Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estratégia de web cache utilizando redes P2P de clientes sobre WebRTC

Carlos Botelho de Paula Filho

Brasília
2016

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estratégia de web cache utilizando redes P2P de clientes sobre WebRTC

Carlos Botelho de Paula Filho

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. Ricardo Pezzuol Jacobi

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programa de Pós-Graduação em Informática

Coordenador: Prof. Dr. Célia Ghedini Ralha

Banca examinadora composta por:

Prof. Dr. Ricardo Pezzuol Jacobi (Orientador) — CIC/UnB
Prof. Dr. Eduardo A. P. Alchieri — CIC/UnB
Prof. Dr. Ugo Dias — ENE/UnB

CIP — Catalogação Internacional na Publicação

Filho, Carlos Botelho de Paula.

Estratégia de web cache utilizando redes P2P de clientes sobre WebRTC / Carlos Botelho de Paula Filho. Brasília : UnB, 2016.
62 p. : il. ; 29,5 cm.

Tese (Mestrado) — Universidade de Brasília, Brasília, 2016.

1. Caching inteligentes, 2. Web, 3. P2P, 4. Torrent, 5. WebRTC

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estratégia de web cache utilizando redes P2P de clientes sobre WebRTC

Carlos Botelho de Paula Filho

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof. Dr. Ricardo Pezzuol Jacobi (Orientador)
CIC/UnB

Prof. Dr. Eduardo A. P. Alchieri — CIC/UnB
Prof. Dr. Ugo Dias — ENE/UnB

Prof. Dr. Célia Ghedini Ralha
Coordenador do Programa de Pós-Graduação em Informática

Brasília, 18 de novembro de 2016

Dedicatória

Dedico este trabalho a minha família e amigos, pois sem suporte não teria conseguido obter sucesso nesta empreitada.

Agradecimentos

Gostaria de agradecer a minha família e amigos pelo suporte. Além de meu orientador.

Resumo

Cache na internet é um mecanismo chave para o aumento de performance e estabilidade de sites em momentos de grande número de acessos. Entretanto a maioria das estratégias de *cache* não levam em conta o poder do *cache* localizado entre nós clientes. Novas tecnologias desenvolvidas para a nova versão da especificação do HTML, o HTML5 possibilitaram que nós se conectem e interajam entre si, possibilitando a troca de dados sem a instalação de *plugins* ou *addons* de navegadores. Este trabalho descreve uma estratégia de *cache* que cria uma rede *P2P* de clientes agindo como servidores de *proxy cache* assim que os mesmos fizerem a primeira requisição ao servidor, distribuindo os dados potencialmente entre todos os visitantes do site. Aumentando a disponibilidade do site em picos de acesso, e minimizando drasticamente os custos de infraestrutura, e como os nós irão prover conteúdo dentro das redes dos provedores de internet, a estratégia pode potencialmente aumentar a velocidade de abertura de página ou de conteúdos de mídia. Assim como uma diminuição no custo de transferência de dados nos provedores de internet.

Palavras-chave: Caching inteligentes, Web, P2P, Torrent, WebRTC

Abstract

Web caching has been a key player in increasing performance and website stability in times of heavy usage. However most caching strategies do not take into account the power of localized caching beetween client nodes. New technologies developed for the new version of HTML specification, HTML5 have enabled nodes to connect and interact with each other, enabling them to share data without the addition of browser plugins or addons. This work describes a caching strategy that creates a P2P network of clients that act as proxy cache servers as soon as they issue a request to the server, distributing the caching data potencially across all web site visitors. Increasing the web site availability in high peaks, minimizing drastically the costs of infraestructure and, since the nodes will provide content to localized clients within their ISP networks, the strategy will potentially result in a faster overall spead in page loads, or in media content loading. As well as decreasing costs for ISPs due to the minimization of data exchange outside of the ISP network.

Keywords: WebRTC, Distributed caching, Web caching , Caching, P2P

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Desde o inicio da computação a performance sempre foi uma grande preocupação. Alguns recursos de um sistema computacional demandam mais tempo para executar e com isso podem afetar a performance do sistema como um todo. Para melhoria de velocidade ou para melhor utilização de recursos, alguns acessos e serviços cujo custo operacional ou temporal eram altos começaram a ser armazenados em locais menos custosos ou de acesso mais eficiente. Tornando estes dados disponíveis em um tempo menor na próxima consulta. O termo em inglês *cache* define esta estratégia. Ao se utilizar de uma *cache* um sistema torna futuros acessos a um item que tenha sido colocado na *cache* mais rápidos.

Na internet a *cache* é essencial para que melhores velocidades e maiores quantidades de dados trafegados sejam atingidas. Quando uma página é acessada por um computador, uma grande quantidade de dados tem que ser transmitida pela rede, muitas vezes acessando servidores que estão fisicamente distantes, por isso os navegadores de internet tem mecanismos de *cache* pré instalados. Com esses mecanismos os próximos acessos a essa página poderão utilizar itens que estejam armazenados localmente, evitando um número grande de requisições. Para tirar vantagem desse mesmo mecanismo mas em um nível acima os provedores de internet *Internet Service Providers ISPs* também se utilizam de estratégias para realizar a *cache*, alguns dos mecanismos mais utilizados incluem: servidores *proxy*, onde o conteúdo de páginas com grande número de acesso é armazenado localmente à sub rede, e os próximos acessos retornam o conteúdo deste servidor, não chegando a chamar o servirdor orignial, e redes de distribuição de conteúdo *Content delivery networks (CDNs)* que são redes pagas interconectadas e distribuídas pelo mundo, uma página que tenha acessos em vários países diferentes pode se utilizar desse mecanismo para tornar o acesso ao seu conteúdo mais local, evitando os pulos internacionais, que tem uma latência alta.

Contudo, à medida que a internet foi crescendo, a natureza do conteúdo acessado também entrou em constante mutação. No início os conteúdos eram produzidos por equipes como portais de notícias ou sites pessoais, com grandes conteúdos estáticos, mas com o crescente uso de redes sociais, o conteúdo está cada vez mais sendo gerado e consumido por milhares de usuários, diminuindo a natureza centralizada de anos anteriores.[3]. O conteúdo está cada vez mais rico de multimídias como imagens, áudios e vídeos pessoais. Além disso as antigas páginas com vasto conteúdo estático estão dando lugar a aplicações complexas que tentam prender a atenção do usuário dentro de sua página pelo maior tempo possível, visto que a maiora dos aplicativos tem sua fonte de receita derivada de

publicidade, e quanto maior o tempo de um usuário em um site, maior será o valor proveniente da publicidade exibida. Nestas novas aplicações é comum a utilização de mecanismos que criam a sensação de que o conteúdo é infinito, sendo buscado automaticamente e de forma assíncrona cada vez que o usuário move a barra de rolagem. Estas mudanças na maneira como o conteúdo é gerado e exibido para os usuários cria novos desafios para os mecanismos tradicionais de *cache*.

Além do conteúdo textual e de imagens provenientes de páginas e emails, a internet hoje possibilita o consumo massivo de conteúdos de mídia, como arquivos de som e vídeo. Em um estudo recente [13] estimou-se que a utilização de serviços de vídeo equivalem a 21% do trafego total da internet nos Estados Unidos. Para efeito de comparação todo o trafego associado a navegação na internet equivale a 18%. Nesse cenário é natural que discussões sobre o peso que os conteúdos audiovisuais têm sobre a carga total da internet criem discussões entre provedores e aplicativos.

O problema se dá na maneira como os *ISPs* se conectam à rede mundial. Em sua maioria, os provedores têm infraestruturas locais que conectam os computadores de uma região entre si *Wide Area Networks WAN*, e para acessar a rede mundial, os provedores fecham parcerias com os seus provedores de *uplink*, estas parcerias envolvem custos por dados trafegados. Assim quanto maior for a quantidade de dados trafegados para fora das redes dos *ISPs* maior será o custo associado ao conteúdo sendo transmitido. Pode-se imaginar que vídeos representam grandes cifras nos custos dos provedores.

Este trabalho busca propor um mecanismo de *cache* que se utilize das características atuais de como o conteúdo é gerado e consumido, levando em conta o impacto que acessos fora das redes dos *ISPs* tem no custos do acesso, tanto em tempo quanto em valor financeiro.

1.1 Problema

Os algoritimos e mecanimos de *cache* atuais não se utilizam das características de como os dados são produzidos e consumidos atualmente. A arquitetura atual dos *ISPs* tem uma estrutura localizada que não é aproveitada em sua plenitude visto que a configuração dos mecanismos de *cache* atuais se aplicam em sua maioria fora do escopo local das redes dos *ISPs* aumentando o custo total dos acessos.

1.2 Objetivo

O objetivo deste trabalho é projetar uma estratégia de mecanismos de *cache* que se aproveite da característica diferenciada de como os dados são gerados e consumidos nos tempos atuais, levando em conta as estruturas de arquitetura das redes que conectam os *ISPs* de forma a potencializar o trafego de dados dentro das sub-redes internas dos *ISPs* diminuindo o trafego externo ao ISP, mais lento e mais custoso que o local.

