrão unificado seria que os publicadores poderiam gerar apenas um conjunto de arquivos a serem distribuídos em todas as plataformas [**12**]. O primeiro obstáculo é o fato que a Apple continua impondo HLS para iOS. O segundo problema é que, sendo compatível com qualquer codec[5], o formato DASH não resolve os problemas de

[5] Codificador/decodificador de sinal.

compatibilidade de codec entre navegadores. Isso significa que os publicadores ainda teriam que codificar os vídeos de várias formas para garantir uma compatibilidade com a maior parte dos navegadores.

Do ponto de vista técnico, DASH é parecido com HLS. Ele é constituído por um arquivo manifest "Media Presentation Description" (parecido com "Playlist") em XML, com as diferentes qualidades de vídeo, e pelos arquivos por si mesmo. Ao contrario de HLS, áudio e vídeo podem ser separados em arquivos diferentes (facilidade para trocar de pista de língua sem baixar de novo as imagens por exemplo). O vídeo é dividido em segmentos.

III.2. IPTV

a) Conceito

Internet Protocol Television (IPTV) é um sistema que fornece serviços de televisão via Internet usando uma arquitetura e protocolos de redes do conjunto do Internet Protocol (IP). Serviços de IPTV podem ser classificados em três grupos principais: televisão ao vivo, programada, e vídeo sob demanda.

A ideia original com IPTV para os ISP era de fornecer os serviços de televisão, vídeo sob demanda (VOD), etc. usando Internet, em uma infraestrutura contida que permitisse controlar a qualidade de serviço, gerando oportunidades de renda. Esses serviços de IPTV costumam ser oferecidos pelos ISP em ofertas *"Triple Play"*, junto com acesso a Internet e *Voice Over IP* (VoIP). A instalação pode ser feita através de uma caixa ou *"Box"* para utilizar os serviços em uma televisão, ou mais simplesmente os serviços podem ser acessíveis via um computador [**13**].

b) Mercado

De acordo com a Point Topic [**14**], em Março de 2013 tinham 79.3 milhões de subscrições de IPTV no mundo (Fig. 6). Pode-se observar na mesma figura que o crescimento deste número diminuiu a partir do fim de 2011 (7%), para chegar perto de 4% em 2013.

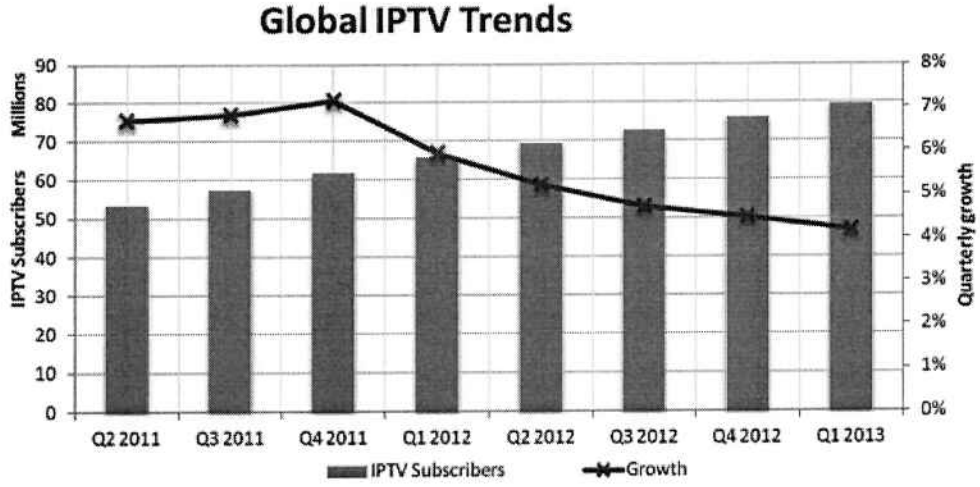

Fig. 6 – Subscrições IPTV no mundo (Q1 2013) [14]

Em termos de regiões, os maiores países para IPTV são China, França e EUA (Fig. 7). No caso da França, que foi um dos primeiros países a adotar a tecnologia em grande escala, isso representa quase 25% da população.

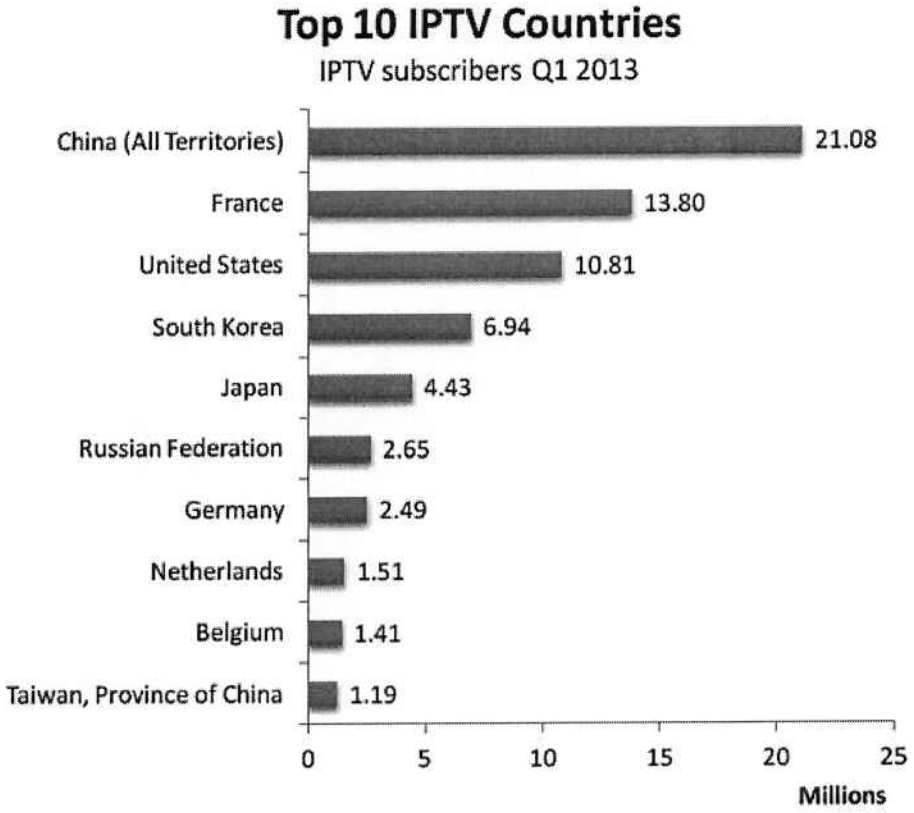

Fig. 7 – Top países para IPTV (Q1 2013) [14]

Quando olhar no detalhe das novas subscrições de janeiro até março de 2013, são países asiáticos nos primeiros lugares (China, Coréia, Japão, Viet Nam).

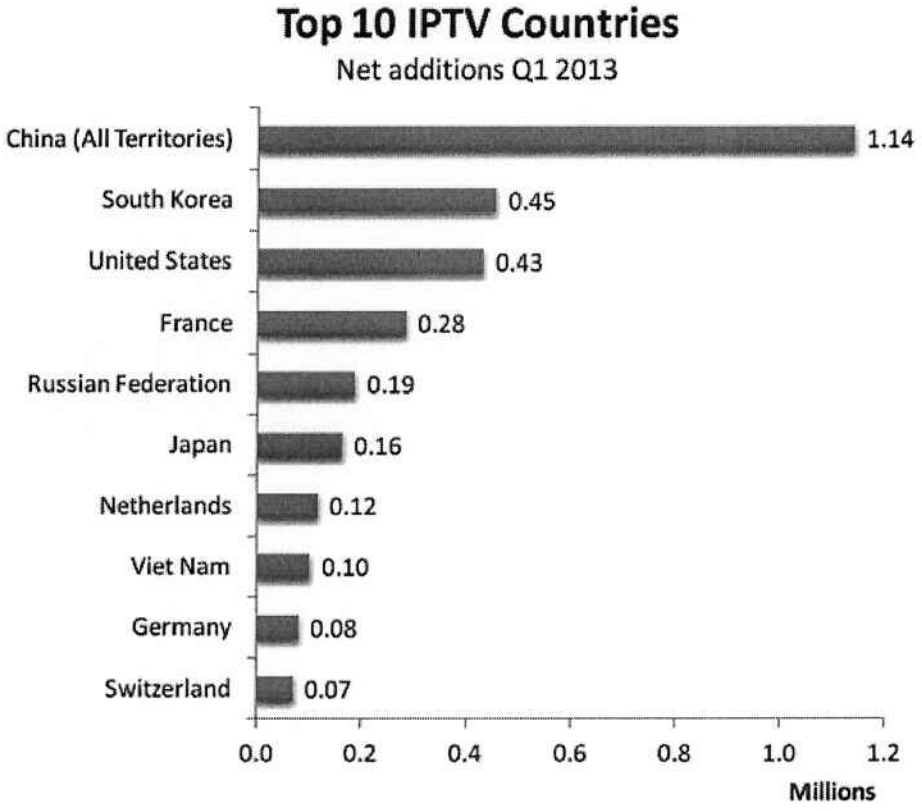

Fig. 8 – Crescimento de IPTV por país (Q1 2013) [14]

c) Exemplo de arquitetura

A arquitetura típica de uma IPTV está descrita na Fig. **9**. A informação broadcast que chega por uma antena satélite no "Super Head-end" é distribuída principalmente em MPEG-2 Multi-Program Transport Stream (MPTS) até o nó de serviço de vídeo. A distribuição do próprio conteúdo do canal é feita através de vários equipamentos na rede de acesso, como *Digital Subscriber Line Access Multiplexers* (DSLAM) e outras tecnologias como *Fiber-to-the-Home* (FTTH) que fazem a interface com a « *Box* » do usuário. Para IPTV, cada canal é distribuído usando um IP multicast.

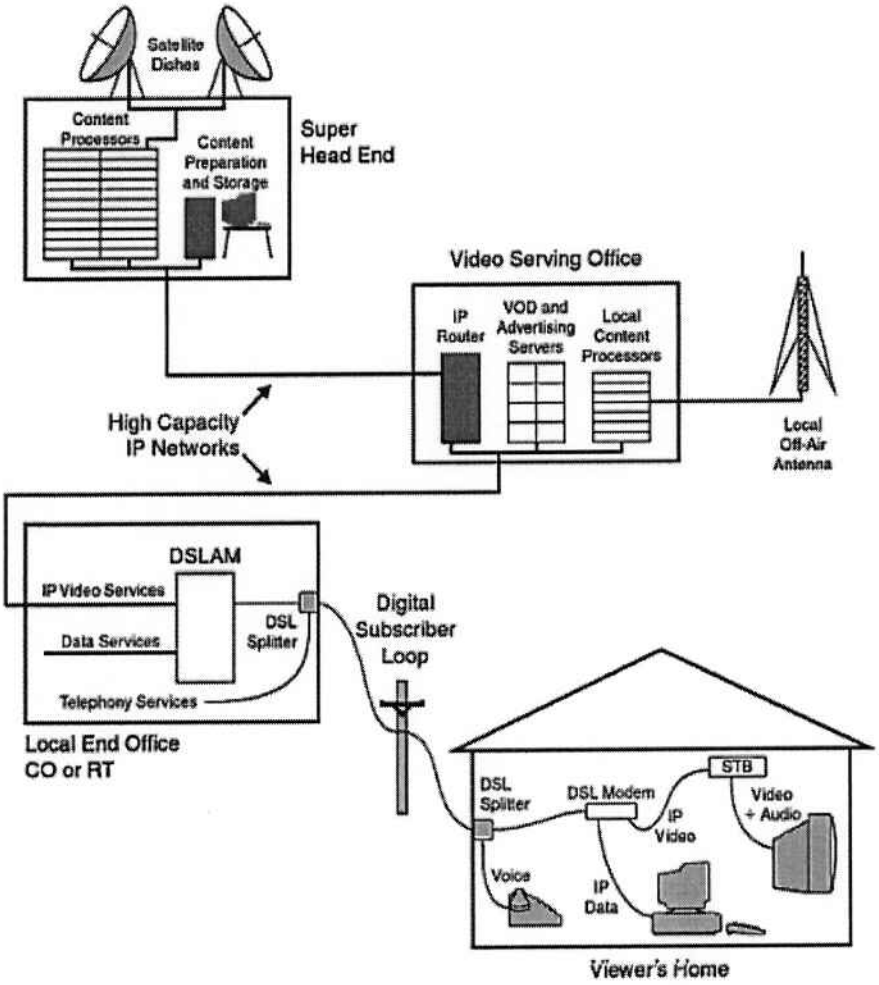

Fig. 9 – Arquitetura típica de IPTV [13]

III.3. Web Player

Em um site de mídia com vídeo, o web player é o elemento de software que mostra o vídeo para o usuário. O player toma como entrada o arquivo de vídeo (.mp4 por exemplo), decodifica este arquivo para transformá-lo em uma lista de quadros (imagens) e faz a reprodução desses quadros de maneira animada na página web desejada.

Quando escolher um player, é muito importante levar em conta vários aspectos, como formatos de vídeo (progressive download, streaming adaptativo, etc.) e plataformas compatíveis (OS, navegador, mobile, etc.). Existem muitos players disponíveis no mercado, com modelos de licença diferentes. Podemos citar Video.js da Brightcove, JWPlayer, o player da Kaltura, o player da Ooyala, e o Primetime player da Adobe [**15**]. Quando eles forem adaptados as suas necessidades, é recomendado usar um desses players já disponíveis. Foi o que foi feito neste projeto, usando o video.js.

Porém, em certos casos pode ser interessante construir o seu próprio player para satisfazer requisitos particulares.

Distinguimos 2 grandes tecnologias de players para Web: players baseados em Flash e players baseados em HTML 5.

A tecnologia Flash é a mais madura das duas e o legado dos web players está em Flash. Com a aparição do tag <video> no HTML 5, Flash está sendo substituído por players em HTML 5. Isso também é devido às várias falhas de segurança descobertas a cada semana no Flash, bem como à vontade da Apple de banir o suporte a Flash nos aparelhos iOS.

Olhando para o requisitos da nossa aplicação P2P, hoje em dia Google Chrome e Mozilla Firefox implementaram *Media Source Extensions* (MSE), que é a parte de HTML 5 que permite manipular os arquivos de vídeo diretamente com JavaScript no navegador. Deste ponto de vista também, isso significa que não é mais necessário usar Flash para aplicações P2P.