1.3 Objetivos específicos

- Levantamento do estado da arte de web cache sobre P2P

- Definir uma proposta de arquitetura de proxy cache sobre P2P
- Implementar esta solução com o auxílio de um protocolo de redes P2P já estabelecido
- Implementar esta solução para que rode nativamente nos navegadores web, sem a necessidade de extensões de navegadores ou programas externos.
- Implementar um caso de uso da arquitetura em um cenário ilustrativo
- Análise dos resultados coletados.

1.4 Organização do trabalho

O restante deste trabalho é organizado da seguinte forma:

- o Capítulo 2 introduz conceitos básicos relativos a caching e relembra terminologias e princípios de caching
- o Capítulo 3 descreve a atual estrutura de conexão dos provedores de internet e sua relevância nos modelos de *cache*.
- o Capítulo 4 apresenta conceitos básicos de Redes *P2P* e suas características mais relevantes para serem utilizadas na composição de uma estratégia de *cache*e também apresenta conceitos básicos de redes de *cache* colaborativas e o seu estado da arte.
- o Capítulo 5 apresenta as adições ao protocolo HTTP e a especificação do WebRTC.
- o Capítulo 6 descreve a solução proposta de estratégia de web *cache* utlizando redes *P2P* sobre *WebRTC* , e exemplifica a solução adotada na implementação deste trabalho A seção 6.4 descreve os detalhes da solução implementada mais a fundo como mecanismos de captura de imagens, e características específicas do protocolo torrent envolvidas no processo
- o Capítulo 7 apresenta o caso de uso e cenários de utilização da proposta, bem como uma avaliação quantitativa da mesma com relação as soluções atuais;
- e algumas considerações finais são apresentadas no Capítulo 8.

Capítulo 2

Referencial teórico - cache

Neste capítulo serão abordados conceitos referentes a *cache*, apresentando conceitos básicos e a terminologia utilizada na área. Também serão listados exemplos de estratégias de *cache* utilizadas na internet. No final serão detalhados dois casos de uso reais de utilização de *cache* em um grande portal brasileiro.

2.1 Cache

O computador é composto por diversos componentes que tem tempos de execução distintos para cada uma de suas tarefas. Assim um componente pode demorar mais que outro, o que pode acabar criando gargalos de performance que podem ser otimizados. O objetivo desta otimização é tentar diminuir o tempo de acesso médio para melhorar a performance geral do componente. No caso, operações de entrada e saída costumam ter os maiores tempos de acesso, e frequentemente utilizam-se de estratégias para aumentar a performance do acesso a leitura/escrita de dados. A *cache* é o local onde um dado é armazenado temporariamente para que futuros acessos sejam mais eficientes, é uma estrutura intermediária e limitada em tamanho, portanto a utilização de bons algoritmos de *cache* é essencial para que a performance seja a melhor possível.

2.1.1 Princípio da localidade

Os dados utilizados por programas em um computador possuem características importantes, que podem ser utilizadas para a melhoria da performance. Até mesmo o código dos programas por exemplo tende a ter seus dados lidos durante a execução de maneira localizada. Os programas normalmente são executados em trechos de códigos contíguos, mesmo que não imediatamente em sequência, mas com uma concentração grande em posições de memória próximas. Existem dois tipos de localidades:

- **Localidade espacial**: É atribuida a proximidade de endereçamentos de memória, ou disco no acesso aos dados. Por estatística existe uma grande tendência de que dados próximos ao que estão sendo lidos atualmente sejam lidos em sequencia.

  No caso da *cache* aplicada na internet, a localidade espacial pode ser atribuída a dados, sejam eles imagens, vídeos ou dados fisicamente ou logicamente próximos a área exibida em tela tem a mesma tendência a serem utilizados em sequência ao

conteúdo atual. Por isso muitos navegadores utilizam-se da estratégia de antecipar acessos de leitura, fazendo o chamado *read-ahead*, onde conteúdos são baixados antecipadamente, para melhorar a velocidade aparente de acesso a uma página.

- **Localidade temporal**: É atribuida a necessidade de dados serem reutilizados em relação ao tempo. Um dado lido a poucos mili-segundos atrás tem maior probabilidade de ser lido novamente do que se comparado ao resto dos endereços de memória disponíveis. Na *cache* da internet, o mesmo comportamento é amplamente visto. Há uma grande tendência de que um conteúdo em uma página seja acessado novamente, sejam eles imagens ou dados, alguns recursos estão presentes em várias páginas e por isso os navegadores utilizam essa futura necessidade de acessar os itens como justificativa para salvá-los na máquina.

Quando utilizados em conjunto a melhoria de performance pode ser significativa, mas o conteúdo deve também ser otimizado. Em códigos por exemplo os compiladores tendem a agrupar operações repetitivas e dados muito acessados para que essa carecterística seja melhor aproveitada. Na internet os desenvolvedores devem priorizar as marcações de conteúdos que serão utilizados com maior frequência, para que os navegadores façam o uso da *cache* de maneira mais eficiente. Esta marcação é feita utilizando-se de cabeçalhos específicios que serão abordados na seção 2.4.3

2.1.2 Tipos de mapeamento de memória para cache

A estratégia de se utilizar uma hierarquia de acesso para a memória foi implantada para se aproveitar de estruturas ou dispositivos mais rápidos , porém normalmente por serem mais rápidos e caros, costumam ser mais escassos. Nesse caso é necessário realizar alguma forma de mapeamento entre o dado em uma estrutura mais rápida e sua posição original na menos rápida. Como exemplo de hierarquia de acessos: é comum que assim que dados de arquivos que estão no disco rígido (com grande volume de armazenagem) sejam acessados frequentemente , os mesmos sejam colocados em memória RAM (com tamanhos consideravelmente menores que os discos rígidos) e possivelmente na memória *cache* dos processadores (que costuma ter alguns *mega bytes*).

Existem algumas formas de se realizar este mapeamento:

- **Mapeamento direto**: É a forma mais simples de mapeamento, onde a posição da *cache* depende do endereço do dado na memória principal. Com a utilização de um *bit* de validade, pode-se saber que o dado naquela posição não está preenchido e deve ser lido da memória principal. Nessa abordagem a memória principal é dividida em áreas e essas áreas, em linhas, assim o endereço original é desmembrado em *tag* e linha, que faz o mapeamento com as posições finais.

  Uma pequena variação desse mapeamento é a utilização de blocos, onde uma maior quantidade de dados é lida e armazenada em posições adjacentes na linha. Justamente para se aproveitar mais do princípio da localidade.

  As vantagens dessa técnica são a simplicidade da procura. A desvantagem é o uso de parte da cache para controle, e o *miss rate* , que é a taxa de itens não encontrados na *cache* pode ser alta.

- **Mapeamento associativo** Como o mapeamento direto faz a tradução direta dos endereços pela posição, mesmo tendo espaço na *cache*, pode-se ocorrer o *cache miss*, que é o ato de não encontrar um item na *cache*. Para isso no mapeamento associativo, a tag não fica mais na *cache* e sim em outra estrutura especial, a memória associativa. Porém com isto deve haver uma política de substituição, e o dado deve ser procurado em outra estrutura antes de ser lido. As políticas de substituição serão abordadas na próxima seção.
- **Associativa por conjunto** Para se utilizar de características dos dois mapeamentos acima, a associação por conjunto busca atingir a utilização de toda a *cache*, porém sem a utilização das comparações que eram necessárias na utilização do mapeamento associativo. Uma parcela dos bits é destinada a mostrar qual bloco de memória deve ser acessado, através de apenas um decodificador.

No caso da *cache* para a internet, normalmente os navegadores tem limites grandes para armazenamento de dados localmente. Mas fazendo uma associação para fins de comparação entre o acesso a dados na internet com o acesso a arquivos em um computador normal, podemos considerar os arquivos e dados nos servidores como estando na memória *RAM*, e no computador do usuário como sendo na memória *cache*. Neste cenário, o mapeamento utilizado seria o direto, utilizando-se o nome completo do arquivo, com o endereço do site, seu caminho relativo e o nome de arquivo com a extensão como sendo a associação aos dados à posição original de memória RAM(os servidores).

2.1.3 Algoritmos de substituição de dados na cache

Como o armazenamento na *cache* é limitado , muitas vezes deve-se decidir quais dados devem ser retirados da *cache* para que novas entradas sejam aramazenadas. A seguir seguem algumas dessas políticas:

- *LRU - least recently used*: Menos utilizada recentemente - o dado que está armazenado a mais tempo é substituido, precisa-se de uma variável de tempo para realizar esta escolha.
- *FIFO - first in first out*: O primeiro a entrar é o primeiro a sair - utiliza-se um contador baseado em relógio que aponta a próxima posição a ser excluída.
- *LFU least frequently used*: Menos frequentemente utilizada - dado que é menos utilizado é substituido, precisa-se de uma variável para contar os números de acessos.
- Escolha alatória: O dado é escolhido aleatóriamente. Simples mas pode elevar a chance de um *cache miss*

Na internet os navegadores costumam utilizar-se dos cabeçalhos específicos para *cache* para saber quais itens não são mais válidos. Porém a memória disponível costuma ser grande, e cada navegador pode implementar um algoritmo de substituição caso o espaço esteja excedido. Esta informação varia de acordo com o navegador.

2.1.4 Políticas de escrita

Ao se escrever um dado no dispositivo final, deve-se pensar como deve ser feita a sobreposição da cópia do dado que esteja em *cache*. Asseguir seguem políticas de escrita comumente utilizadas:

- *Write through* : A escrita é feita em ambos os locais antes de ser marcada como finalizada para o executor. Assim o acesso mais lento ditará o tempo total do processo de escrita.
- *Write around*: Usa uma técnica similar a de *write through* mas escreve apenas diretamente no dispositivo final, não escrevendo na *cache*.
- Write back: Usa uma técnica oposta ao *write around*, o dado é escrito diretamente na *cache* e confirmado para o executor. E os dados são escritos no dispostivo final posteriormente de forma assíncrona.

No caso de atualizações de dados nos servidores na internet, a cache pode ser substituída ao utilizar-se de alguns mecanismos que serão explicados posteriormente como *E-TAGs* para sinalizar que o dado foi alterado. Nesse caso ou no caso do recurso literalmente mudar seu nome, os dados são atualizados na *cache*.

2.2 Cache na internet

Cache é o ato de salvar respostas reutilizáveis para tornar requisições futuras a estas respostas mais rápidas. Na internet o *cache* é tão importante que está definido dentro da especificação do protocolo HTTP [23]. Desta forma, para que o tráfego seja minimizado e para melhorar a percepção de responsividade do sistema como um todo, são utilizadas *cache* em vários níveis durante a jornada do conteúdo dos servidores até o navegador dos clientes. Utilizando os cabeçalhos de meta dados dentro do protocolo HTTP os conteúdos são armazenados fazendo com que requisições subsequentes possam ser realizadas localmente, ou em menos pulos do que na requisição original.

Alguns dos benefícios de se utilizar estratégias de *cache* são :

- Diminuição de custos de rede. Dependendo de onde o conteúdo vem, normalmente quanto mais próximo ao cliente menor o custo de transporte do dado.
- Melhoria na responsividade. A velocidade total da requisição não afeta necessariamente a percepção da mesma para o usuário, entretanto *cache* de navegadores por exemplo, podem transparecer que o acesso ao conteúdo foi instantâneo.
- Melhoria de performance no mesmo hardware. Sem a necessidade de se aumentar a potência ou a quantidade de memória tanto no servidor quanto no cliente, pode-se ter uma performance melhorada pois menos recursos precisarão ser transmitidos.
- Disponibilidade de conteúdo durante falhas na rede. Com algumas políticas de *cache* pode-se ter sites funcionais mesmo quando descontectados de redes ou sem acesso a internet.

2.3 Terminologia

Alguns termos são muito utilizados ao se referir a *cache*, segue abaixo uma lista dos mais comuns e que serão utilizados neste trabalho:

- **Servidor de origem** O servidor original do conteúdo. Ele é o servidor responsável por servir o conteúdo que não será transmitido da *cache*
- **Razão de acerto de cache** ou *Cache hit ratio* É um medida de eficência de *cache*, que traz um percentual de requisições que foram lidas da *cache* sobre o numero total de requisições feitas.
- **Freshness** é o termo que se refere ao intervalo de tempo que deve ser considerado para que o conteúdo na *cache* ainda seja considerado válido. Esse número é geralmente definido por campos de cabeçalho do protocolo HTTP.
- **Conteúdo expirado** é justamente o conteúdo cujo *freshness* ou seu tempo de vida já foi atingido, e que deve ser requisitado novamente ao servidor.
- **Validação** é o processo onde se faz a verificação acerca do conteúdo em *cache* se ele ainda é a versão mais nova do recurso.
- **Invalidação** é o processo de remover o conteúdo da *cache* antes de seu tempo final de expiração. Isto é necessario quando o conteúdo mudou no servidor de origem.

2.4 O que , onde e como?

2.4.1 O que deve ser cacheado

As páginas na internet já evoluiram bastante desde o início da rede. De simples páginas textuais , páginas com texto e imagens a aplicativos poderosos e complexos. Desta tamanha complexidade de recursos surge a dúvida sobre qual conteúdo pode ser passível de *cache*. O mais comum é fazer o *cache* dos seguintes recursos:

- Imagens
- Arquivos HTML
- Videos
- Scripts javascript
- Documentos de estilo CSSs
- Requisições de dados assíncronas

2.4.2 Onde pode-se fazer a cache

O conteúdo pode ser passível de *cache* em vários níveis (figura 2.1) na cadeia de entrega de conteúdo:

- No navegador. Todos os navegadores tem uma pequena *cache* tipicamente guardando conteúdos grandes como imagens e videos.
- Em *Servidores de proxy cache* intermediários, entre um servidor de origem e o cliente costumam existir servidores proxy de *cache* que são mantidos pelos próprios provedores de internet ou mesmo pelos administradores da rede local sendo utilizada pelo cliente.
- Antes do servidor, em uma *cache* reversa. Normalmente os servidores se utilizam de camadas de cache antes de chegarem diretamente aos servidores de origem.
- *CDNs - content delivery networks* ou redes de distribuição de conteúdo. Ao invés de direcionar o usuario ao servidor de origem, utiliza-se redes distribuidas geograficamente mais próximas do usuario para prover os dados solicitados.

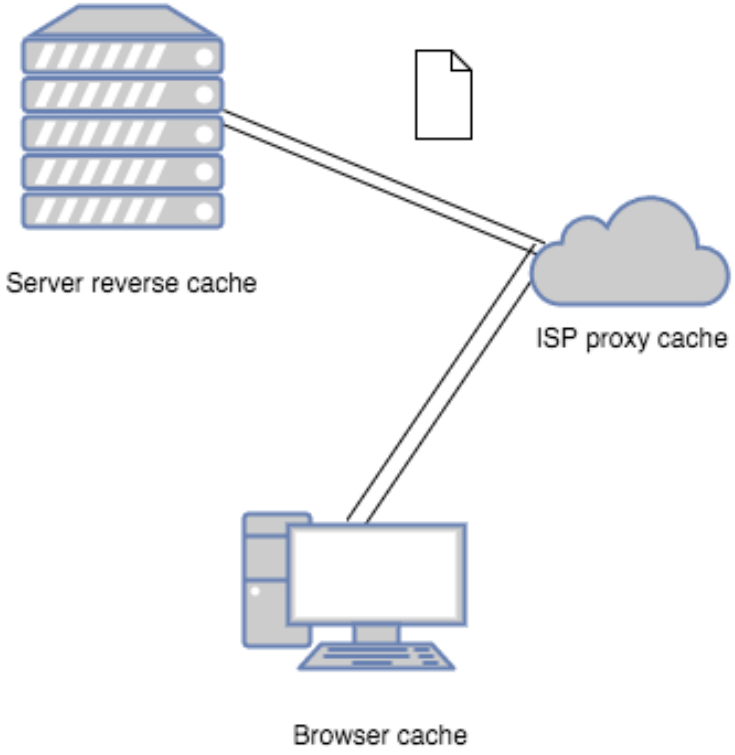

Figura 2.1: Locais onde a cache pode ocorrer

2.4.3 Como fazer a cache

Para sinalizar que o conteúdo deve ser passível de *cache* existem no protocolo HTTP [23] cabeçalhos que informam vários aspectos relacionados ao conteúdo, abaixo segue uma lista de alguns cabeçalhos relevantes para esta proposta:

- **Expires** o cabeçalho *expires* define uma data no futuro que deve ser o limite para que o conteúdo seja considerado válido, após isso ele deve ser descartado e o item deve ser requisitado ao servidor.

- **Cache-control** define politicas diferenciadas de *cache*, isto será melhor comentado abaixo.
- **Etag** é uma etiqueta única que define o conteúdo específico, assim para validar se o item continua válido basta perguntar ao servidor de origem se o item tem o mesmo *ETAG*, se tiver isto indica que o conteúdo do item não mudou.
- **Last-modified**: indica a data de última modificação, pode ser usado em conjunto para garantir a utlização de versões mais recentes.
- **Content-Length** não é definido especificamente para *cache* mas pode ser usado na alocação correta de espaço de armazenamento.

O campo *Cache-control* tem indicadores importantes, para definir como o *cache* do item deve ser tratado. Cada tipo de valor neste campo define políticas diferentes:

- **no-cache** define que o conteúdo não deve ser passível de *cache*. Isso automaticamente marca o conteúdo como expirado, forçando sua requisição todas as vezes.
- **no-store** define que o conteúdo não pode ser passível de*cache* de maneira alguma. É utilizado para dados sensíveis.
- **public** Marca o conteúdo como público, habilitando assim o *cache* tanto pelo navegador como por todos os servidores de *cache* intermediários.
- **private** Marca o conteúdo como privado, somente podendo ser passível de *cache* pelo navegador do usuário.
- **max-age** define que o conteúdo deve ser revalidado em segundos , este item substitui o campo *expires*.
- **s-maxage** similar ao *max-age* porém é utilizado por servidores de *cache* intermediários.
- **must-revalidate** indica que os campos definidores de data devem ser utilizados obrigatóriamente, como o *max-age*
- **proxy-revalidate** o mesmo do *must-revalidate* mas somente para servidores intermediários
- **no-transform** Avisa a *cache* que não podem alterar o conteúdo sobre nenhuma circunstância. O conteúdo não pode ser comprimido por exemplo.