Por outro lado, o uso de HTML 5 nos players web está sempre em crescimento. Podemos citar por exemplo a Youtube que passou de um player em Flash a um player em HTML 5 por padrão em 2015 [**16**]. Em termos de compatibilidade, HTML 5 tem uma grande vantagem sobre Flash para os dispositivos móveis. HTML 5 também oferece grandes possibilidades de interatividade com o vídeo, com elementos como o *Canvas* por exemplo. Porém, HTML 5 ainda é uma padrão em construção, e todas as funcionalidades não estão ainda implementadas ou funcionais em todos os navegadores.

III.4. Peer-to-peer (P2P)

a) Conceito

Peer-to-Peer (P2P) é uma arquitetura de rede distribuída onde os participantes, chamados de peers, têm os mesmos privilégios. Esta rede P2P se superpõe a uma infraestrutura de comunicação como a Internet por exemplo. Neste sistema, que pode ser representado por grafos, cada peer forma um nó do grafo P2P, e as conexões entre os peers são as arestas entre eles.

Redes P2P são usadas geralmente para aplicações de comunicação ou de distribuição de conteúdo. O protocolo P2P utilizado deve ser capaz de rastrear o conteúdo através da rede, e escolher como conectar os peers entre si. O conteúdo é transferido diretamente entre os peers, a qualidade do enlace entre eles determinando a qualidade da transmissão. Podemos distinguir dois tipos de redes P2P:

- Redes não estruturadas: as conexões estão feitas de maneira arbitrária, sem restrições sobre a estrutura da rede. Geralmente essas redes não são muito eficientes;
- Redes estruturadas: os peers estão conectados e organizados de acordo com políticas implementadas em algoritmos (por exemplo favorecer a localidade na descoberta de peers). As redes resultantes têm propriedades que geralmente oferecem um melhor eficiência e escalabilidade. A qualidade do enlace entre dois peers determinará o desempenho da rede, o que explica porque as politicas de conexão entre os peers são um ponto fundamental.

Como veremos em exemplos a seguir, uma rede P2P bem implementada pode ao mesmo tempo melhorar a qualidade de experiência do usuário, descongestionar a rede Internet e reduzir os custos dos atores envolvidos. Isso depende muito de como é feita a seleção dos peers, em função da distância geográfica entre eles por exemplo. Um outro fator essencial na seleção dos peers é o ISP deles. Estudos mostram que o custo de troca de sistema autônomo (a rede de um ISP é um sistema autônomo) é muito alto [**17**]. Por exemplo, as vezes pode ser mais interessante para um usuário em São Paulo pedir um conteúdo para um outro peer em Manaus mas no mesmo sistema autônomo, do que para um peer no Rio de Janeiro mas em um sistema autônomo diferente.

As optimizações de caminho do P2P e do ISP também podem trabalhar uma contra a outra. O P2P vai tentar otimizar a entrega de conteúdo entre seus peers, procurando o melhor caminho. Mas o aumento de tráfego nesta rota vai diminuir a banda disponível, incentivando o ISP a redirecionar o tráfego desta aplicação para um outro caminho. Estas realocações sucessivas podem acabar não realmente melhorando o desempenho [**18**].

Além disso, uma rede P2P é por definição decentralizada. Isso significa que o conteúdo não fica em um servidor central ou na mão de uma organização. Ele é acessível por todos os membros da rede P2P, de maneira livre e compartilhada. A disponibilidade

de um conteúdo depende completamente do comportamento dos peers da rede. Em casos de streaming com forte demanda, isso é uma grande vantagem. Em outros cenários, como download de um conteúdo que perdeu popularidade, isso pode se tornar uma desvantagem, pois aquele conteúdo será difícil de se encontrar.

b) Atores do mercado

É interessante saber que empresas como a Spotify usaram uma rede P2P para distribuir o conteúdo deles enquanto eles não tinham servidores suficientes para entregar as músicas para todos os clientes [**19**]. Essa estratégia permitiu fazer economias nos servidores no início da empresa. Os usuários do aplicativo para desktop armazenavam no seu disco uma versão cifrada das músicas ouvidas. Quando o usuário queria ouvir essa música de novo, ela estava acessível no disco dele. Além disso, se era a primeira vez que ele escutava essa música, a aplicação ia tentar baixá-la do disco de outros usuários do Spotify desktop (os peers da rede). Em último caso, a música estava baixada do servidor.

Hoje em dia a Spotify tem servidores suficientes para atender a demanda, e manter a rede P2P seria mais custoso.

Podemos também citar a BitTorrent Inc. [**20**], que foi a empresa que criou o protocolo BitTorrent em 2001 e ainda comercializa produtos relacionados de distribuição de conteúdo. Este é o protocolo utilizados por clientes populares como Popcorn Time e µTorrent.

c) Projetos open-source

Podemos citar alguns projetos open-source relacionados com Web Streaming e que usam P2P.

Primeiro, o Popcorn Time que foi um projeto para oferecer uma plataforma parecida com a da Netflix, mas baseada em torrents. A parte interessante deste projeto é que foi liberado de graça pelos seus desenvolvedores, com código aberto.

Um outro projeto de código aberto interessante é o WebTorrent [**21**]. Ele é uma aplicação web e para desktop escrita em JavaScript que permite fazer streaming de torrents usando WebRTC.

III.5. CDN

a) Conceito

A Internet começou com aplicações militares e acadêmicas, e nessa época ninguém imaginava que poderia um dia fornecer e suportar à tantas aplicações como acontece hoje. As possibilidades da Internet são virtualmente infinitas. Mas isso também quer dizer que a estrutura e os protocolos de comunicação não foram pensados para suportar transmissões ao vivo em qualidade 4K[6] para milhões de usuários simultâneos por exemplo.

A Web centralizada com arquitetura clássica Cliente-Servidor reduz as possibilidades de escalabilidade a duas opções: adicionar mais servidores sempre mais poderosos, e/ou usar estratégias para melhorar o desempenho da Web. O problema de latência percebida pelo usuário ao acessar uma página Web é um problema relevante que precisa ser resolvido, embora melhorias significativas aconteceram desde os primeiros tempos. A redução dessa latência foi possibilitada pelo desenvolvimento das redes cabeadas e sem fio, sempre com foco em acelerar a velocidade da Internet para os usuários.

Porém, ainda hoje esse problema persiste, por dois motivos principais:
- o caráter não estruturado da Internet implica que não é otimizada para entrega de conteúdo;
- como já foi mencionado na introdução, o tráfego Internet está em constante crescimento por causa do crescimento dos usuários e do volume de dados trocados pelas aplicações

Esses dois motivos fazem que a primeira solução que consiste em apenas adicionar servidores para responder à demanda não pode bastar. Também é necessário ter uma abordagem mais estratégica a respeito de distribuição de conteúdo, especificamente conteúdo Web. Esse é o desafio que as *Content Delivery Networks* (CDN, Redes de Distribuição de Conteúdo) estão aqui para responder.

Uma CDN é uma rede de distribuição de informação que permite fornecer conteúdo Web de uma forma mais rápida a um grande número de usuários, distribuindo o conteúdo por múltiplos servidores (chamados de servidores "de borda", posicionados na borda da rede) de forma a efetuar a duplicação do mesmo e direcionar o conteúdo ao usuário com base na proximidade do servidor [**22**]. Ao contrário da tecnologia de servidores de cache, as CDN adotam uma atitude proativa, e não reativa. O conteúdo já é distribuído para servidores próximos aos usuários antes deles requererem o conteúdo pela primeira vez.

Isso permite otimizar a latência e o uso de banda. Primeiro, a latência será diminuída porque o usuário buscará o conteúdo de um servidor próximo a ele, e não de um servidor central que pode ser longe dele, tanto geograficamente como topologicamente. Em segundo lugar, o servidor central receberá apenas pedidos dos

[6] Formato de imagem digital de 3840 pixels por 2160 [**64**]

servidores da CDN e não de todos os clientes, o que irá diminuir drasticamente a sua carga e seu uso de banda.

Este sistema também permite economias de escala, porque vários criadores de conteúdo podem usar a mesma CDN, e aproveitar a sua infraestrutura de distribuição. CDN é um serviço utilizado por criadores de conteúdo de médio porte: eles não precisam se preocupar com os mecanismos de entrega.

Finalmente, o último benefício das CDN é que conseguem ter inteligência na distribuição do conteúdo. Elas têm alguns mecanismos para mudar as suas configurações dinamicamente em função de picos de demanda, condições de rede e muitos outros parâmetros. Para otimizar a entrega nesse sentido, os quatro assuntos mais importantes que fazem a complexidade e a qualidade de uma CDN são:

- políticas de distribuição do conteúdo através da CDN
- a escolha da localização dos servidores de borda
- o roteamento dos pedidos (*request routing*)
- os mecanismos de coleta de informações de rede e estatísticas de uso

Por outro lado, o uso de CDN tem algumas desvantagens. O custo deste tipo de serviço é geralmente elevado, e o mercado se divide entre poucos atores. Também pode gerar problemas de privacidade, porque o tráfego é roteado pelos servidores da CDN, e isso pode gerar uma situação onde as empresas de CDN teriam um poder sobre os conteúdos da Internet, e portanto a Web de forma geral.

As CDN são usadas para distribuir conteúdo de vários tipos de aplicações (web, atualizações de software, etc.). Apesar de focarmos no caso de conteúdo Web e particularmente Streaming, o uso de CDN não se restringe a esse caso de uso.

A título de exemplo, o sistema IPTV USP usa uma CDN própria, especificamente para entregar conteúdo entre os vários campi da USP de forma mais eficiente.

b) Arquitetura de uma CDN

A figura Fig. **10** apresenta a arquitetura genérica de uma CDN, com o fluxo de informação entre os elementos desde o servidor origem até o cliente. O funcionamento da rede é descrito a seguir:

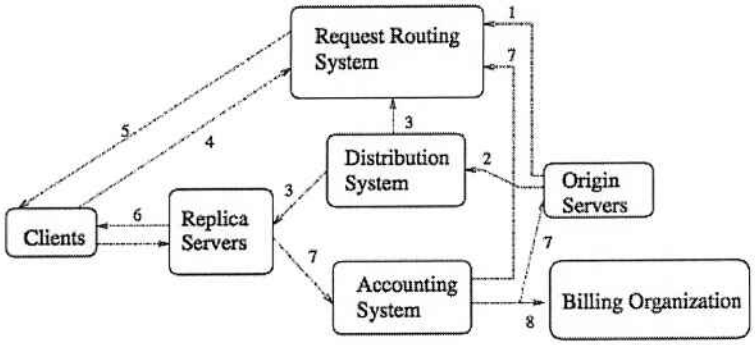

Fig. 10 – Organização geral de uma CDN [23]

1. O servidor origem do provedor de conteúdo delega o gerenciamento do endereçamento do conteúdo pelo sistema de roteamento de requests da CDN
2. O provedor de conteúdo transfere o conteúdo para o sistema de distribuição da CDN para ser distribuído adequadamente com a suas politicas de desempenho
3. O sistema de distribuição manda o conteúdo para os servidores de borda e da um feedback para o sistema de roteamento de requests sobre a localização do conteúdo na rede
4. Os clientes fazem os pedidos que chegam no sistema de roteamento de requests, e não mais no servidor origem
5. O sistema de roteamento de requests redireciona de alguma maneira o pedido para o servidor de borda mais adequado à situação, de acordo com as politicas adotadas
6. O servidor de borda manda o conteúdo para o cliente
7. O servidor de borda retorna estatísticas de uso para o sistema de contabilidade, que por sua vez retorna essas estatísticas para o sistema de roteamento de *requests* e o servidor origem
8. Essas informações também estão usadas para contabilizar o uso da CDN pelo provedor de conteúdo e cobrá-lo.

c) Tecnologias

Após ter uma visão geral do funcionamento de uma CDN, podemos entrar nos detalhes das tecnologias usadas para executar as quatro funções fundamentais identificadas acima. Serão estudadas distribuição, localização dos servidores de borda (no sentido de colocação), roteamento dos *requests* e enfim coleta de estatísticas de uso e de rede.

i) Distribuição

A distribuição do conteúdo do servidor origem até os servidores de borda pode ser feita principalmente de duas maneiras [**23**]. A primeira consiste em usar a própria Internet. Nesse caso, a CDN mantém o controle da localização do conteúdo com uma arvore hierárquica ou uma rede overlay sobre a Internet. Essa abordagem tem a

vantagem da simplicidade, mas também sofre da baixa previsibilidade da Internet em relação a desempenho. Essa abordagem é usada por *Akamai Technologies* e *Sandpiper Networks*.

A segunda opção consiste em usar entrega via *broadcast* por satélite. É mais barata e garante uma entrega com boa previsibilidade de alta qualidade. Esta tecnologia é usada por CyberStar e Edgix.

A questão da distribuição também envolve o número de cópias do conteúdo propagadas, e para onde elas estão mandadas. Intuitivamente, o sistema de distribuição da CDN tentará colocar o conteúdo de maneira a equilibrar a carga em todos os servidores de borda.

ii) Localização de servidores

É fácil entender que a questão da localização dos servidores de borda é de alta importância para uma CDN. Intuitivamente, poderíamos pensar que deve ser o mais próximo possível dos usuários. O objetivo aqui é no mesmo tempo minimizar a latência de acesso ao conteúdo para o cliente, e minimizar a banda global usada na rede. Na verdade, alguns critérios entram em consideração.

Primeiro, a distância física entre o servidor de borda e o cliente é importante. Um servidor mais próximo de um cliente nem sempre significa uma latência menor ou uma melhor banda alocada, mas é uma boa indicação. Como veremos na parte sobre roteamento de *requests*, em alguns casos a CDN não tem informações sobre o cliente, mas infere essas informações da posição do DNS resolver que mandou o *request*. Em outros termos, a posição estimada do cliente depende muito do fato dele estar próximo ao primeiro servidor de DNS que ele contata ou não.

O segundo critério a ser considerado é a topologia da rede, e particularmente a organização dos Sistemas Autônomos (SA). O roteamento entre sistemas autônomos pode ser mais custoso que roteamento de pacotes intra-SA, em termos de desempenho e econômicos. O roteamento inter-SA é feito com o algoritmo BGP, e essa mudança de SA pode virar um gargalo dependendo dos acordos entre sistemas autônomos.

Existem vários algoritmos teóricos para otimizar esta colocação dos servidores na Internet [**24**] [**25**] [**26**]. Eles costumam ser relativamente caros computacionalmente e o uso de heurísticas e mais comum [**27**] [**28**].

iii) Request Routing

O *request routing*, ou roteamento de pedidos (*requests*), tem a função de redirecionar o pedido de conteúdo do cliente para o servidor de borda mais adaptado à situação, considerando alguns parâmetros. O objetivo é maximizar a qualidade de experiência para o usuário final e minimizar o custo da entrega. Consequentemente, os critérios utilizados para escolher o melhor servidor de borda são proximidade e carga [**23**]. As maneiras de conseguir essas informações serão explicadas na seção a seguir sobre coleta de estatísticas.

Três métodos principais são utilizados para fazer *request routing* nas CDN [**18**]: *anycast*, DNS *redirect*, e HTTP *redirect*. Vamos ver as vantagens e pontos fracos de cada método.

Anycast

Este método utiliza *anycast* para escolher o melhor servidor de borda para achar o conteúdo. Vários servidores que possuem o mesmo conteúdo se cadastram no mesmo endereço *unicast*. Quando um usuário quer aquele conteúdo, ele manda um pedido para o endereço *unicast* para que qualquer dos servidores cadastrados neste endereço responda ao pedido. O processo de *request routing* é feito no nível do roteamento de pacotes IP neste método e faz parte do roteamento da rede. Obviamente, a CDN perde o controle do *request routing* neste processo porque é a própria rede que escolha a melhor rota para atingir um dos servidores de borda possíveis, de acordo com as suas métricas. Isso significa que o roteamento é otimizado em relação aos critérios do ISP e não mais da CDN.

É interessante mencionar que em [**29**], os autores propõem uma alternativa de anycast no nível de aplicação. Esta alternativa daria mais flexibilidade, no sentido que poderia considerar-se métricas sobre cada servidor do grupo *anycast* também.

DNS Redirect

A segunda opção é o redirecionamento por mecanismo de DNS. Essa é uma técnica simples, e é a técnica usada pela maioria das CDN comerciais hoje em dia. Na etapa de delegação do endereçamento do conteúdo para a CDN, ela modifica o URL do conteúdo para algo no seu próprio domínio. Dessa maneira, quando o cliente pede o conteúdo, este pedido vai normalmente para o DNS resolver. Como o endereço está no domínio da CDN, o resolver vai pedir o endereço para o servidor autoritativo da CDN. Esse na sua vez retornará o melhor ou os melhores servidores de borda para se buscar o conteúdo. O poder deste mecanismo se situa no servidor autoritativo e a maneira que ele tem de procurar os melhores servidores. Sua escolha é baseada em distância até o DNS resolver (medida pelas métricas descritas na parte seguinte), disponibilidade e custo monetário. É importante notar aqui que o servidor autoritativo não tem informações sobre o cliente, ele recebe o pedido do resolver. Então ele deduz a distância até o cliente, supondo que ele esteja próximo ao DNS resolver.

HTTP Redirect

Enfim, a última opção apresentada aqui é proceder por redireção HTTP. Ao inverso dos casos anteriores, aqui o cliente pede o conteúdo diretamente para o servidor origem como se não tivesse nenhuma CDN. Quando o pedido chegar, o próprio servidor responde com um *status code* especifico para indicar ao cliente que o conteúdo está disponível em outro URL, que aponta para o servidor de borda que ele escolheu.

A vantagem deste método é que o servidor conversa diretamente com o cliente e não precisa inferir as informações sobre ele e fazer suposições. Ele também sabe exatamente qual objeto está sendo pedido. Podemos pensar que isso irá melhorar a qualidade da escolha do melhor servidor de borda. Por outro lado, este método tem um

maior overhead. É necessário criar uma segunda conexão TCP com o servidor de borda, o que envolve uma segunda resolução por DNS, criação de uma segunda conexão TCP (com slow start), etc.

iv) Coleta de estatísticas

O último ponto crítico de uma CDN é a coleta dos dados para fazer o *request routing* ou a cobrança dos clientes. A figura a seguir (Fig. **11**) lista alguns exemplos de métricas úteis para inferir a estrutura e o desempenho da rede e dos servidores particulares. As técnicas para obter essas métricas incluem *active probing* (o cliente manda mensagens para o servidor periodicamente para conhecer a sua carga), *server push* (o servidor manda seu estado de carga periodicamente), *ping* (tempo de resposta) e *traceroute* (número de hops em uma rota).

Metrics	***Goals***	***Measurement Techniques***
Latency	Select replica with lowest delay	Active Probing / Passive Measurement
Packet loss	Select path with lowest error rate (useful for streaming traffic)	Active Probing / Passive Measurement (TCP header info)
Network proximity	Select the shortest path	Active Probing
Avg. Bandwidth	Select the best path for streaming traffic	
Startup Time		
Frame Rate		
Geographical proximity	Redirect requests from a region to the same POP	IP header information, bind information
CPU load, net. interface load, active connection, storage I/O load	Select the server with the aggregated least load	feedback agents/active probing

Fig. 11 - Métricas usadas na seleção do servidor de borda durante o roteamento de request [30]

III.6. WebRTC

a) Introdução

Web Real-Time Communication (WebRTC) é uma coleção de padrões, protocolos e APIs JavaScript (Application Programmation Interfaces) de código aberto, cuja combinação permite troca de áudio, vídeo e dados em P2P entre navegadores web (ou *browsers*, os peers). Em vez de ficar dependendo de plug-ins de terceiros ou de programas proprietários, WebRTC torna essas possibilidades de comunicação em tempo real em uma funcionalidade básica que qualquer aplicação pode aproveitar via uma API JavaScript simples [**31**].

Habilitar comunicações de tempo real dentro do navegador é um grande desafio, provavelmente uma das maiores contribuições à plataforma web desde a sua aparição. WebRTC se afasta do modelo clássico Cliente-Servidor, o que resulta na necessidade de repensar a camada de rede no navegador, e também traz um novo conjunto de ferramentas para um tratamento eficiente de áudio e vídeo, em tempo real.

Dois grupos de trabalho da W3C (World Wide Web Consortium) e da IETF (Internet Engineering Task Force) são responsáveis pela padronização do WebRTC:

- Web Real-Time Communications (WEBRTC) W3C Working Group é responsável pela definição das APIs web;
- Real-Time Communication in Web-browsers (RTCWEB) é o grupo da IETF responsável pela definição dos protocolos, formatos de dados, segurança e todos os outros aspectos necessários para habilitar comunicações peer-to-peer no navegador.

Enquanto o primeiro objetivo do WebRTC é habilitar peer-to-peer no navegador, ele é também projetado para ser integrado com sistemas de comunicação preexistentes: *Voice Over IP* (VOIP), vários clientes SIP (*Session Initiation Protocol*), e até a rede de telefonia clássica (PSTN), entre outros; mas não especifica nenhum requisito de interoperabilidade. Isso significa que está trazendo as possibilidades da Web até o mundo das telecomunicações, o que explica por que tem sido muito considerado por grandes empresas.

Em Setembro de 2016, WebRTC estava suportado pelos seguintes navegadores e ambientes [**32**]:

- Google Chrome
- Mozilla Firefox
- Opera
- Android
- iOS

Portanto, perto de 67% de usuários de navegadores desktop podem usar WebRTC [**33**] (devido em grande parte ao atraso de implementação de partes importantes no Internet Explorer e Edge).

A oportunidade do WebRTC é tão grande que várias aplicações famosas que estavam baseadas em uma arquitetura cliente-servidor passaram a usar WebRTC para adicionar peer-to-peer e ganhar em escalabilidade e eficiência, e acabar com a necessidade de instalar um *plugin* (programa adicional ao navegador) para o usuário. O maior exemplo é o Google Hangouts (aplicação de videoconferência), que passou a usar WebRTC no Chrome desde 2014 [**34**].

Logo depois, a Mozilla anunciou o uso de WebRTC no seu novo programa de videoconferência chamado Hello, nativamente integrado na versão seguinte do navegador Firefox [**35**].

Para confirmar a tendência forte no mercado a favor do WebRTC, Skype anunciou em Julho de 2016 a versão alpha do seu cliente para Linux, usando WebRTC pela primeira vez [**36**].

Um outro domínio de aplicação importante para o WebRTC, e o assunto deste trabalho de conclusão de curso, é o streaming. Introduzir um componente de peer-to-peer na entrega de vídeo, que seja sob demanda ou ao vivo, tem o potencial de cortar os custos de banda de mais de 50% de acordo com um caso de estudo da Streamroot [**3**], e ao mesmo tempo melhorar a qualidade de experiência do usuário final (reprodução mais fluida, menos rebuffering, melhor qualidade de vídeo). Neste mesmo caso da Streamroot, o servidor origem falhou e graças ao peer-to-peer 50% dos usuários nem perceberam o problema.

Os três maiores atores deste segmento são a Streamroot (França/USA) [**37**], Peer5 (Israel/USA) [**38**] e Viblast (Bulgária) [**39**]. Essas empresas ainda são bastante jovens e não temos muita informação sobre os seus clientes, mas elas têm um grande potencial dado o tamanho do mercado de web streaming.

Finalmente, podemos citar o caso da PeerCDN, uma startup criada por alumni da Universidade de Stanford que desenvolveu uma "CDN de próxima geração usando WebRTC para entrega de conteúdo eficiente via peer-to-peer" [**40**]. Essa empresa foi comprada pela Yahoo em Dezembro de 2013, depois de sete meses de existência [**41**].

b) Visão técnica geral

Criar aplicações RTC ricas e de boa qualidade, tais como teleconferência de áudio e vídeo ou troca de dados peer-to-peer, necessita muitas novas funcionalidades dentro do navegador como: capacidades de processamento de áudio e vídeo, novas APIs de aplicações, e interação com meia dúzia de novos protocolos de rede. Para fornecer uma interface mais simples, o navegadores abstrai a maior parte desta complexidade atrás de três APIs primarias [**42**]:

- MediaStream: aquisição de fluxos áudio e vídeo
- RTCPeerConnection: comunicação de dados de áudio e vídeo
- RTCDataChannel: comunicação de dados arbitrários de aplicação

Com apenas uma dúzia de linhas de código JavaScript, qualquer aplicação web pode permitir uma experiência de teleconferência rica com troca de arquivos em peer-to-peer. Porém, essas três APIs são apenas a parte visível. Vários componentes são necessários para juntar tudo: *signaling,* descoberta de peers, negociação da conexão, segurança, e camadas inteiras de novos protocolos.

Como podia ser esperado, a arquitetura e os protocolos permitindo WebRTC determinam as características do seu desempenho: latência de conexão e sobrecarga de protocolos por exemplo. Ao assistir um vídeo o cérebro humano é muito mais sensível a atraso que a partes faltantes; ele consegue preencher esses espaços em branco. Isso implica que neste tipo de aplicação, tempo e atraso são mais críticos do que perda de pacotes. Este fato motivou a escolha de UDP, um protocolo não confiável na entrega de pacotes com menos sobrecarga de desempenho que TCP, como protocolo da base para o transporte de dados. Contudo, UDP é apenas o ponto de inicio. Precisa de muito mais do que UDP básico para conseguir realizar comunicações de tempo real no navegador. Precisamos de mecanismos para atravessar redes "Nateadas", ou seja, redes que utilizam Network Address Translators (NAT), firewalls, negociar as conexões, prover criptografia, implementar controle de congestão e de fluxo, etc. Isso resulta na seguinte pilha de protocolos necessários ao funcionamento do WebRTC (Fig. 12).

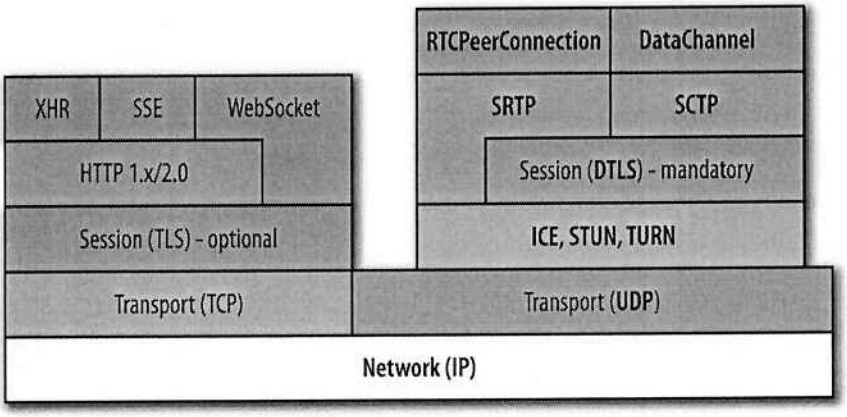

Fig. 12 - Pilha de protocolos WebRTC

- ICE: Interactive Connectivity Establishment (RFC 5245)
  - STUN: Session Traversal Utilities for NAT (RFC 5389)
  - TURN: Traversal Using Relays around NAT (RFC 5766)
- SDP: Session Description Protocol (RFC 4566)
- DTLS: Datagram Transport Layer Security (RFC 6347)
- SCTP: Stream Control Transport Protocol (RFC 4960)
- SRTP: Secure Real-Time Transport Protocol (RFC 3711)

ICE, STUN e TURN são os protocolos necessários para iniciar e manter uma conexão peer-to-peer em cima de UDP. DTLS garante a segurança da transmissão de dados entre os peers; criptografia é obrigatória em WebRTC. SCTP e SRTP servem para multiplexar os fluxos, prover controle de congestão e fluxo, e garantir uma entrega parcialmente confiável em cima de UDP. Enfim, SDP é o formato usado para negociar os parâmetros das conexões peer-to-peer.

No escopo deste projeto, vamos usar apenas duas das três APIs citadas acima:

- RTCPeerConnection, para criar uma conexão entre os peers (papel de controle),

- RTCDataChannel, para criar o canal de comunicação que nos permitirá transmitir os segmentos de vídeo na forma de pacotes de dados brutos.

A terceira API, MediaStream, é mais usada no contexto de aplicações de videoconferência, onde precisa adquirir imagem da câmera e som do microfone.

c) Criar uma conexão peer-to-peer

Para criar uma conexão WebRTC entre dois peers, é necessário cumprir três tarefas:

- Notificar o outro peer que queremos abrir uma conexão com ele;
- Identificar rotas potenciais para a conexão P2P e entregar essa informação para os dois lados;
- Trocar as diferentes informações sobre parâmetros dos fluxos de dados (como protocolos, codecs, etc.)