Vale salientar que alguns destes valores são mutuamente excludentes:

- *no-cache* e *no-store* se estiverem presentes não causam o *cache*
- Enquanto que *public* e *private* causam.

Grande parte desses artifícios de *cache* foram definidos em um momento quando o conteúdo era criado por poucos usuários e hospedado em servidores, contúdo nas últimas decadas o conteúdo é criado em massa pelos próprios usuários , e graças as conectividades sociais providas pelas plataformas de redes sociais, consomem o mesmo conteúdo, tornando a eficiência dos mecanismos de *cache* cada vez pior.[3]

2.5 Outras estrategias

Além da estratégia pura e simples de se realizar o *cache* de conteúdo, existem outras áreas de estudo para a melhoria na performance das comunicações na internet que serão listadas abaixo:

Content prefetch Se refere ao estudo de realizar o *download* de forma antecipada, antes do usuário necessariamente requisitar o conteúdo explicitamente. Alguns navegadores realizam o download automatico de páginas inteiras que por estatísticas costumam ser acessadas atraves da pagina atual. Existem várias pesquisas sobre esse tema[14].

Download em paralelo Se refere ao estudo de maneiras para realizar o *download* do conteúdo em paralelo. Uma página costuma ter vários recursos que devem ser exibidos ao mesmo tempo, por padrão os navegadores costuma baixar até três itens simultaneamente [23], pois os servidores *web* costumam limitar a esse número, porque esses acessos randômicos ao mesmo tempo podem gerar muitos acessos ao disco de forma desordenada sendo uma possível brecha para um ataque de negação de serviço. Para tentar melhorar o paralelismo, normalmente usa-se sites espelhos em domínios distintos.[5]

Web proxy Se refere ao estudo de estratégias entre os servidores *proxy* e os servidores de conteúdo, como realizar atualizações em massa, como distribuir esses servidores geograficamente ou topologicamente.[16][18][9][15][4][26][17]

P2P web proxy Se refere inicialmente a estratégia de conectar os servidores de *proxy cache* em uma rede P2P para se beneficiar das facilidades da rede para a atualização do conteúdo de maneira altamente distribuida e descentralizada [7].

2.6 Caso de uso real

Para que se possa ter uma visão da utilização da cache no mundo real, segue abaixo a análise do conteúdo de um grande portal brasileiro. A escolha de portais é baseada no volume de acessos dos mesmos, e também pela rica presença de recursos passíveis de *cache*.

2.6.1 Análise da cache do site uol.com.br

Para realizar a análise do conteúdo e da *cache* foi utilizada a ferramenta *Developer tools* dentro do navegador *Google Chrome*.

Antes de realizar o primeiro acesso ao site, o navegador teve todos os seus dados de *cache* limpos.

Os grandes portais tem um número alto de conteúdo, seja em texto ou em imagens, é comum a utilização de otimizações para que a página abra mais rapidamente. A mais comumente utilizada é a de *lazy loading* de conteúdo. Esta técnica apenas requisita o conteúdo quando o mesmo está próximo (gráficamente localizado) a atual região sendo exibida ao usuário. Por isso na análise do site uol.com.br , é possível salientar dois momentos distintos, um ao acessar o site e outro ao exibir o site por completo.

Primeiro acesso - imóvel

Após o primeiro acesso, sem mover a posição original da tela, os dados trafegados e o número de requisições podem ser visualizados na figura 2.2. No caso o tamanho atual da página é de 2.7 MB, com 236 requisições. O total de segundos transcorridos desde a requisição até o conteúdo ser desenhado na tela do usuário é de 8.95 segundos.

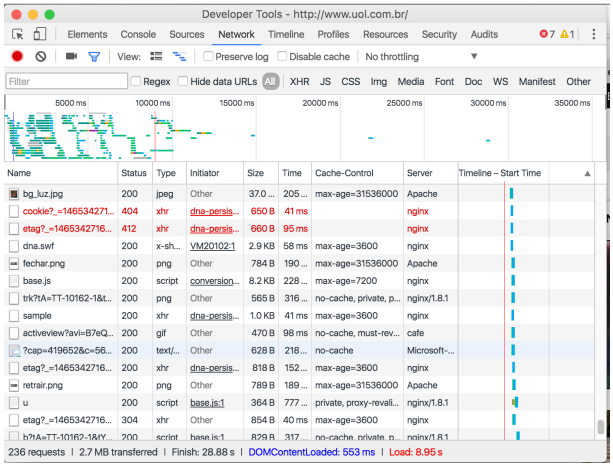

Figura 2.2: uol.com.br em seu primeiro acesso

Primeiro acesso após exibir toda a página

Como explicado acima, o uol.com.br se utiliza da estratégia de *lazy loading*, para estimular a requisção de todos os conteúdos, todo o site foi exibido utilizando a barra de rolagem lateral. Este cenário mostra o tamanho total da página : 4.4 MB. Além disso pode-se ver o elevado número de requisições: 388. Como pode ser visto na figura 2.3.

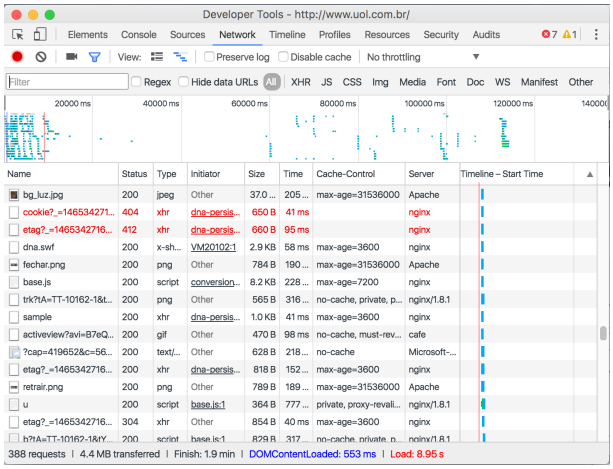

Figura 2.3: uol.com.br em seu primeiro acesso com estímulo total

Uma página contém outros conteúdos como arquivos de scripts folhas de estilos e arquivos multimidia como vídeos e animações, na figura 2.4 exibe o tamanho total das imagens no site.

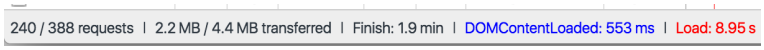

Figura 2.4: uol.com.br valor total das imagens requisitadas

Segundo acesso após exibir toda a página

Nesse momento todas as imagens e outros recursos marcados com os cabeçalhos de cache como por exemplo : *max-age=86400*, já se encontram na máquina, e portanto não necessitam ser baixados do servidor. Como pode ser visto na 2.5, o total de dados trafegados caiu de 4.4 megas para 958 Kbs. No caso das imagens, a queda de 2.2MB para 376 Kbps.

Figura 2.5: uol.com.br as imagens na segunda exibição

Como pode ser observado nos números a quantidade de dados não transmitidos no segundo acesso é significativamente menor. Esta economia traz uma impressão de velocidade ao usuário, ao mesmo tempo que evita sobrecarga nos servidores. Outro grande benefício é a diminuição dos dados trafegados no *uplink* do provedor de internet, com isso, o acesso a este site literalmente se torna mais barato, nos acessos posteriores.

2.7 Sumário

Neste capítulo foram introduzidos conceitos sobre *cache*, desde conceitos básicos a definições sobre políticas de manipulação de dados em *cache*, bem como foram definidos terminologias comuns a utilização de mecanismos de *cache*. Foram também abordados os cabeçalhos de meta dados definidos no protocolo HTTP, que servem de base para definir o comportamento dos navegadores com relação a *cache*. O próximo capitulo levanta características importantes da arquitetura dos provedores de internet e como os dados trafegam entre os mesmos.

Capítulo 3

Arquitetura dos provedores de internet

Neste capítulo serão abordados os temas relacionados a estrutura dos provedores de internet, como é o modelo de passagem de pacotes entre os pontos de troca na internet, e o seu impacto nos valores totais de tempo e de custos financeiros ao usuário.

3.1 A estrutura

Os provedores de internet realizam a conexão do usuário comum com a rede mundial de computadores. Existem vários modelos de conexão cliente-provedor : rádio, fibra ótica, 4G, DSL, entre outros. Entretanto todos eles criam uma *WAN* do inglês *wide area network*, uma rede de grande extensão que conecta os assinantes aos roteadores da operadora. Um pacote enviado por um usuário sai de sua máquina e chega ao roteador principal da operadora, deste ponto em diante, o pacote de dados pode: continuar nessa rede ou sair da mesma rumo a outros servidores fora da rede. Estes são os chamados de servidores de *upstream* ou *uplink*. Uma terceira opção é o pacote ir para uma rede entre provedores. Estas conexões entre redes de provedores é chamada de *IXP* do inglês *Internet eXchange Points*, pontos de extensão de internet. A tabela 3.1 tem uma listagem dos maiores IXP do mundo, vale a pena salientar a alta conexão entre países do primeiro mundo, o Brasil também está presente na tabela ocupando a sexta posição em total de dados trafegados com 1990 Gbits por segundo.

O propósito principal dessa rede é conectar diretamente redes de alto volume de tráfego, sem que haja o envolvimento de outras redes, com isso os custos com a latência

Tabela 3.1: Maiores IXP do mundo ordenados por média de tráfego

Nome	Países	Desde	Média de tráfego	Medido em
DE-CIX	Alemanha, EUA, Itália, França Turquia	1995	3159 Gbit/s	03/2016
AMS-IX	Holanda	1997	2481 Gbit/s	03/2016
LINX	EUA, Reino Unido	1994	1990 Gbit/s	04/2016
DATA-IX	Russia, Ucrania, Kazakstão, Alemanha	2009	1300 Gbit/s	03/2016
MSK-IX	Russia	1995	1234 Gbit/s	06/2016
IX.br	Brasil	1994	1990 Gbit/s	06/2016

e banda podem ser reduzidos. O tráfego que passa por esses pontos costuma não ser tarifado, já as transferências com o *upstream* server possuem taxas altas por envolverem conexões entre largas distâncias como países ou continentes. Assim ao aumentar a quantidade de conexões com outros provedores ou redes de conteúdo *CDNs* o custo médio por bit tende a diminuir caso o tráfego seja localizado nas redes destes provedores com quem o provedor faz parcerias. Este custo médio por bit em conjunto com outros fatores como custos de instalação, aluguel da rede entre outros, compõem o custo final que é repassado aos assinantes. Desta forma a medida que os dados sejam mais localizados dentro dos limites das redes dos provedores, o custo associado aos mesmos tende a cair. A imagem 3.1 exibe a estrutura de um provedor de internet do Reino Unido, o tráfego interno pode ser visto nos 4 primeiros pulos, apenas o tráfego que sobe para o *upstream server* é tarifado por bit, o resto da infraestrutura é diluído nos custos de manutenção/instalação, e não são tão variáveis em função do volume trafegado.

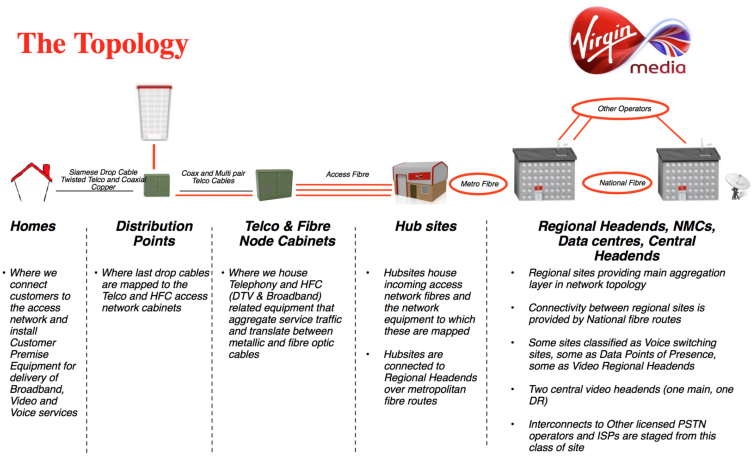

Figura 3.1: Imagem retirada do site do provedor de internet Virgin

Além da localização dos dados, outro aspecto que interfere bastante na rede dos provedores de internet é o tipo de tráfego, que sera abordados na seção a seguir.

3.2 Os dados

A internet é rica em conteúdos diversos, de textos, imagens a arquivos de multimídia como áudios e vídeos. Com o aumento da velocidade dos provedores, alguns serviços começaram a surgir, mudando a maneira de como os dados são consumidos.

Serviços como Napster, iTunes, permitiram que os usuários utilizassem suas conexões de internet para baixar legalmente músicas. Vários canais americanos também possibilitam que os espectadores baixem legalmente os arquivos de vídeos em alta definição de suas séries favoritas. Todos estes arquivos foram durante muito tempo baixados de forma assíncrona, onde o usuário , adquire o conteúdo caso seja pago, e o coloca em uma fila para ser posteriormente descarregado. Após algum período de tempo, o arquivo se torna disponível para execução no computador do usuário.

A medida que as velocidades melhoraram e os acordos com empresas de mídia evoluíram, o modelo de como os conteúdos são consumidos tem mudado. Ao invés da posse do item, como o modelo seguido pelo iTunes, o que está sendo praticado é a cobrança de assinatura, para ter acesso a serviços que propiciam que o usuário execute as mídias de áudio e vídeo a medida que as mesmas estão sendo baixadas, o chamado *stream*. Vários serviços se tornaram extremamente populares como o Netflix e o Spotify. Isso acarretou numa mudança significativa no volume de dados trafegados nos provedores. Textos e imagens tipicamente consomem menos banda do que áudio e vídeo. E caso não existam redes provedoras de conteúdo com as mídias, os custos impostos aos provedores aumentam a medida que o conteúdo se torna mais volumoso. Para exemplificar o cenário: no ano de 2015 nos Estados Unidos, em um estudo realizado pela empresa Sandvine [13] o volume aferido apenas para o tráfego do serviço Netflix foi de 24% do trafego total como pode ser visto na figura 3.2.

Table 1 - North America, Fixed Access, Peak Period, Top Applications by Bytes

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	52.01%	Netflix	29.70%	Netflix	24.71%
2	HTTP	8.31%	HTTP	18.36%	BitTorrent	17.23%
3	Skype	3.81%	YouTube	11.04%	HTTP	17.18%
4	Netflix	3.59%	BitTorrent	10.37%	YouTube	9.85%
5	PPStream	2.92%	Flash Video	4.88%	Flash Video	3.62%
6	MGCP	2.89%	iTunes	3.25%	iTunes	3.01%
7	RTP	2.85%	RTMP	2.92%	RTMP	2.46%
8	SSL	2.75%	Facebook	1.91%	Facebook	1.86%
9	Gnutella	2.12%	SSL	1.43%	SSL	1.68%
10	Facebook	2.00%	Hulu	1.09%	Skype	1.29%
	Top 10	83.25%	Top 10	84.95%	Top 10	82.89%

SOURCE: SANDVINE NETWORK DEMOGRAPHICS

sandvine

Figura 3.2: Estudo realizado para o tráfego dos EUA por tipo de serviço

Além disso é importante frisar o volume de dados do serviço Youtube de 9.85% . O tráfego HTTP se limita a 17.18%. Neste cenário de consumo de dados, uma estratégia de *cache* que possa se aproveitar mais da rede local dos *ISPs* e que possa ser utilizada tanto para textos, imagens como áudio e vídeo, pode ter impactos significativos tanto em velocidade como em redução de custos.

A seguir o cálculo dos custos por bit é explicado em maiores detalhes.

3.3 Os custos

A média do custo por bit do provedor segue uma fórmula relativamente simples. Segue abaixo um exemplo prático: Se o provedor tem uma linha de transmissão que roda a 40% de utilização, são 2 megabits por segundo (2mbps) de capacidade potêncial, mas 800bps de utilização real.

800kbps = 48m bits/minuto
= 2.88g bits/hora
= 69.12 g bits / dia
= 2.10816 t bits / mês

Se o custo para o provedor desta conexão é de US$ 12,000 por mês, uma interpretação superficial poderia dizer que o custo seria de US$ 6,000 megabits / segundo / mês. Entretanto, isto não leva a utilização do canal em conta. O custo por bit para 2,108.16 gigabits a US$ 12,0000 é:

12,000 / 2,108.16 = US$ 5.69 / 1

Ou **USD$ 5.69 por gigabit**.

Em um segundo cenário, o provedor pode ter um *link* de 10mbps com um *IXP*, se gastarem USD $ 3,000 em equipamentos, eles poderiam ser amortizados em dois anos, resultando em um custo de USD$ 125 / mês associado a conexão com *peers*. Se utilizarem 15% totaliza : 1.5 mbps.

1.5mps
= 90mbits / minuto
= 5.4gbits / hora
= 129.6gbits / dia
= 3.9528tbits / mês

O custo médio por bit para 3.9528 gigabits a US$125 :

$ 125 / 3,952.8 = $ 0.0316/1

Ou **USD 0.03 por gigabit**.

Em um terceiro cenário o provedor poderia ter os dois circuitos de entrega de pacotes, com uma capacidade total de 2.3 mbps e pagando USD 12,125 / mês:

$12,000 + $125 = 12,125

800 bps + 1.5 mbps = 2.3 mbps

2.3 mps * 2,635,200 segundos / mês = 6.06096 t bits / mês

$ 12,125 / 6060.96 = $2.00 / 1

O custo total para o provedor é $ 2.00/ gigabit. Existem dois fatores que contribuem com o custo: o trânsito a $5.69 e o custo de *peer* a $0.03 / gigabit.