A segunda tarefa já está sendo resolvida pelo WebRTC através do protocolo ICE. O *Signaling* e a negociação inicial de sessão ficam a carga do desenvolvedor.

i) Signaling e Session Description Protocol

Antes de começar a negociar uma sessão, precisamos saber se é possível trocar informação com o outro peer de algum jeito. WebRTC não especifica nenhum padrão para realizar essa tarefa, e diferentes opções podem ser utilizadas para fazer o *Signaling*, ou seja notificar o outro peer que queremos conversar com ele, e transmitir as mensagens até ele. Opções incluem *Session Initiation Protocol* (SIP), *Jingle*, e *ISDN User Part* (ISUP). Uma outra solução muito comum, e que foi usada no projeto descrito nesta monografia, é criar um protocolo de signaling customizado. No caso, os clientes se conectam com um servidor de signaling (o "Tracker") via um WebSocket, e assim eles podem mandar mensagens para outros clientes via esse canal. Esta parecia ser a escolha mais simples e adaptada ao projeto.

Assumindo que os dois lados compartilham um canal de comunicação (o *Signaling Channel*), podemos negociar os parâmetros da conexão P2P. Para isso, WebRTC usa o protocolo *Session Description Protocol* (SDP). SDP descreve o perfil de sessão, uma lista de propriedades da conexão: tipo de mídias a serem trocadas, protocolos de redes usados, codecs, informações de banda, e outros metadados. A criação e a troca dos arquivos de SDP (a *Description*) já é abstraída em alguns métodos da API, de forma que os desenvolvedores não precisam se preocupar com os detalhes.

O processo está descrito na Fig. **13**. Amy cria a sua *Description* com o método *SetLocalDescription* e manda um *Offer* para o Bob via o *Signaling Channel*. Do outro lado, o Bob grava essa offer com o método *SetRemoteDesciption*, cria a sua própria descrição, e retorna ela para a Amy na forma de uma *Answer*. Amy segue os mesmos passos por sua vez.

Depois disso, as duas partes têm negociado os tipos de fluxos de dados a serem trocados. Apenas faltam testes de conectividade e mecanismos de travessa de NAT, feitos com o ICE.

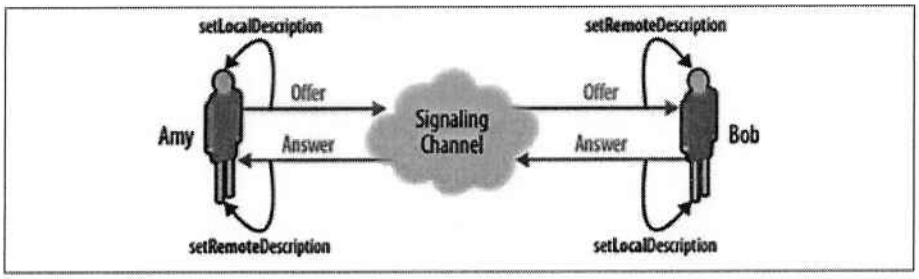

Fig. 13 – Troca de *Offer/Answer* entre os peers [31]

ii) Interactive Connectivity Establishment (ICE)

O protocolo ICE permite achar uma rota possível entre os dois peers para criar a conexão P2P. Para fazer isso, cada objeto RTCPeerConnection possui um *ICE Agent*. Este agente é responsável por coletar pares IP:Porta que são bons candidatos para estabelecer a conexão P2P (*ICE Candidates*). Uma vez que cada cliente recebeu os candidatos do outro, o agente vai executar testes de conectividade até encontrar dois candidatos funcionando. Durante o tempo da conexão, o agente também é responsável por mandar mensagens de "*keepalive*" para manter a conexão ativa.

O primeiro processo é coletar os *ICE Candidates* de cada lado. O *ICE Agent* pede primeiro o IP local para o sistema operacional. Depois, para conhecer o seu par IP:Porta público, ele pede para o servidor STUN configurado. Este servidor funciona da maneira seguinte: ele recebe o *request* do cliente, o que faz aparecer para ele o endereço público deste cliente. O STUN simplesmente retorna este endereço público para o cliente. Este mecanismo é útil em caso de sub-redes privadas atrás de NAT. Em alguns desses casos o cliente não pode conhecer o seu próprio endereço público e então ele tem que perguntá-lo pelo servidor STUN. Esta questão de NAT está explicada em detalhes na seção ii)Firewall / NAT.

Se estiver configurado, o agente coloca o servidor TURN em última opção, se não conseguir uma conexão P2P. Naquele caso, o TURN funciona simplesmente como servidor de retransmissão entre os dois clientes.

A partir do momento que uma *Session Description* está configurada, o *ICE Agent* começa a coletar os candidatos e a troca é feita através do processo de *Offer/Answer*. Depois deste processo, cada lado tem uma lista de candidatos para criar uma rota até o outro cliente. O agente testa as possibilidades até encontrar um rota. Supondo que algum desses candidatos seja bem sucedido, a conexão P2P entre os dois clientes está criada.

iii) Implementação

O processo de criação da conexão WebRTC não é simples, mas algumas partes da complexidade estão abstraídas dentro da API. O trecho de código seguinte representa as etapas necessárias para criar tal conexão (Fig. **14**).

```
var ice = {"iceServers": [
   {"url": "stun:stunserver.com:12345"},
   {"url": "turn:user@turnserver.com", "credential": "pass"}
]};
var signalingChannel = new SignalingChannel();
var pc = new RTCPeerConnection(ice);

pc.createOffer(function(offer) {
   pc.setLocalDescription(offer);
   signalingChannel.send(offer.sdp);
}

pc.onicecandidate = function(evt) {
   if (evt.candidate) {
      signalingChannel.send(evt.candidate);
   }
}

signalingChannel.onmessage = function(msg) {
   if (msg.candidate) {
      pc.addIceCandidate(msg.candidate);
   }
}
```

Fig. 14 – Exemplo de código para criação de conexão WebRTC (JavaScript)

Primeiro, configuramos o STUN com o servidor de STUN público da Google. Depois, criamos o canal de comunicação e o objeto *PeerConnection*. Mandamos uma oferta, e a cada novo *ICE Candidate*, ele é mandado no canal de comunicação. Na recepção de *ICE Candidates*, eles estão adicionados ao objeto *PeerConnection* para iniciar os testes de conectividade e finalmente estabelecer a conexão.

d) Datachannel

O *DataChannel* é uma outra API do conjunto WebRTC que está sendo usada neste projeto. Um *DataChannel* permite a troca bidirecional de dados arbitrários entre dois peers conectados via uma conexão WebRTC. De novo, parte da complexidade está abstraída dentro da API, e qualquer um dos peers pode abrir um *DataChannel* usando alguns métodos da *PeerConnection*.

Um fato importante sobre *DataChannel* para a nossa aplicação é a limitação de tamanho de cada mensagem. Embora não tenha um limite fixo na especificação original, parece que os navegadores implementam um limite da ordem de 200 bytes. Isso justifica

decisões do projeto explicadas na seções seguintes sobre a maneira de transmitir os segmentos de vídeo entre os peers.

Outras limitações do WebRTC estão explicadas na próxima seção, e) Limitações.

e) Limitações

WebRTC sendo ainda um padrão em desenvolvimento, é necessário levar em conta alguns problemas e limitações inerentes a ele. Vamos citar três desses.

i) Segurança

Como qualquer aplicação na Web, uma aplicação usando a API WebRTC é vulnerável a várias ameaças, como o famoso ataque do "Homem no meio" por exemplo (um exemplo de tal ataque neste artigo, por negligência na verificação do certificado do servidor de sinalização, porque cada cliente gera o seu certificado auto-assinado [**43**]). Também podemos citar o fato que o WebRTC revela o seu endereço IP pelo próprio funcionamento do protocolo ICE, que cria candidatos na forma de IP:porta através de um servidor STUN [**44**].

Porém, há vários argumentos a favor do WebRTC sobre segurança, comparado com varias outras tecnologias e aplicações, como:

- As comunicações são criptografadas com o protocolo Datagram Transport Layer Security (DTLS, RFC 6347), que é similar ao Transport Layer Security (TLS, RFC 5246) mas adaptado para ser usado em cima de UDP (fora a autenticação dos clientes que as vezes deve ser feita pela própria aplicação, está questão sendo discutida ainda [**31**]);
- O ritmo de atualizações é alto (a cada 6 a 8 semanas no navegador). Em caso de falha descoberta, um patch será disponível automaticamente e rapidamente;
- Pede permissão para adquirir o som do microfone e a imagem da câmera para o uso da API MediaStream.

ii) Firewall / NAT

Existem casos onde a conexão peer-to-peer não pode ser estabelecida com WebRTC. Os dois casos mais significativos são os seguintes.

A conexão entre os peers é feita com o protocolo ICE, que cria vários candidatos IP:porta, começando com o endereço IP local. Se não funcionar, ele vai tentar o endereço privado, que funcionará se os dois clientes estiverem na mesma rede. Se ainda não funcionar, ele vai precisar conhecer o seu endereço IP público, com a ajuda do servidor STUN. Este mecanismo é essencial para atravessar NATs (Network Address Translators). Na maioria dos casos, o servidor STUN vai conseguir retornar para o cliente seu endereço IP; mas no caso de um NAT Simétrico (caso particular de NAT), isso não vai funcionar. Neste caso, o endereço público do cliente atrás do NAT depende do endereço do host que mandou uma mensagem. Isso quer dizer que o endereço IP

público do cliente retornado pelo STUN será particular a esse STUN, e será inutilizável para qualquer outro cliente externo a essa sub-rede [**45**].

Um outro caso bastante comum é o bloqueio de UDP por um Firewall. Como já visto anteriormente, os protocolos de comunicação do WebRTC rodam em cima do protocolo de transporte UDP, então quando os datagramas UDP foram bloqueados por um firewall, a conexão WebRTC não pode ser estabelecida. Bloquear as conexões UDP nas regras de um firewall corporativo é uma pratica comum, por questões de segurança (planejar uma ataque de Distributed Denial of Service (DDoS) em UDP é fácil, executando um *UDP flood* por exemplo, e por isso UDP pode ser bloqueado mais facilmente que TCP já que poucas aplicações corporativas usam UDP).

iii) Perda de pacotes (UDP)

WebRTC usa protocolos de transporte que usam UDP, que não é um protocolo com entrega confiável. A perda de pacotes em uma aplicação de vídeo chat ou streaming pode resultar em uma queda da qualidade de experiência para o usuário final. Não vamos entrar nos detalhes das propostas de técnicas de atenuação deste efeito, mas podemos citar o trabalho de uma equipe da Google que propõe um método hibrido com *Negative Acknowledgement*, *Forward Error Correction* e *Temporal Layers* para balancear os requisitos de qualidade do vídeo, regularidade da reprodução e atraso [**46**].

IV. Arquitetura da proposta

IV.1. Requisitos

Em relação aos objetivos mencionados na seção "II-Objetivo e Motivação", a finalidade do projeto será atingida ao conseguir maximizar a proporção de tráfego através da rede P2P em relação à rede CDN.

Este sistema é um protótipo preliminar a uma integração futura com a IPTV da USP, e o projeto foi desenvolvido com essa ideia de integração em mente desde o início.

Os requisitos funcionais do sistema são os seguintes:

1. Implementar mecanismo complementar de entrega de vídeo com objetivo de diminuir o uso de recursos. O cliente deve baixar segmentos de vídeo a partir de peers quando for possível, e da CDN quando não;
2. Implementar esse mecanismo de forma integrada com mecanismos de entrega de conteúdo tradicionais, em que será utilizado como estudo de caso o IPTV da USP;
3. O sistema é aplicado a vídeos de tipo VOD (vídeo sob demanda). O caso dos fluxos ao vivo (live streaming) não é atendido pelo sistema inicialmente;
4. O sistema deve permitir a reprodução em uma página web via um player;
5. O sistema funciona com o formato de streaming adaptativo HLS inicialmente, pois é o formato usado pela plataforma IPTV USP.

O desempenho do sistema foi avaliado durante as fases de testes e de demonstração. Esta avaliação foi feita através de duas métricas principais, comparadas no caso da arquitetura com P2P e sem P2P:

1. Porcentagem dos dados vindo da CDN / porcentagem dos dados vindo do P2P (com objetivo de maximizar a proporção de P2P);
2. Qualidade de experiência medida em tempo de bufferização, número médio de pausas de rebufferização por minuto.

IV.2. Arquitetura geral

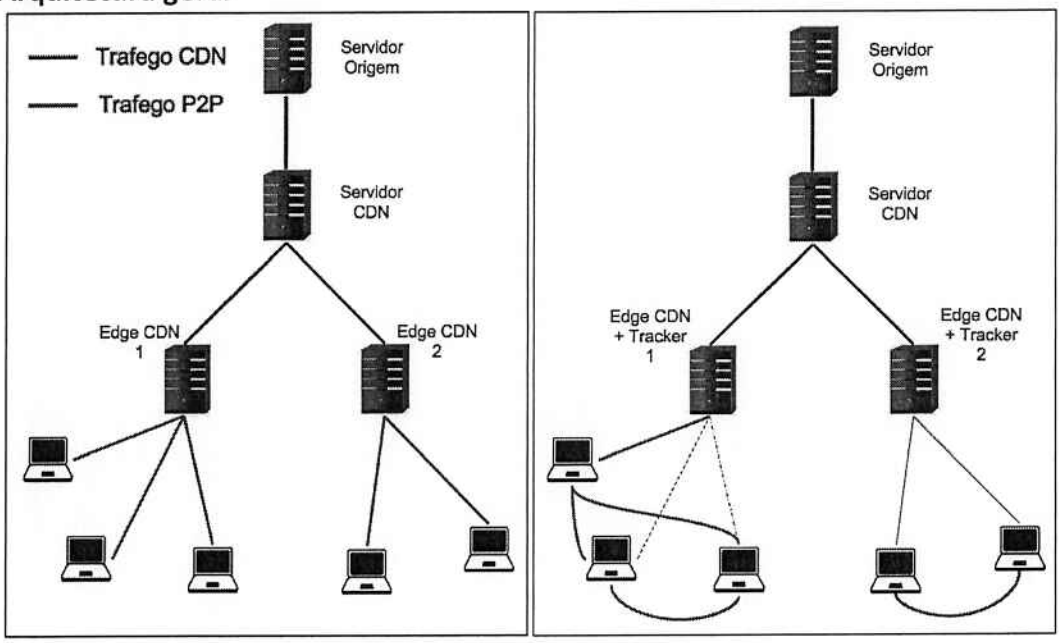

Fig. 15 – Arquitetura geral sem P2P a esquerda (A) e com P2P a direita (B)

A arquitetura geral de um cenário de streaming distribuído com uma CDN é apresentada na figura Fig. 15. À esquerda (A), temos o caso clássico de distribuição exclusivamente via CDN, sem P2P. O conteúdo é inicialmente criado pelo servidor de streaming. Depois, este conteúdo é mandado para a rede CDN para otimizar a sua distribuição. Finalmente, cada cliente requer os segmentos de vídeo para o servidor de borda da CDN que foi atribuído a ele. O tráfego de dados entre o servidor de borda ("*Edge*") e o cliente é representado em vermelho.