O quarto cenário o tráfego é transferido do *link* mais caro para o mais barato. Neste cenário supondo uma transição de 8.7% de 200kbps, para o *link* mais barato, o *peer*. Isto aumentaria o *peer* de 1.5 mbps para 1.7 mbps. Diminuindo de 800 kbps para 600 kbps.

Isto reduziria a utilização do *link* de 2 mbps de 40% para 30%

Se isto reduzir de 2 mbps para 1 mbps, o que poderia reduzir de USD 12,000 para USD8,000 e com uma utilização de 60% isso resulta no seguinte cálculo:

600kbps * 2,635,200 segundos / mês = 1,581.12 g bits/ mês

$ 8,000 / 1,581.12 = $ 5.06 / 1

1.7 mbps * 2,635,200 segundos / mês = 4,479.84 g bits / mês

$125/ 4,479.84 = $ 0.0279 / 1

2.3 mbps * 2,635,200 segundos / mês = 6.06096 t bits / mês

$ 8,125 / 6060.96 = $1.35 / 1

Com uma mudança de **100 kbps** o provedor reduziu de **$ 2.00** para **$ 1.34**, reduzindo os custos de **$ 12,125 / mês** para **$ 8,125 / mês**.

Assim pode-se concluir que o tráfego local entre *peers* pode reduzir significativamente os custos dos provedores de internet.

3.4 Sumário

Neste capítulo foi abordado a estrutura nas quais os provedores de internet fornecem seus serviços. Esta composição é vital para exemplificar os custos envolvidos em tráfegos locais e externos à rede dos *ISPs*.

Capítulo 4

Redes P2P e redes colaborativas

Neste capítulo serão abordados os temas relacionados a redes P2P, como características importantes e sua adequação para ser utilizada como infraestrutura para uma estratégia de *cache*. Também será abordado o tema *cache* colaborativa.

4.1 Redes P2P

Redes P2P são uma arquitetura de aplicação distribuida que particiona tarefas ou cargas de trabalhos entre os nós. Nós são igualmente privilegiados e não há uma diferenciação como em arquiteturas tradicionais onde existe um nó central e nós secundários. Os nós se comunicam de maneira descentralizada e agem tanto como produtores bem como consumidores.

Do ponto de vista da arquitetura pode-se salientar as seguintes configurações:

- Estruturada : onde existe uma topologia básica entre os nós, definida por protocolo, assim pode-se realizar uma topologia desejada.
- Desestruturada: Como não existe nenhuma estrutura defininda os nós se conectam de maneira *ad-hoc*.

Uma rede P2P pode prover e utilizar recursos, um deles pode ser a disponibilidade de conteúdo. O compartilhamento de arquivos ficou amplamente conhecido com protocolos e aplicativos em rede como : Bittorrent, Gnutella, entre outros.

Sob o ponto de vista de criação de redes de distribuições globais, várias CDNs utilizam atualmente redes *peer to peer* para distribuir os conteúdos entre as redes de forma automatizada, realizando assim uma sincronia global mais simples do que se houvesse a necessidade de trocar todos os arquivos de maneira estruturada ou sincronizada em alguma data/horario especificado.

Outras características relevantes das redes P2P

Comunicação direta Os nós se comunicam de maneira direta, e não utilizando um servidor intermediário, garantindo assim uniformidade de acesso ao conteúdo ou serviços disponíveis.

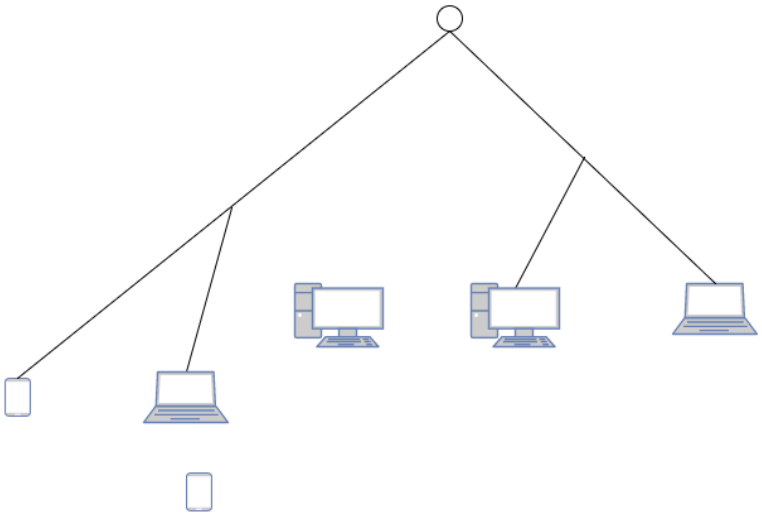

Figura 4.1: Redes P2P estruturadas, nós não se arranjam aleatóriamente, mas seguindo uma estrutura definida

Adaptabilidade Como o meio de acesso não é relevante para que a conexão seja mantida, uma vez estabelecido o acesso a rede, o mesmo se mantem ou podem adaptar para outro tipo de conexão, por exemplo um acesso a rede iniciado em uma conexão DSL, pode ser continuado por acesso 3G.

Escalabilidade Como relação de número de usuários não está associada a capacidade de um servidor intermediário, a quantidade de nós na rede pode ser literalmente sem limites. devido a essa característica se torna fácil evitar congestionamentos, basta buscar o conteúdo ou serviço de outro provedor.

Mobilidade O usuário pode se desconctar da rede e reconectar novamente sem que isso cause algum prejuízo a rede como um todo ou aos acessos a serviços ou conteúdos que o usuário estivesse acessando no momento.

Auto organização Como não há autoridade central, a rede pode se organizar da maneira que for mais interessante para o funcionamento da mesma. Se um grupo de nós se beneficia mais por estarem próximos geograficamente ou topologicamente , a rede pode por algoritmo priorizar os acessos a conteúdos e serviços entre esses nós caso os mesmos estejam presentes, caso contrário a rede opta pela opção mais distante.

Elevada disponibilidade Como não há a dependência de um único nó a rede pode prover seus serviços e conteúdos mesmo que vários dos nós que a compoem não estejam conectados.

Colaboração na partilha de recursos Tanto no caso de armazenamento como processamento, os nós da rede se auxiliam para que estes recursos possam ser compartilhados,

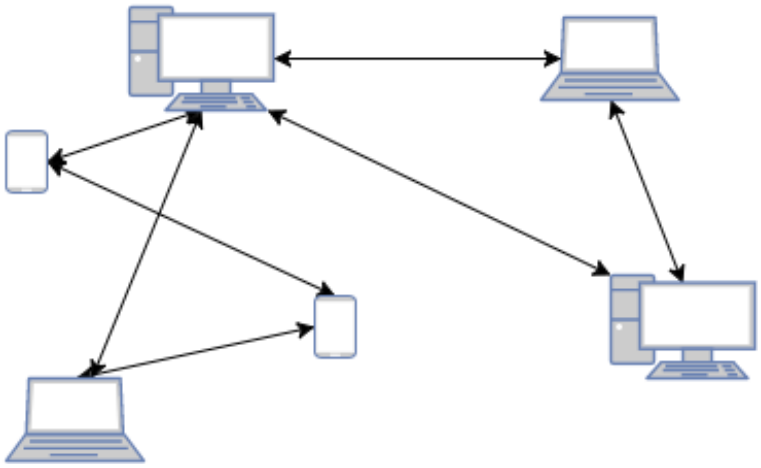

Figura 4.2: Redes P2P não estruturadas, o arranjo dos nós é aleatório

de forma colaborativa, arquivos podem ser transferidos de várias fontes para um destino só, assim como o poder computacional de vários podem auxiliar a computação de um nó.

Menor vulnerabilidade a ataques DDOS Como não há um servidor único, ataques de negação de serviço são infrutíferos, porém pode-se realizar um ataque à comunicação da rede, realizando a prática de envio excessivo de mensagens *flood* .

4.2 Redes de cache colaborativas

Neste capítulo serão abordados os temas relacionados a redes de *cache* colaborativas, como características importantes e o seu atual estado da arte.

Redes colaborativas são redes de computadores que através de suas interconexões buscam colaborar para um fim em comum. No caso das redes de *cache* esta colaboração busca realizar a *cache* de maneira altamente distribuida, onde cada nó busca prover uma cache confiável e escalável.

Para enteder a diferença entre uma *cache* simples e uma *cache* distribuída, é necessário entender suas características:

- **Cache de conteúdo** é diferente de **Endereçamento de conteúdo** No caso de um cache simples o conteúdo já está diretamente associado ao local, enquanto na rede distribuída o conteúdo ainda tem que ser "encontrado"dentre os nós que compõem rede.
- **Capacidade efetiva** é diferente de **Tamanho agregado total da cache** A capacidade efetiva da rede em um sistema simples é diretamente associado a quantidade de dados disponível e capacidade do servidor. No caso da distribuida o tamanho da rede não necessariamente é associado a sua capacidade pois a complexidade de procura pode dificultar o acesso aos dados.