Do lado direito, a mesma arquitetura mas com P2P. O tráfego P2P entre os clientes (peers) é representado em verde. Podemos perceber que com a ajuda do tráfego P2P, os clientes irão depender muito menos do servidor de borda. A maioria dos segmentos serão baixados a partir dos outros clientes, e o servidor de borda será usado apenas no evento da falta daquele segmento na rede P2P (swarm). De maneira evidente, a quantidade de banda usada em saída de cada servidor de borda será reduzida.

Os benefícios são múltiplos:

- Cortar custos de banda: o site transmissor paga uma taxa para a empresa de CDN baseada na quantidade de banda que ele vai precisar. Diminuir o uso de banda significa a diminuição de uso de recursos para a CDN (banda, processamento, número de servidores de borda), e portanto uma diminuição de custo;

- Aguentar um tráfego extremo ou maior do que foi planejado ao contratar a CDN: este tipo de situação pode acontecer em eventos esportivos, saída de um seriado em uma plataforma de VOD, conferência. Se não foram planejados bastante recursos e o número de conexões de clientes for maior ao máximo que a CDN pode aguentar, os servidores vão cair e/ou recusar conexões. Se o

P2P estiver habilitado, isso não acontecerá, pois maior o número de peers, melhor a rede funciona. Os servidores irão ficar relativamente menos carregados que com um tráfego médio;

- Tolerância a falha do servidor de streaming: se, por qualquer motivo, o servidor de streaming chegasse a falhar, a rede P2P poderia tomar conta da distribuição sem impacto para os usuários. Os clientes irão pedir o conteúdo diretamente para outros peers, e a falta de servidor não seria sensível para alguns clientes;

- Qualidade de experiência do usuário (melhor qualidade de vídeo, reprodução mais fluida, etc.): o fato de ter mais nós distribuindo o conteúdo em vez de um servidor só pode ter um impacto sobre a qualidade de experiência do usuário. Por exemplo, a ausência de limite de banda do lado de um servidor permite que o fluxo seja distribuído em melhor qualidade para um usuário se ele tiver banda suficiente.

Pode-se observar na Fig. **15**, do lado do P2P, a presença nos servidores de borda da CDN de "trackers". Estes trackers são responsáveis por conectar os peers entre si e pelo bom funcionamento do P2P. De maneira clássica e mais natural, poderíamos pensar utilizar apenas um tracker central responsável por todos os peers. Todos os clientes se conectariam a este mesmo tracker central, e todos poderiam se conectar potencialmente entre si em P2P. Mas uma outra abordagem foi escolhida para este trabalho, e isso constitui parte da sua originalidade.

Aqui, foi escolhido ter um tracker diferente para cada servidor de borda. Isso significa que apenas os clientes redirecionados para aquele servidor de borda se conectarão ao tracker hospedado na mesma máquina que o servidor de borda. Consequentemente, os clientes susceptíveis de se conectar em P2P serão clientes "próximos", de acordo com os clientes da CDN. A ideia aqui é de aproveitar os mecanismos da CDN (especificamente o request routing, como visto na seção III.5.c)iii) Request Routing) para conectar apenas clientes que estão próximos e que poderão aproveitar no máximo do P2P, de forma a evitar fenômenos não desejados mencionados na seção III.4 Peer-to-peer (P2P) (como a travessa de vários sistemas autônomos por exemplo) que poderiam aumentar o custo e diminuir as vantagens do sistema com P2P.

O sistema proposto neste trabalho delega totalmente à CDN a etapa de escolha do sub-conjunto de clientes para formar as redes P2P, que podemos chamar de processo de formação do swarm. O sistema aproveita, de maneira implícita, das informações sobre a rede que a CDN tem para aumentar a sua própria eficiência. Também serão expostas as consequências desta escolha no cenário do IPTV da USP na parte correspondente.

IV.3. Escolhas técnicas

No início e ao longo do projeto, algumas escolhas técnicas tiveram que ser feitas. Uma vez a ideia definida, foi necessário escolher qual tecnologia de P2P íamos usar, qual

protocolo de streaming e qual player para reproduzir o vídeo. Depois disso, ao desenvolver as diferentes partes, surgiram necessidades pontuais para juntar tecnologias. Discutiremos isso também no último item.

a) Protocolo para P2P: WebRTC

A partir da ideia de adicionar o poder do peer-to-peer ao web streaming, foi necessário elaborar como integrar as tecnologias. Poderia ter sido via um programa externo (um plugin) no navegador. Neste caso, não teria sido possível usar um player HTML5 por motivos técnicos, e então teria sido necessário usar um player Flash.

Porém, quando apareceu a opção do WebRTC, imediatamente foi claro que era a melhor possibilidade. Reunia todas as vantagens, das quais podemos citar:

- adicionar peer-to-peer sem precisar de um plugin, de maneira transparente,
- facilidade de implementação com API em JavaScript que é a linguagem da Web,
- uso das funcionalidades nativas de vídeo de HTML5, sem precisar de Flash Player. Desenvolver em Flash implicava a necessidade de usar ActionScript, uma outra linguagem especifica, o que teria aumentado a complexidade de desenvolvimento. Também olhando do lado do mercado, o futuro do setor de vídeos na web está com tecnologias HTML5 e não Flash, que já foi banido na plataforma iOS desde 2010 [**47**]. Mais recentemente, novas falhas de segurança foram e ainda estão sendo descobertas [**48**],
- tecnologia pela grande maioria dos projetos envolvendo web e peer-to-peer hoje, o que da acesso a uma comunidade e a recursos atualizados e detalhados. Isso foi uma grande ajuda durante o desenvolvimento, tanto pelos documentos explicativos que pelos projetos open-source liberados para o uso de todos.

Por outro lado, uma desvantagem do WebRTC é sua relativa novidade. WebRTC não é uma tecnologia madura, ainda não está totalmente implementada em todos os navegadores principais, e implementações podem variar entre navegadores. Isso diminui a sua abrangência, e pode resultar em bugs.

Levando tudo isso em conta, usar WebRTC foi uma escolha natural e a mais evidente de todas.

b) Protocolo de streaming adaptativo: HLS

Quando se fala de streaming hoje em dia, os protocolos dominantes são os protocolos adaptativos, por todos os motivos expostos na parte III.1. Dentro desta família, os dois maiores no mercado são HLS e MPEG-DASH [**49**]. Como já explicamos na parte III.1.b), de um ponto de vista técnico, o mais completo é o MPEG-DASH. Isso faz sentido pois ele foi elaborado para ser um padrão, o que ele é desde 2011. Não é o caso do HLS, propriedade da Apple, que foi criado anos antes, o que pode justificar que ele seja um pouco menos adaptado aos usos atuais em web streaming.

Para resumir, as diferenças técnicas maiores são a multiplexação de áudio e vídeo na mesma trilha (perda de flexibilidade) e um arquivo de "manifest" ou playlist que é mais difícil de ser manipulado no caso de HLS.

Porém, é necessário usar HLS para fazer streaming na plataforma iOS da Apple. Considerada a popularidade dos aparelhos como o iPhone ou o iPad, muitas plataformas adotaram o HLS. Essa popularidade se manifesta também no número de projetos de streaming e WebRTC que são construídos com iOS. Da mesma maneira que para WebRTC, o tamanho da comunidade e a quantidade de recursos sobre HLS são indícios de confiança para o futuro do formato.

Na hora de escolher entre os dois, o princípio de integração com o IPTV da USP foi a prioridade. Durante o desenvolvimento do projeto, o IPTV estava em transição para implementar o HLS, por motivos de suporte a aparelhos iOS. Consequentemente, foi decidido optar por HLS para o projeto.

c) Player: video.js

Em relação ao playback, o uso de um *player* não é absolutamente necessário. Porém, como nos dois itens anteriores, os fatores determinantes na escolha final foram o *player* usado pelo IPTV da USP e os recursos e projetos *open-source* disponíveis. O IPTV da USP usa video.js, um player *open-source*, que é usado em muitos projetos (inclusive projetos que poderiam se revelar úteis para o resto do projeto). De novo, a interoperabilidade com o IPTV USP ficou como prioridade e foi decidido usar o video.js como player.

d) Outros módulos open-source

Para juntar esses componentes, foi necessário usar alguns pequenos projetos de compatibilidade. Não é raro que as empresas do setor liberem estes módulos em *open-source* para incentivar o uso da tecnologia. É o caso da Streamroot por exemplo, já mencionada anteriormente. O fato desses projetos serem de código aberto permite criar forks para adaptar esse código a uma necessidade específica do contexto do projeto. Levando em conta a gratuidade, eles se tornam a opção ideal, como detalharemos na seção de tratando da arquitetura do software.

IV.4. Arquitetura do software

a) Parte de mídia

O diagrama seguinte representa a estrutura geral do software (Fig. **16**). Cada bloco representa um modulo com a sua função, como vamos detalhar em seguida. Para atender as funcionalidades e os requisitos expostos na parte anterior, precisa-se de uma arquitetura complexa, envolvendo vários blocos tecnológicos de origens diversas. Os blocos representados em azul são os módulos externos, de código aberto e de uso livre. Os blocos em verde são forks de projetos abertos. Isso quer dizer que são módulos de código aberto também, mas que tiveram que ser adaptados à nossa arquitetura para funcionar com os outros componentes. Finalmente, os blocos em preto são os módulos desenvolvidos desde o inicio.

O software foi desenvolvido em JavaScript para a parte cliente, que é a linguagem feita para escrever programas executados dentro do navegador. JavaScript também naturalmente é a linguagem da API WebRTC. Essa escolha foi natural.

Os módulos são:

- index.html: a página web onde será reproduzido o vídeo, junto com as métricas de desempenho.

- SourceHandler: módulo que permite atrelar a fonte de streaming ao elemento html do player [**50**].

- video.js: módulo do player. O video.js é um player html5 de código aberto e gratuito. Ele é um dos players mais usados na web hoje em dia, e empresas como Twitter, Instagram, Microsoft, Dropbox e Github o utilizam [**51**].

  hls.js: biblioteca JavaScript que implementa um cliente HLS [**52**]. Se apoia em vídeo HTML5 e Media Source Extensions [**53**], que é o componente HTML5 que permite manipular fluxos de vídeo diretamente com JavaScript no navegador. Essa parte é essencial no nosso caso, pois o fluxo precisa ser processado e reconstituído pela nossa aplicação em JavaScript. O hls.js gerência o buffer e transmite os pedidos de segmentos para o P2PModule. Transmultiplexa MPEG-2 Transport Stream (formato dos segmentos HLS) em segmentos ISO BMFF (MP4). Essa etapa de transmultiplexação pode ser feita de maneira assíncrona usando Web Workers (jeito de paralelizar o processamento no navegador), se for possível.

- hlsjsP2PWrapper: atrela o módulo P2PModule ao hls.js [**54**]. Cria uma instância do hls.js mudando a configuração padrão para uma configuração onde o elemento de download de segmentos não é mais aquele do hls.js, mas passa a ser o P2PModule. Isso permite adicionar as funcionalidades do módulo P2P ao hls.js. Principalmente, ele oferece a possibilidade do P2PModule fornecer segmentos para o hls.js, tanto do P2P como da CDN, de forma transparente.

- P2PModule: módulo P2P, adiciona a capacidade de baixar segmentos de vídeo de outros peers para complementar o modelo de carregamento da CDN. Ele contém a lógica de chaveamento entre P2P e CDN, e isso fica abstraído para o hls.js. O seu funcionamento será detalhado com mais profundidade na próxima seção.

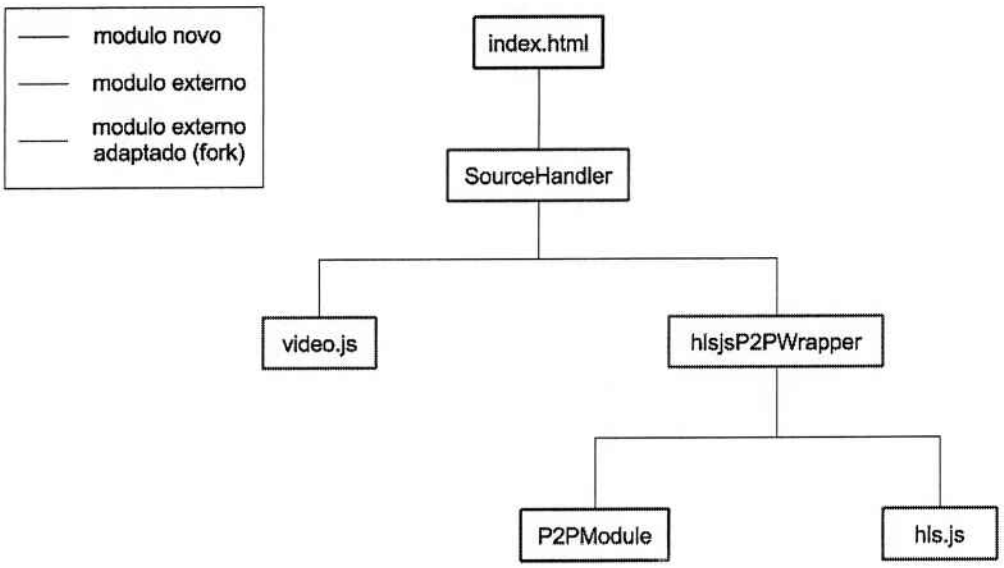

Fig. 16 – Diagrama de blocos da estrutura geral do software

Para resumir, o hlsjsP2PWrapper, hls.js e P2PModule formam um conjunto que representa uma instância de hls.js acrescentada com capacidade de P2P.

Além disso, o SourceHandler permite indicar para o video.js aquela instância modificada do hls.js como sua fonte de mídia.

Ao longo do desenvolvimento, outras possibilidades de arquitetura surgiram. Por exemplo, teria sido possível não usar o hlsjsP2PWrapper e modificar a classe *Loader* do hls.js para acrescentar a funcionalidade de P2P. Porém, esta abordagem presentava desafios estruturais ligados com o funcionamento do hls.js que conduziram a decidir não mexer no hls.js e usar o Wrapper para fazer a interface com um módulo P2P externo.

b) Parte P2P

A estrutura do módulo P2P é apresentada no diagrama de classes a seguir (Fig. 17). Este diagrama representa as classes do software cliente e a maneira na qual elas interagem. Vamos detalhar o papel específico de cada classe e as funções que ela tem que assumir para o bom funcionamento do módulo como um todo.

- *PeerAgentModule*: ponto de entrada do módulo P2P. Faz a interface entre o módulo mídia e o módulo P2P. Principalmente, ele passa os pedidos de segmentos para o P2P e retorna os segmentos baixados e as estatísticas para a mídia.

- *Main*: ponto central do módulo P2P. Passa os pedidos de segmentos para o *ResourceRequester* verificando se o segmento já não se encontra no cache local, e ao receber o segmento da rede, ele atualiza as estatísticas no *PlaybackInfo* e armazena o segmento no *Storage*.

- *PlaybackInfo*: conserva e calcula as estatísticas sobre origem dos downloads e sobre os peers para passar para o front-end.

- *Storage*: classe de armazenamento dos segmentos de vídeo. Os segmentos não são armazenados em formato binário mas em string base 64.

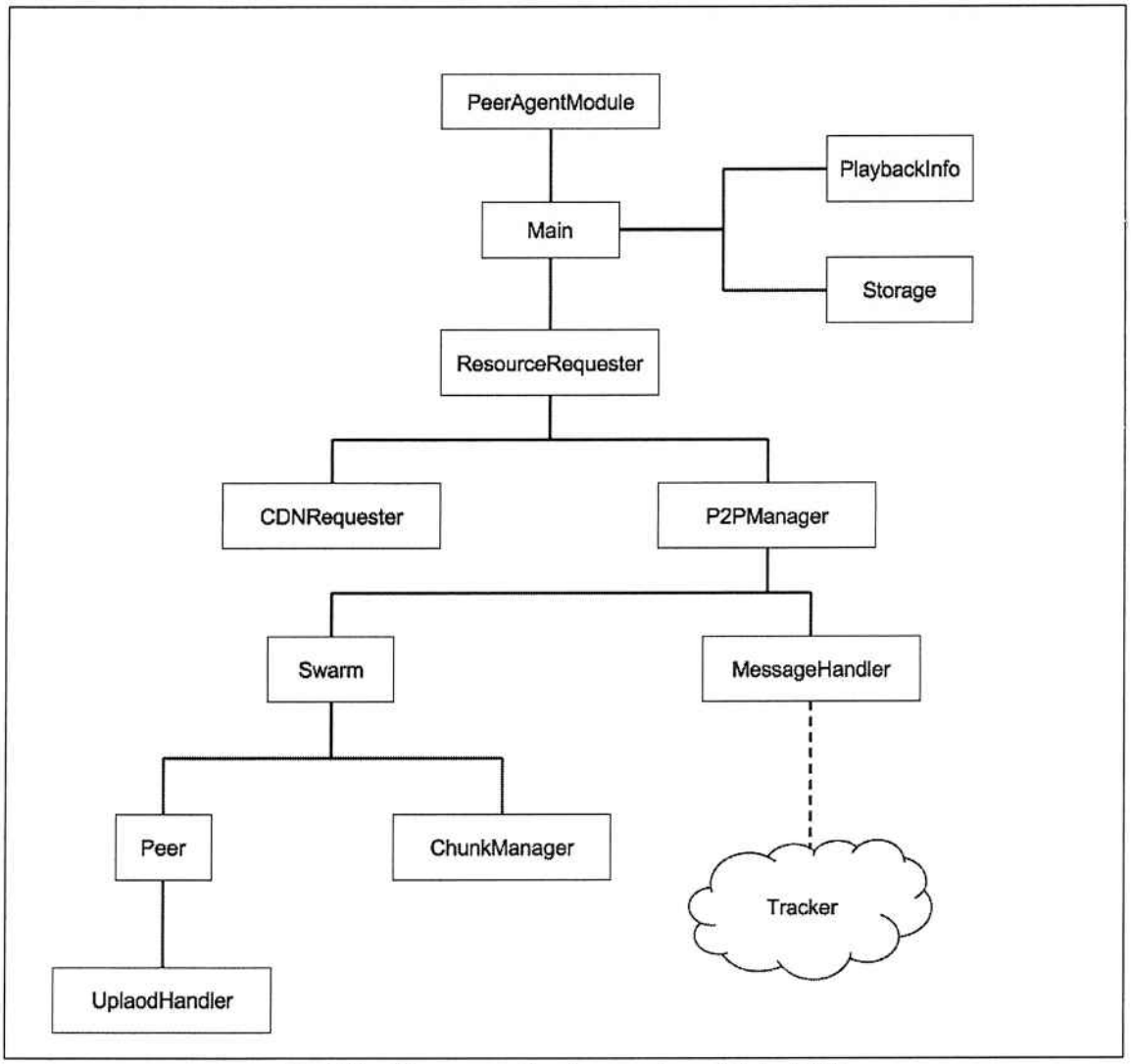

Fig. 17 – Diagrama de classes do módulo P2P

- *ResourceRequester*: responsável pelo roteamento dos downloads entre a CDN e o P2P. Usa alguns critérios configuráveis para escolher, como por exemplo o número de segmentos a baixar da CDN para conseguir iniciar o playback rapidamente enquanto nos conectamos ao tracker e aos demais peers via WebRTC.

- *CDNRequester*: chamado para fazer download do segmento da CDN. Faz um XML HTTP Request para a rede na url de identificação do segmento.

- *P2PManager*: comparável ao *CDNRequester*, mas para o P2P. O *P2PManager* cuida do P2P como um todo. Ele é responsável por se conectar ao tracker via websocket, criar as conexões WebRTC com os outros peers, e repassar os pedidos de segmento para o *Swarm*.

- *MessageHandler*: recebe as mensagens do tracker (lista de peers, signaling WebRTC, etc.) e toma as ações necessárias (responder a uma offer com uma answer por exemplo no caso do signaling).

- *Tracker*: o tracker é um servidor com qual cada cliente se conecta ao iniciar o playback e que rastreia os peers e o conteúdo que eles têm. Ele serve para indicar com quais peers um cliente pode se conectar, e serve também como servidor de signaling para iniciar as conexões WebRTC.

- *Swarm*: representa o swarm, ou seja o conjunto de peers com quem estamos conectados. Ele manda mensagens para peers para saber quem tem o conteúdo desejado e depois seleciona o melhor peer para pedir o segmento.

- *Peer*: representa um peer. Escuta e responde no DataChannel WebRTC (dados, pedidos, etc.)

- *UploadHandler*: responsável pelo upload, basicamente retorna se existem slots de upload disponíveis para um peer dado no momento.

- *ChunkManager*: esta classe trata o segmento no nível de chunk. Cada segmento é cortado em vários chunks de tamanho menor antes de ser enviado para outros peers. Ao receber os chunks, o *ChunkManager* também reconstitui o segmento que será mandado para a parte de mídia.

Podemos visualizar a troca de mensagens na figura a seguir (Fig. **18** - Conexão com o tracker e download de um segmento de um outro peer), tanto com o tracker como com um outro peer. A parte de conexão WebRTC já foi explicada na seção III.6.c) Criar uma conexão peer-to-peer e não é detalhada de novo aqui. Aparece resumidamente na expressão "Signaling WebRTC".

No início, o Peer A se conecta com o tracker que foi atribuído a ele via websocket. Esse tracker responde com a lista de peers com quem ele tem uma conexão aberta e que

têm o mesmo conteúdo que o Peer A está assistindo. Esses são os peers potenciais do nosso Peer A. No caso do protótipo, temos apenas um conteúdo e todos os clientes conectados ao tracker são peers potenciais para A. Com essa lista, o Peer A cria uma conexão WebRTC com cada um desses peers potenciais, usando o tracker como canal de signaling. A partir daqui será considerado um Peer B com quem o Peer A se conectou.

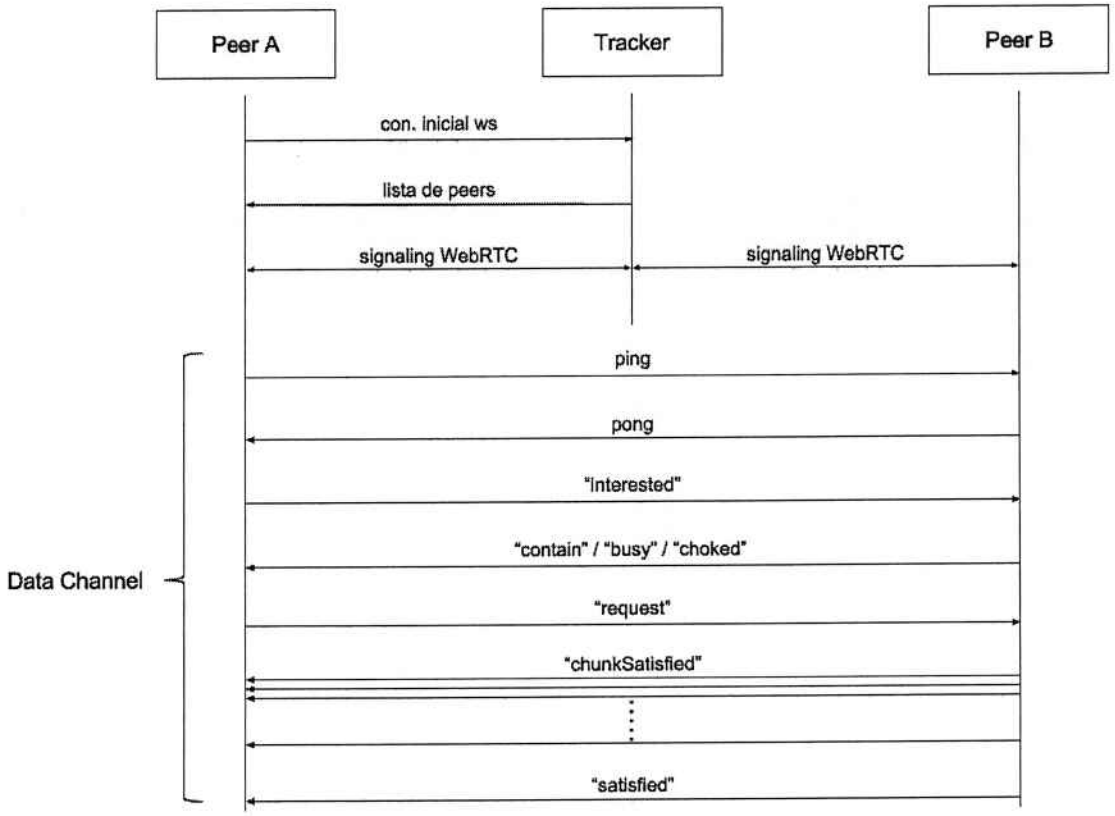

Fig. 18 - Conexão com o tracker e download de um segmento de um outro peer

Ao abrir o DataChannel com o Peer B, a primeira coisa feita pelo A será mandar uma mensagem "ping" (diferente da noção de ping na camada de rede) pelo DataChannel para testá-lo. O B deve responder com um "pong". Se ele não responder, a conexão não funcionou e o processo acaba aqui. Caso contrário, a partir daqui todas as comunicações entre peers A e B estão feitas através do DataChannel, par a par.

Ao longo do playback, o A irá pedindo os segmentos de vídeo para o sistema híbrido. Para garantir uma latência inicial baixa, o sistema pede automaticamente os dois primeiros segmentos da CDN, para dar tempo ao P2P para se iniciar. Depois disso (alguns segundos), cada pedido de segmento entrará pelo *PeerAgentModule* até o *ResourceRequester*. Aqui, o sistema decide se ele pede o segmento para o P2P ou para a CDN. Os parâmetros que ele usa são a presença de peers (o B, no caso) e se o P2P está ativado pelo usuário ou não. Se não for, pede o segmento para a CDN. No caso contrário, passa o pedido para a classe *Swarm*. Primeiro, precisa saber quem tem o conteúdo dentro do swarm. Esta classe manda uma mensagem de "interested" com os detalhes do segmento que ela quer para todos os peers do swarm.

Eles verificam no seu *Storage* se tiverem aquele segmento, e no *UploadHandler* para saber se estão em condição de mandar conteúdo (slots de upload livres). Depois desta verificação, eles respondem com uma das três mensagens seguintes:

- "busy": significa que aquele peer B já está no seu limite de upload, e não pode mais mandar conteúdo para outros peers no momento. Este limite de upload é configurável e é necessário para não saturar o peer e diminuir o desempenho da rede;
- "choked": significa que o peer B não possui aquele segmento no qual o A tem interesse;
- "contain": significa que o B possui aquele segmento e está com condição de mandá-lo para o A.

Ao receber as respostas desses peers, o A constitui a lista de seeders potenciais para este segmento com os peers que responderam com um "contain".

A próxima etapa é escolher o melhor peer entre os seeders potenciais para fazer o pedido do segmento. Para isso, a classe *Swarm* se baseia na pontuação que ele atribuiu para cada peer do seu swarm. Quando iniciar uma conexão P2P entre dois peers, cada um cria uma pontuação que é inicialmente igual para todos, e que serve para avaliar os outros peers. Esta pontuação evolui: ela cresce quando o peer manda bons segmentos, e diminui quando ele não responde ou não manda um segmento que ele prometeu mandar por exemplo. Portanto, esta pontuação é pessoal, no sentido que um peer X pode ser avaliado com a pontuação 50 por um peer Y e -100 por um outro peer Z. Usando esta pontuação, o A escolhe o melhor peer (B no nosso caso) e manda para ele a mensagem de "request", ou seja o pedido com os detalhes do segmento que ele quer.