- **Ótimo local** é diferente de **Ótimo global** Estratégias locais de *cache* podem ser mais complexas em redes distribuídas, assim o melhor modelo pode não ser o mesmo que é utlizado na estratégia local.

Como pode ser visto uma das grandes dificuldades em redes distribuídas é o mapeamento das entradas nos nós que possuem os dados. Muitas vezes não é viável ter uma cópia completa do conteúdo em todos os nós, e mesmo se fosse, devem existir algoritmos para manter a sincronía do conteúdo entre os nós e nesse processo os dados podem ser invalidados.

Outra complexidade se dá no roteamento dos dados entre os nós, ao aumentear cada servidor extra na rede suas rotas podem se tornar mais complexas do que as rotas em um sistema simples, assim o tempo total pode aumentar e fazer a rede mais lenta que o acesso direto prejudicando o propósito original da rede.

Existem alguns desafios que são estudados na literatura.

4.3 Desafios

- Como medir a eficiência das redes distribuidas.
- Políticas de distribuição dos dados entre os nós da rede
- Política de pesquisa de dados
- Número máximo/mínimo ideal de réplicas
- Políticas de colaboração incentivadoras
- Políticas de agrupamento de nós

4.4 A configuração ótima

A configuração ótima entre número de nós e suas contribuições para a rede pode ser definida por um coeficiente, o da eficiência de Pareto. Melhorias podem ser realizadas sem que os participantes tenham perdas no processo. Abaixo segue um exemplo de redes colaborativas atingindo a fronteira de eficiência de Pareto. Os pontos B D e C na figura 4.3 exibem as fronteiras de eficiência. Com as variações das políticas de cooperação Tipo II, Tipo III, pode-se chegar na eficiência ótima D Tipo IV.

4.5 Estado da arte

Dentro da bibliografia consultada o trabalho [6] descreve uma abordagem de cache colaborativa baseada na popularidade do conteúdo. O trabalho descreve também uma maneira de concentrar o tráfego e seu conteúdo dentro das redes dos *ISPs*. Conseguindo atingir uma diminuição significativa de *hops* entre os nós.

Outro trabalho que toca o tema de caches colaborativas e redes P2P [19] . Neste trabalho uma rede P2P é montada para realizar a cache, os dados são distribuidos de maneira semelhante a este trabalho. Porém para realizar a implementação os autores

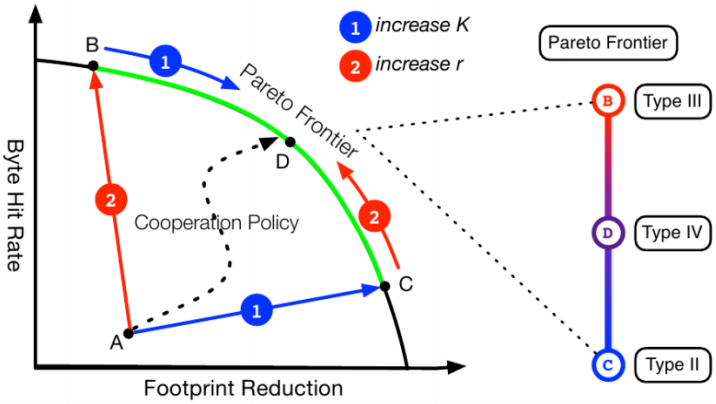

Figura 4.3: Limiares da configuração ótima

criaram um *add-on* de navegador para realizar as transferências e a interceptação dos dados. Um grande entrave na adoção da solução em larga escala.

O trabalho [24] descreve uma estratégia de localização de dados para redes P2P essa abordagem é interessante pois como foi descrito na introdução o percentual do trafego de redes P2P aumenta a cada ano e *cache* para redes P2P acarretaria numa melhoria do tráfego como um todo pois o volume é significante perante a outros conteúdos como o tráfego HTTP. Outro aspecto interessante é que a solução proposta poderia também se beneficiar de mecanismos de *cache* para redes P2P.

O trabalho [25] define uma alteração no protocolo HTTP para que o mesmo possa ser servido através de servidores em uma rede P2P. Interessante ver o mecanismo para tal funcionalidade é basicamente o que será implementado na proposta, bastando estender a proposta para incorporar conteúdos de texto de marcações HTML, assim seria teóricamente possivel executar o conceito da solução proposta pelo trabalho acima sobre a tecnologia desenvolvida nesta proposta.

Como ja foi mencionado na introdução o trabalho [3] mostra que na maneira atual que o conteúdo é produzido e consumido novas tecnologias de *cache* devem ser implementadas pois a descentralização diminui a eficiência das estruturas atuais.O trabalho foca mais nos itens passíveis de *cache*, mas a pesquisa realizada por ele corrobora a necessidade de soluções como a proposta neste trabalho.

O trabalho [11] define uma estratégia para servidores de cache utilizarem redes P2P para fazer a distribuição de conteúdo. O foco do trabalho é a sensibilidade à localidade e interesse, baseado nas métricas dos servidores.

O trabalho [8] define uma estratégia de agrupamento dos nós baseada em tempo de conexão a um servidor intermediário. Com esses grupos que funcionam como pétalas o número de nós próximos ao novo nó provêem o conteúdo teoricamente mais topologicamente próximo, melhorando a velocidade total da *cache*.

4.6 Sumário

Neste capítulo foram apresentadas as características das redes de *cache* colaborativas e seu estado da arte. Também foi apresentado o conceito de *cache* colaborativas nesse contexto de redes P2P, também com o estado da arte da área de redes de *cache* colaborativas.

Capítulo 5

HTML5 e WebRTC

Neste capítulo será apresentado o HTML5, as adições de novas interfaces de programação e também será detalhado o WebRTC.

5.1 HTML5

Desde de que a versão inicial do HTML se tornou pública , em outubro de 1991, sua adoção não parou de crescer. Em 1992 o primeiro rascunho foi elaborado e após algumas revisões foi publicado em 1993 a primeira proposta de padrão, que culminou na especificação do HTML 2.0 na RFC 1866[22].

Desde esta época até 2008, algumas propostas foram feitas mas nenhuma ganhou muita notoriedade, até mesmo a proposta do HTML 3.0 acabou não sendo muito popular. O grande problema se dá no tamanho que a internet acabou ganhando no mercado mundial. A padronização entre os navegadores disponíveis acabava dificultando algumas empresas de se destacarem de seus concorrentes. Esta falta de padronização tornou popular linguagens de scripts e plugins que possibilitavam maiores funcionalidades além da especificação do HTML.

O Adobe Flash foi uma tecnologia que surgiu justamente para tentar suprir uma lacuna no desenvolvimento para a internet, por ser proprietário surgiu o problema de sua implementação ser fechada e paga. Com o dinamismo que seu ecossistema de ferramentas proporcionava, o Flash se tornou padrão de mercado e a maioria dos sites empregava uso de painéis: publicitários ou outros painéis de mídia mais avançados como a reprodução de arquivos de áudio e vídeo. Houve uma proliferação de sites que utilizavam esses recursos e o Flash se tornou o padrão para a exibição de vídeos e manipulação de recursos de mídia do computador como, câmera e microfone. A própria versão original do Youtube utilizava a tecnologia Flash.

Para tentar definir interfaces padrões abertas para todos os navegadores, em 2008 saiu a primeira proposta de padronização , que buscava incorporar recursos avançados de manipulação de mídia e recursos do computador dentro da especificação, para que extensões e *plugins* de navegadores não fossem mais necessários para se ter uma experiência rica.

A utilização dos recursos de mídia foi se tornando possível pois com o aumento da capacidade de recursos dos computadores, bem como o aumento da velocidade média de conexão , as páginas se tornaram mais dinâmicas e com maior quantidade de conteúdos interativos. Esta tendência criou a demanda pela adoção do padrão no mercado de nave-

gadores, e por coincidência , o mercado já estava bem diferente dos anos anteriores. As outras tentativas de padronização de navegadores não foram muito bem sucedidas parcialmente devido a grande fatia do mercado ser dominada pela Microsoft com o seu navegador Internet Explorer. Após a venda do navegador Netscape, uma fundação e iniciativa open source nasceu e acabou produzindo o navegador Mozilla. Após uma diferença de opiniões crescente entre a Mozilla e um de seus maiores apoiadores, a Google, alguns funcionários deixaram a fundação e foram trabalhar em um novo produto da Google, o navegador Chrome. Com a adoção em massa de navegadores alternativos ao Internet Explorer, a adoção dos padrões se tornou mais frequente, e também trouxe novas oportunidades para padronizar os recursos mais utilizados no novo cenário da web, os conteúdos de mídias digitais.

Assim em 2011 o *W3C* iniciou o processo de escrita final do padrão HTML5, culminando em sua publicação oficial em 2012. Desde então já houveram novas propostas e recomendações, no momento da escrita desta dissertação a última recomendação havia sido publicada em 28 de outubro de 2014.