Do lado de B, ao receber a mensagem na instância da classe *Peer* que representa o peer A, ele recupera o segmento no seu *Storage*. Para mandar o segmento para A, ele precisa usar o *ChunkManager* que vai cortar o segmento em vários chunks, ou seja pedaços menores de vídeo. Esta etapa é necessária por causa do limite de tamanho das mensagens no DataChannel, como foi visto na descrição do WebRTC. B manda cada chunk para A com um número de sequência, nas mensagens de "chunkSatisfied". Do seu lado, A reconstitui o segmento com os chunks. Quando o segmento for completo, ele fica satisfeito. O *Main* recupera o segmento, atualiza as estatísticas (porcentagem total de P2P, tamanho do swarm, upload e download de cada peer do swarm) e retorna o segmento para o módulo de mídia.

Quando B se desconecta, o A será notificado pelo seu listener de fechamento do DataChannel (e pelo tracker também), e a sua instância do peer B será descartada.

IV.5. Desenvolvimento do projeto

Até então, o texto focou muito na arquitetura e no funcionamento do sistema. Também é interessante e faz parte do trabalho detalhar como foi o processo ao longo do ano e os passos seguidos para chegar no protótipo final.

Na origem, eu queria trabalhar com streaming, CDN e P2P e fui procurar a professora Regina Melo Silveira do LARC para me ajudar na escolha final do tema.

Também conversamos com o Samuel Kopp, especialista de CDN do LARC, e depois de algumas reuniões definimos o assunto do trabalho em abril de 2016. A primeira decisão importante foi de organizar reuniões frequentes com a professora e o Samuel para agilizar o avanço de projeto. A cada duas semanas, eu mostrava meus avanços e os novos problemas que tinham aparecido, e nos tomávamos as decisões necessárias, juntos, nessas reuniões.

O projeto se dividiu em quatro partes, não necessariamente sucessivas:

- pesquisa exploratória
- desenvolvimento do protótipo
- redação da monografia
- elaboração da demonstração e testes

Além das reuniões frequentes e de forma a ainda mais facilitar a comunicação com os meus orientadores, decidi trabalhar no LARC a maioria do tempo. Este laboratório me ofereceu um ambiente ideal para realizar as minhas tarefas, sempre com a possibilidade de procurar alguém para me ajudar com qualquer problema ou pergunta. Com experiência, posso dizer que foi uma das decisões mais importantes do trabalho.

A segunda ideia muito forte que sempre segui durante este projeto é a ideia de dividir o trabalho em pequenas sub-tarefas com datas de entrega de produtos intermediários definidas com antecedência. Isso ajudou de várias maneiras. Primeiramente, fixou objetivos a atingir. Segundo, me permitiu dividir as dificuldades em dificuldades menores. Enfim, o fato de ter objetivos menores significa que você atinge os seus objetivos mais frequentemente (tipicamente a cada uma ou duas semanas), e isso gera uma satisfação que ajuda a sempre guardar o foco e não se desmotivar ao olhar a quantidade de coisas para fazer ao longo do projeto.

Este conceito geral se aplicou muito bem neste projeto, com os objetivos sucessivos seguintes:

- arquitetura básica
- página web com player video.js reproduzindo um video .mp4 armazenado no disco
- página web com vídeo HLS hospedado por um servidor público reproduzindo sem player, usando hls.js
- uso do SourceHandler para reproduzir o fluxo HLS no player video.js
- conexão com o Wrapper para ter uma interface viável para um módulo P2P
- conexão do modulo P2P vazio com o Wrapper
- realização da conexão WebRTC com um peer via um tracker local
- mensagens de teste entre peers via DataChannel
- software de gestão do mídia no módulo P2P
- intercambio de segmentos via P2P, durante o playback
- estatísticas
- possibilidade de ligar e desligar o P2P durante o playback

De um ponto de vista técnico, a quase totalidade de trabalho foi feita com servidores locais. A migração para servidores externos foi feita em último lugar.

Durante o desenvolvimento, algumas dúvidas surgiram em relação às tecnologias necessárias. Um bom exemplo de escolha técnica difícil e determinante foi de decidir de usar ou o Wrapper da Streamroot, ou desenvolver a partir do zero uma interface com o P2P. Neste tipo de casos, a abordagem foi uma de teste de duração determinada: decidi tentar a primeira opção durante duas semanas. Em caso de sucesso, ia pela frente. Em caso contrário, tentava outra opção, e assim por diante. Provou ser uma abordagem eficiente.

Como foi explicado na seção sobre arquitetura (IV.4.a), o projeto é uma integração de módulos open-source prontos (video.js, hls.js), de módulos open-source modificados (SourceHandler e Wrapper da Streamroot) e de código original com algumas inspirações de outros projetos open-source (módulo P2P). Logo no início do projeto foi decidido deixar estes módulos separados. O controle de versão destes vários módulos foi feito com a ferramenta Git, com um repositório separado para cada módulo. Essa ferramenta permite guardar um histórico do desenvolvimento, ter várias versões de um mesmo código simultaneamente, e retornar a uma versão anterior em caso de erro. Dessa maneira, cada módulo ficou separado e simplificou a cadeia de dependências.

Parte do código foi escrita com sintaxe ES6 (norma ECMAScript 6 para sintaxe de JavaScript), a mais recente, que ainda não é suportada pela maioria dos navegadores web. Portanto, é necessário compilar o código ES6 em código ES5. Da mesma maneira, é necessário transformar o código JavaScript dos diferentes módulos em um único arquivo que contenha tudo o código com as dependências correspondentes. Esta transformação se chama de "Browserificação".

Na prática, tudo isso foi feito com a ferramenta de automação para JavaScript chamada Grunt [**55**], usando os plugins Babel para compilação ES6-ES5 [**56**] e Browserify para browserificação do código [**57**]. Esta ferramenta permite criar comandos de terminal automatizando a compilação, download de dependências, etc. Estes comandos são locais para cada projeto e especificados no arquivo gruntfile.js de cada projeto. A tarefa "Watchify" recompila em tempo real a cada alteração do código, o que permite poupar muito tempo nos ciclos de teste no browser – correção de erros de código – teste.

V. O projeto no cenário do IPTV da USP

O presente projeto de formatura se inscreve na ideia de uma integração com o IPTV da USP. Todos os ganhos já expostos se aplicam da mesma maneira no caso do IPTV da USP: diminuição de custos de banda e melhoria da qualidade de serviço principalmente.

Além disso, o caso particular do IPTV-USP apresenta características que deixam pensar que ele poderia aproveitar ainda mais a integração de um serviço de entrega híbrida de vídeo baseada em WebRTC.

A ideia principal é aproveitar a concentração dos peers em algumas regiões. O IPTV-USP retransmite principalmente palestras e aulas dadas na USP. Isso significa que boa parte dos usuários serão alunos e professores dos vários campi que a USP possui dentro do estado de São Paulo. Levando em conta o fato que proximidade entre peers aumenta o desempenho de um sistema P2P (seção III.4), a arquitetura do presente projeto foi pensada para aproveitar este fato no máximo.

Devemos também considerar a CDN que a USP possui e usa para distribuir o conteúdo entre esses vários campi. Podemos representar uma estrutura simplificada da CDN na Fig. 19. Cada campus possui o seu próprio servidor de borda da CDN para distribuir o conteúdo naquela região.

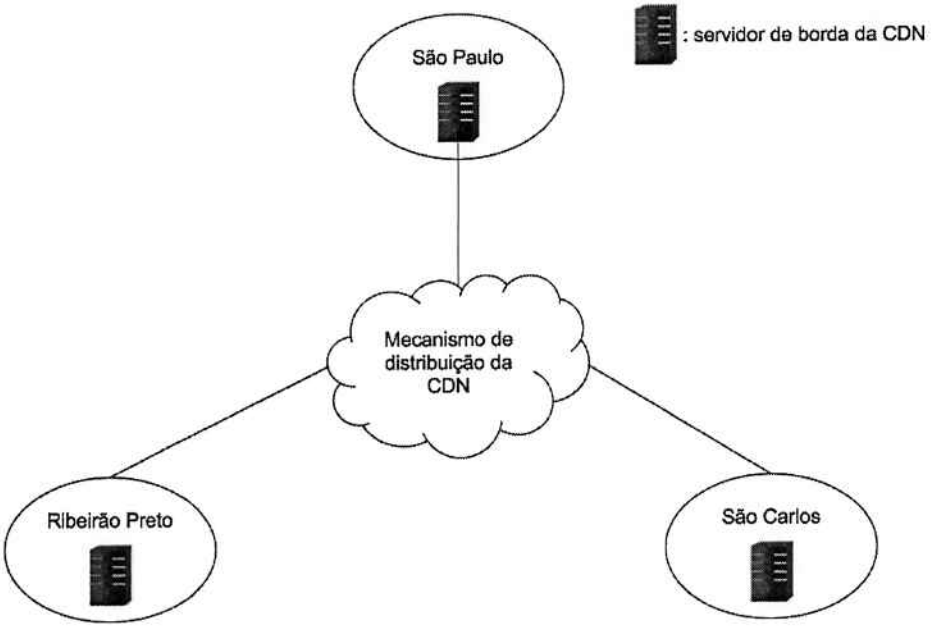

Fig. 19 – Estrutura simplificada da CDN da USP

Com estes dois parâmetros em mente (aumentar a proximidade entre peers e aproveitar a arquitetura da CDN da USP), pensamos em criar um *Tracker* por servidor de

borda em vez de um *Tracker* central para tratar os pedidos de todos os peers. Isso seria fácil com aquela estrutura de CDN, já que temos acesso aos servidores de borda. Basta colocar um tracker em cada um das máquinas, e os usuários daquele servidor de borda serão diretamente direcionados para o *Tracker* correspondente.

Essa arquitetura com vários *Trackers* também tem a vantagem de garantir que os clientes que vão se conectar estarão considerados como próximos pela CDN. A consequência é que dois clientes em cidades distintas não se verão como peers potenciais, dando prioridade para outros clientes da mesma cidade (usando o mesmo servidor de borda), e portanto aumentando o desempenho potencial do P2P.

VI. Demonstração

Nesta seção, vamos apresentar os objetivos e os detalhes técnicos da demonstração do projeto.

Com essa demonstração, o nosso objetivo é demonstrar que:

- O *Streaming* está funcionando (reprodução de um vídeo a partir de um servidor origem);
- Segmentos de vídeo estão sendo trocados entre clientes;
- O sistema é resistente a condições de rede difíceis entre clientes e servidor origem.

Para tal, será feito o streaming do vídeo *Big Buck Bunny* [**58**] na página web do protótipo, hospedada num servidor web Apache do laboratório. A banda entre o servidor origem hospedando o vídeo e o *Access Point* será limitada por um limitador de banda, em um valor inferior à banda necessária para servir todos os clientes.

Num primeiro tempo, a função P2P será desabilitada, e a reprodução deveria apresentar o fenômeno de rebufferização. Num segundo tempo, a funcionalidade P2P será habilitada e a reprodução deveria voltar a ser fluida. Estatísticas como a porcentagem de P2P e o número de peers conectados serão visíveis no quadro de estatísticas de cada cliente.

O dispositivo (Fig. **20**) é constituído por:

- Uma máquina hospedando três servidores
- Um limitador de banda
- Um *Access Point*
- Algumas máquinas clientes

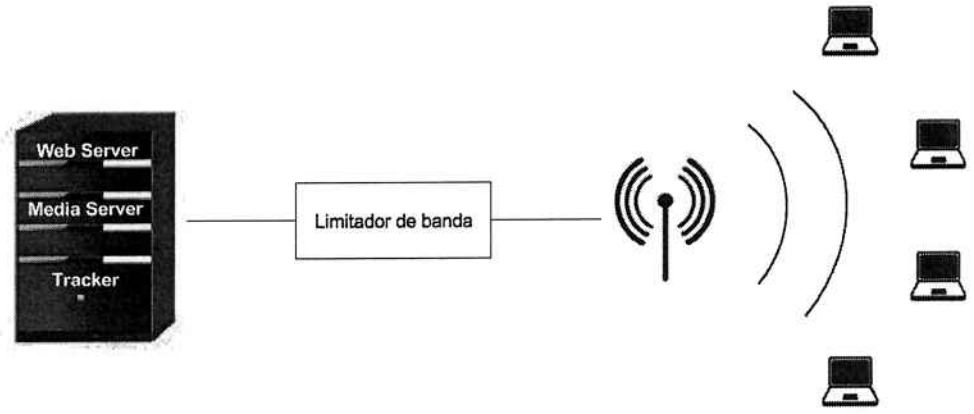

Fig. 20 – Arquitetura da Demonstração

- O primeiro servidor é o Servidor Web, que hospede a página Web do protótipo (servidor Apache, máquina Linux).

- O segundo servidor é o Servidor de Mídia, que hospede os segmentos de vídeo (servidor Apache, máquina Linux).
- Finalmente, o terceiro servidor é o Tracker P2P, apresentado na arquitetura da proposta (servidor Node.js, máquina Linux).

A máquina com os servidores será ligada a um *Access Point* sem fio na sala de demonstração via um link com banda limitada. Várias máquinas clientes se conectarão à rede via este *Access Point* para fazer a demonstração e acessar o vídeo.

Durante a demonstração serão monitoradas as rebufferizações, assim como a porcentagem de conteúdo obtido via P2P para cada cliente.

VII. Testes

Esta seção apresenta os resultados obtidos nos testes realizados. O objetivo desses testes era de confirmar que os segmentos de vídeo estão trocados via P2P, e mensurar as taxas de P2P em função do número de clientes simultâneos.

O ambiente de teste foi o seguinte:

- vídeo Big Buck Bunny [**58**] carregado a partir de uma CDN pública
- navegador Google Chrome 54
- computador MacBook Pro 2011, RAM 4 Go, Intel Core i7 2 GHz
- servidor *Tracker* instalado na máquina localmente
- servidor web instalado na máquina localmente

Parâmetros utilizados para os testes:

- initSegmentsToSkip: número de segmentos inicialmente baixados da CDN sem tentar usar P2P para acelerar o inicio da reprodução.
  Valor usado: 2

- maxStorageChunks: número máximo de segmentos guardados no cache JavaScript por um peer. Quando chegar no limite, o mais antigo é descartado.
  Valor usado: 10

Métrica testada: porcentagem de P2P média, a média das porcentagens individuais.

$$P2P_i = (\text{Bytes baixados em P2P})_i / (\text{Bytes totais baixados})_i$$

Número de clientes simultâneos	Duração do teste	initSegmentsToSkip	Porcentagem média de P2P
2	10 min	2	39%
2	3 min	2	40%
3	3 min	2	53%
3	3 min	1	64%

Tabela 1 – Resultados dos testes

O detalhe de cada teste com as porcentagens individuais de cada peer se encontra no Apêndice A – Resultados detalhados dos testes.

A tendência nos testes é aumentar a porcentagem média de P2P o quanto mais clientes simultâneos tem (Tabela 1). Isso faz sentido porque sempre tem um peer com 0% (o primeiro conectado), então cada peer faz aumentar a média.

Um outro parâmetro importante é o *initSegmentsToSkip*. Em testes de duração curta, ele tem um grande impacto, como vemos nos resultados. Uma mudança de valor de 2 para 1 deu um ganho de porcentagem média de P2P de 11% no caso de 3 clientes, no nosso teste. Podemos deduzir que este parâmetro é muito importante para o

desempenho do sistema, e é necessário achar o valor ideal para ter uma experiência do usuário boa no inicio da reprodução, e ao mesmo tempo um bom desempenho P2P.

Esses resultados são interessantes do ponto de vista do uso de banda nos servidores da CDN. Em comparação, por definição, um sistema sem P2P teria uma porcentagem de P2P média de 0%.

Antes de concluir sobre a viabilidade do sistema, é necessário conduzir testes de mais grande escala para confirmar o desempenho do P2P, e também estudar outros efeitos potencialmente omitidos no estudo que poderiam prejudicar de alguma forma o usuário ou o provedor de serviço.

VIII. Considerações Finais

A presente monografia apresentou o trabalho de formatura sobre um sistema de Web Streaming Híbrido usando WebRTC. Os principais objetivos do projeto eram de especificar e construir o protótipo do produto e demostrar a sua eficiência e os ganhos potenciais que ele poderia trazer para um sistema como o IPTV-USP. Eles foram atingidos, no sentido que um protótipo funcional foi desenvolvido, mostrando a viabilidade da ideia do uso de Peer-to-Peer para entrega de vídeo na Web. Também conseguimos demonstrar os benefícios potenciais na forma de economia de custos de banda proporcional à porcentagem de Peer-to-Peer atingido, e na forma de melhoria da Qualidade de Experiência medida em termos de rebufferização.

Poderia ter sido interessante dar mais um passo a frente, implantando *Trackers* em várias cidades nos servidores de borda da CDN da USP, e testando com vários peers em cada cidade. Isso teria testado a hipótese de melhoria dos resultados do P2P com vários *Trackers* locais em vez de um *Tracker* central só. Esse tipo de trabalho necessitaria mais recursos que os recursos de um aluno no seu projeto de formatura, e seria mais adaptado a um projeto de mestrado por exemplo.

Como projeto de formatura, este projeto conseguiu provar um conceito, e uma futura implementação no serviço de IPTV da USP é uma possibilidade que será estudada no futuro pelo LARC.

IX. Bibliografia

[1] CISCO, "Cisco 2016 Visual Networking Index," 2016.

[2] Kurt Michel. (2013) akamai.com. [Online]. https://blogs.akamai.com/2013/01/live-video-streaming-that-can-handle-traffic-spikes-the-challenge.html

[3] Streamroot, "Peer-assisted adaptive streaming: the key to managing ever- growing online video traffic ," White Paper 2014.

[4] Joel Hruska. (2014) extremetech.com. [Online]. http://www.extremetech.com/computing/186576-verizon-caught-throttling-netflix-traffic-even-after-its-pays-for-more-bandwidth

[5] Presidência da República, "LEI Nº 12.965," vol. Art. 9°, Abril 2014.

[6] Y.-F.R.Chen G. Thompson, "IPTV: Reinventing Television in the Internet Age," *IEEE Internet Computing*, vol. 13, no. 3, pp. 11-14, Maio 2009.

[7] Tankut Akgul, Mark Baugher Ali C. Begen, "Watching videos over the web. Part I: Streaming Protocols," *IEEE Internet Computing*, vol. 15, no. 2, pp. 54-63, March/April 2011.

[8] Jan Ozer, *Video Compression for Flash, Apple Devices and HTML5*, Doceo Publishing, Ed., 2011.

[9] ISO/IEC, "Information technology — Dynamic adaptive streaming over HTTP (DASH) ," 23009-1,.

[10] Apple. developer.apple.com. [Online]. https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/Introduction/Introduction.html

[11] Apple. Example Playlist Files for use with HTTP Live Streaming. [Online]. https://developer.apple.com/library/content/technotes/tn2288/_index.html

[12] Jan Ozer. (2011, November) Streaming Media. [Online]. http://www.streamingmedia.com/Articles/Editorial/What-Is-./What-is-MPEG-DASH-79041.aspx

[13] Ann Malsha De Silva, Yongseng Diao Amal Punchihewa, "Internet Protocol Television (IPTV) ," Multi-media Research Group, School of Engineering and Advanced Technology, Massey University, New Zealand, 2010.

[14] Point Topic, "IPTV Statistics – Market Analysis - Q1 2013," London, 2013.

[15] Jeff Tapper. (2015, March) StreamingMedia. [Online]. http://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=103101&PageNum=1

[16] Natt Garun. (2015, January) The Next Web. [Online]. http://thenextweb.com/google/2015/01/27/youtube-will-now-default-html5-players-better-support-devices/

[17] Srinivasan Seshan, Anees Shaikh Aditya Akella, "An empirical evaluation of wide-area internet bottlenecks," *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pp. 101-114, 2003.

[18] Ingmar Poese, Georgios Smaragdakis, Anja Feldmann, Bruce M. Maggs, Steve Uhlig, Vinay Aggarwal, and Fabian Schneider Benjamin Frank, "Collaboration Opportunities for Content Delivery and Network Infrastructures," *Recent Advances in Networking*, pp. 305-377, July 2013.

[19] Spotify. (2011) P2P Music Streaming. [Online]. http://www.slideshare.net/ricardovice/spotify-p2p-music-streaming

[20] [Online]. http://www.bittorrent.com/

[21] [Online]. https://webtorrent.io/

[22] G., & Vakali, A. Pallis, "Insight and perspectives for content delivery networks," *Communications of the ACM, 49(1), 101-106.*, 2006.

[23] Gang Peng, "CDN: Content Distribution Network," *arXiv preprint cs/0411069.*, February 2008.

[24] Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang Sugih Jamin, "On the placement of internet instrumentation," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE.*, vol. 1, Tel Aviv, 2000, pp. 295-304.

[25] Venkata N. Padmanabhan, and Geoffrey M. Voelker Lili Qiu, "On the placement of web server replicas," in *Proceedings of IEEE INFOCOM 2001 Conference*, Anchorage, 2001.

[26] Yair Bartal, "Probabilistic approximation of metric space and its algorithmic applications," in *37th Annual IEEE Symposium on Foundations of Computer Science*, 1996.

[27] Danny Raz, and Yuval Shavitt P. Krishnan, "The cache location problem," in *IEEE/ACM Transactions on Networking*, vol. 8, 2000.

[28] Cheng Jin, Anthony R.Kure, Danny Raz, and Yuval Shavitt Sugih Jamin, "Contrained mirror placement on the internet," in *Proceedings of IEEE INFOCOM 2001 Conference*, Anchorage, 2001.

[29] M. Ammar, Z. Fei, and S. Bhattacharjee E. Zegura, "Application-layer anycasting: a server selection architecture and use in a replicated web service," in *IEEE/ACM Transaction on Networking*, vol. 8, 2004.

[30] N., Casalicchio, E., & Tucci, S. Bartolini, "A walk through content delivery networks," in *Performance Tools and Applications to Networked Systems.*: Springer Berlin Heidelberg, 2004, pp. 1-25.

[31] Ilya Grigorik, *High-Performance Browser Networking.*: O'Reilly Media, 2013.

[32] webrtc. (2016, Setembro) webrtc.org. [Online]. https://webrtc.org/

[33] (2016, October) NetMarketShare. [Online]. https://www.netmarketshare.com/browser-market-share.aspx?qprid=0&qpcustomd=0

[34] Google. (2014, June) Google +. [Online].

https://plus.google.com/u/0/103171586947853434456/posts/39TCW3PcLye

[35] Mozilla. (2014, December) blog.mozilla.org. [Online]. https://blog.mozilla.org/press-fr/2014/12/05/hello-communiquer-plus-simplement-grace-au-navigateur-firefox/

[36] Skype. (2016, July) blogs.skype.com. [Online]. https://blogs.skype.com/2016/07/13/skype-for-linux-alpha-and-calling-on-chrome-and-chromebooks/

[37] https://www.streamroot.io/.

[38] https://www.peer5.com/.

[39] http://viblast.com/.

[40] Feross Aboukhadijeh. http://feross.org/resume/.

[41] Kylie Jue. (2014, January) stanforddaily.com. [Online]. http://www.stanforddaily.com/2014/01/08/yahoo-purchases-alumni-startup-peercdn/

[42] Adam Bergkvist, Cullen Jennings, Anant Narayanan, Bernard Aboba Daniel C. Burnett. (2016, June) Media Capture and Streams Specification, W3C Editor's Draft. [Online]. https://w3c.github.io/mediacapture-main/#toc

[43] Tsahi Levent-Levi. (2015, June) https://webrtchacks.com. [Online]. https://webrtchacks.com/webrtc-and-man-in-the-middle-attacks/

[44] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," IETF, RFC5245 ISSN 2070-1721, 2010.

[45] Yossi Zada. (2013, October) [Online]. http://fr.slideshare.net/Audiocod/nat-traversal-in-webrtc-context

[46] Mikhal Shemer, Marco Paniconi Stefan Holmer, "Handling Packet Loss in WebRTC," 2013.

[47] Steve Jobs. (2010, April) http://www.apple.com/. [Online]. http://www.apple.com/hotnews/thoughts-on-flash/

[48] CVE. cvedetails.com. [Online]. https://www.cvedetails.com/vulnerability-list/vendor_id-53/product_id-6761/Adobe-Flash-Player.html

[49] Nicolas Weil. (2016, March) Streaming Media Magazine. [Online]. http://www.streamingmedia.com/Articles/Articles/Editorial/Featured-Articles/The-State-of-MPEG-DASH-2016-110099.aspx

[50] Streamroot. [Online]. https://github.com/streamroot/videojs5-hlsjs-p2p-source-handler

[51] Video.js. [Online]. http://videojs.com/

[52] Mangui. [Online]. https://github.com/dailymotion/hls.js

[53] W3C. (2016, August) [Online]. http://w3c.github.io/media-source/

[54] Streamroot. [Online]. https://github.com/streamroot/hlsjs-p2p-wrapper

[55] (2016, October) [Online]. http://gruntjs.com/

[56] [Online]. https://babeljs.io/

[57] (2016, October) [Online]. http://browserify.org/

[58] Big Buck Bunny. [Online]. https://peach.blender.org/

[59] J. K., Meyn, A. J., Jalonen, E., Raivio, Y., & Garcıa Marrero, R. Nurminen, "P2P media streaming with HTML5 and WebRTC.," *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference*, pp. 63-64, April 2013.

[60] DVB, "Digital Video Broadcasting (DVB); MPEG-DASH Profile for Transport of ISO BMFF Based DVB Services over IP Based Networks," 2016.

[61] Streamroot. Github. [Online]. https://github.com/StreamRoot

[62] P. Srisuresh G. Tsirtsis, "Network Address Translation - Protocol Translation (NAT-PT)," RFC 2000.

[63] Jan Ozer. (2011, October) Streaming Media Magazine. [Online]. http://www.streamingmedia.com/Articles/Editorial/What-Is-./What-is-HLS-(HTTP-Live-Streaming)-78221.aspx

[64] ITU. (2016, July) ITU announces new standard for High Dynamic Range TV. [Online]. http://www.itu.int/en/mediacentre/Pages/2016-PR27.aspx

X. Apêndice A – Resultados detalhados dos testes

Teste 1

- 2 peers
- playback 10 minutos
- initSegmentsToSkip = 2
- maxStorageChunks = 10

	Peer 1	Peer 2
%P2P	0%	78%

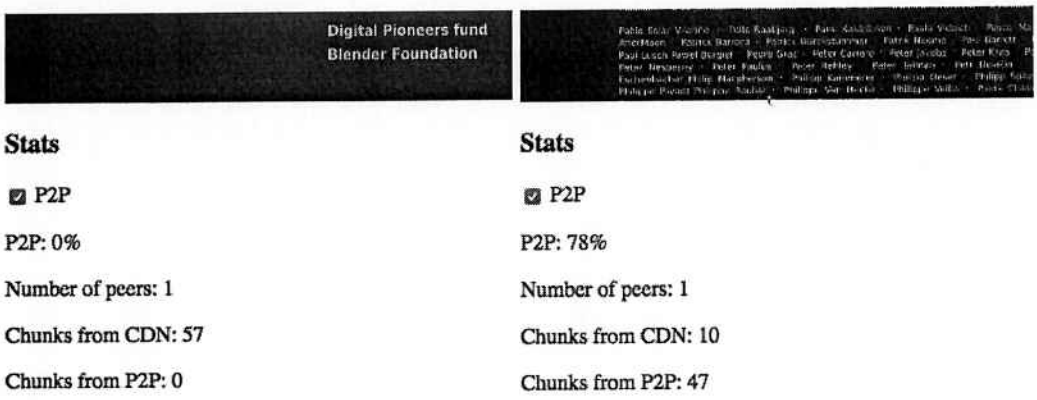

Fig. 21 – Print de tela do teste 1

Teste 2

- 2 peers
- playback 3 minutos
- initSegmentsToSkip = 2
- maxStorageChunks = 10

	Peer 1	Peer 2
%P2P	0%	80%

Teste 3

- 3 peers
- playback 3 minutos
- initSegmentsToSkip = 2
- maxStorageChunks = 10

	Peer 1	Peer 2	Peer 3
%P2P	0%	86%	73%

Teste 4

- 3 peers
- playback 3 minutos
- initSegmentsToSkip = 1
- maxStorageChunks = 10

	Peer 1	Peer 2	Peer 3
%P2P	0%	97%	96%