5.1.1 As adições a especificação

O HTML5 introduziu novos elementos e atributos que possiblitam uma divisão do conteúdo de maneira mais semântica, dando uma conotação a funcionalidade do conteúdo, facilitando assim a compreensão do site por robôs e padronizando as subdivisões internas nos sites. Foram adicionadas várias tags, abaixo seguem algumas *tags* ou marcações para conteúdo:

- <article>- Define um conteúdo auto contido, como uma entrada em um *blog* ou uma notícia.
- <aside>- Define conteúdos que devem ser colocados como barras laterais.
- <details>- Define um conteúdo que pode ser exibido ou escondido
- <figcaption>- Define a descrição de uma imagem ao ser utilizado em conjunto com o <figure>
- <figure>- Define uma maneira de colocar descrições em imagens
- <footer>- Define rodapés para cada artigo, seção ou documento
- <header>- Define cabeçalhos para cada artigo, seção ou documento.
- <main>- Define o conteúdo principal de um documento
- <mark>- Define um texto em destaque
- <nav>- Define âncoras de navegação
- <section>- Define um grupo de conteúdo, semelhante a seção de um livro ou artigo.
- <summary>- Define um cabeçalho visível para a tag de detalhes
- <time>- define a data/hora

Estas tags demonstram uma tendência de tornar o código do HTML mais explicativo sobre a estrutura do seu conteúdo, e faz com que a aparência seja colocada em folhas de estilos externos, realizando a separação do conteúdo das informações de apresentação. A vantagem desta abordagem é que ela possibilita a obtensão do conteúdo de forma assíncrona e segmentada pelas suas subdivisões, algo que pode se beneficiar fortemente de mecanismos avançados de *cache*.

Além do conteúdo, outras *tags* foram adicionadas, como:

- <canvas>- para manipulação direta pixel a pixel em alto desempenho.
- <audio>- para executar arquivos de áudio.
- <video>- para executar arquivos de vídeo.

Existiram várias adições de *apis*, segue abaixo uma imagem que resume todo o ecossistema de mudanças associado ao HTML5.[20]:

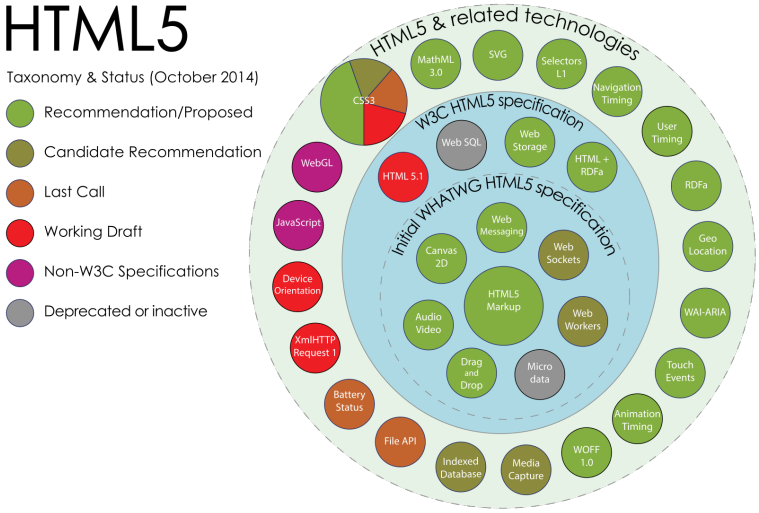

Figura 5.1: APIs relacionadas ao HTML5

Duas novas apis definidas no HTML5 são importantes para o tema de cache e para a sugestão proposta neste trabalho:

5.1.2 Mudanças na cache

A nova especificação define uma maneira para o desenvolvedor do site marcar através de um arquivo de manifesto, todos os recursos , que devem ser gravados localmente. Assim possibilitando a utilização da aplicação sem estar conectado com a rede. A utilização é bem simples, o desenvolvedor deve apenas colocar um atributo na definição da *tag* de abertura do html:

```
<!DOCTYPE HTML>
<html manifest=cache.appcache>
  <body>
  </body>
</html>
```

Neste caso a definição aponta o arquivo "cache.appcache", como sendo o arquivo que contém as definições de cache. O arquivo é simples e tem apenas três marcações que definem características de cache distintas para os arquivos listados abaixo da definição.

- NETWORK: - define uma seção que deve utilizar a rede, ou seja funciona como uma lista do que não deve ser gravado na *cache*
- CACHE: - define os itens que devem ser gravados, utilizando seu caminho relativo a raiz do site.
- FALLBACK: - define subistitutos para recursos que não estão salvos na *cache*, e não pode ser acessados no momento.

Para ilustrar segue abaixo um arquivo de manifesto para a *cache* de um site:

```
CACHE MANIFEST
NETWORK:
/checking.cgi
CACHE:
/test.css
/test.js
/test.png
FALLBACK:
/ /offline.html
```

Além do arquivo de manifesto e da tag a nova especificação define uma API de *Javascript* para acesso a eventos relacionados a *ApplicationCache*:

- Checking - Evento lançado quando o navegador lê a o atributo associado ao cache de aplicação na tag <html>.
- Downloading - Evento lançado se o navegador não tiver baixado os arquivos do manifesto anteriormente, o navegador inicia o processo de baixar os arquivos listados no manifesto.
- Progress - Evento mostrando o andamento do processo de baixar os arquivos .
- Cached - Evento que ocorre ao fim do processo de baixar o arquivo em *cache*.
- Noupdate - É lançado caso o arquivo de manifesto já tenha sido processado anteriormente e não houve nenhuma mudança no arquivo de manifesto, em um novo acesso.
- Updateready - É lançado após a nova versão do manifesto ter sido totalmente processada e baixada em disco.

Apesar desta *API* ter sido definida para facilitar a utilização de aplicativos de maneira desconectada da rede, ela serve como uma boa indicação de quais recursos podem ser passíveis de *cache* em uma página. A manipulação de objetos via a API javascript também sinaliza uma possível expansão para que itens sejam salvos via script e futuramente poder ser adicionados a *cache* conhecida do site. Porém sua adoção ainda é bem pequena e limitada a aplicativos que possam ser totalmente utilizados sem internet.

5.1.3 Acesso a persistência local

Uma das maiores necessidades dos desenvolvedores a medida que os aplicativos começam a se tornar maiores e mais complexos , é a necessidade de ter uma persistência para salvar dados. Na internet esta persistência ainda que de maneira rudimentar foi por anos implementada nos *cookies*. Arquivos locais, residentes na máquina do usuário, que possibilitavam a gravação de dados localmente, porém limitados em tamanho e tendo o seu conteúdo gravado em *string*. Outro fator limitante é que os *cookies* são enviados a cada requisição, aumentando drasticamente a quantidade de dados trafegados. Com a definição da persistência local, o HTML5 possibilita ao desenvolvedor salvar dados que nunca são tranferidos ao servidor.

A API define duas maneiras de se gravar dados no usuário:

- *localStorage* - para gravar dados que não expiram, e podem ser acessados de qualquer aba do navegador.
- *sessionStorage* - para gravar dados que expiram após o fim da sessão do usuário, com o escopo definido por abas do navegador.

Ambos objetos tem a mesma definição de API para acesso aos dados.

- setItem - para gravar o dado, passando um identificador em string e o dado a ser gravado.
- removeItem - passando o identificador do dado a ser removido
- identificador - para acessar o valor gravado na persistência

Além do acesso definido acima para a gravação de dados no formato identificador, valor, o html5 definiu uma API para acesso a arquivos. Infelizmente ela foi descontinuada como padrão pois os navegadores não se interessaram em disponibilizar uma interface para a manipulação de arquivos em disco.

5.2 WebRTC

As aplicações na internet vem se tornando cada vez mais complexas e interativas, mas um grande gargalo na adoção de novos recursos era a forma limitada de como o navegador podia iteragir com recursos do computador do usuário. Acesso a câmeras, microfones e até mesmo transmissão de dados só podiam ser feitos utilizando-se de tecnologias proprietárias na forma de extensões ou *plug-ins* de navegadores. Por esse motivo surgiu a tecnologia WebRTC, para conseguir expor via APIs em *javascript* o acesso a esses recursos antes escondidos ao desenvolvedor.

As três responsabilidades do WebRTC são:

- Adquirir áudio e vídeo - Solicitar o acesso ao *stream* de áudio e vídeo.
- Comunicar áudio e vídeo - Transmitir e receber os dados de mídia
- Comunicar dados - Transmitir e receber quaisquer dados

Estas três responsabilidades são mapeadas em três APIs

- MediaStream
- RTCPeerConnection
- RTCDataChannel

5.2.1 MediaStream

Representa uma única fonte de dados sincronizados de áudio/vídeo ou ambos. Cada *stream* de mída pode também conter faixas como pode ser observado na figura 5.2. Como exemplo em um laptop, a câmera e o microfone provêem *streams* e eles podem ser separados ou sincronizados. Para ter acesso a estes recursos o desenvolvedor deve chamar o método: *navigator.getUserMedia()*. Assim que o usuário aceita disponibilizar o recurso o *stream* fica disponível ao desenvolvedor.

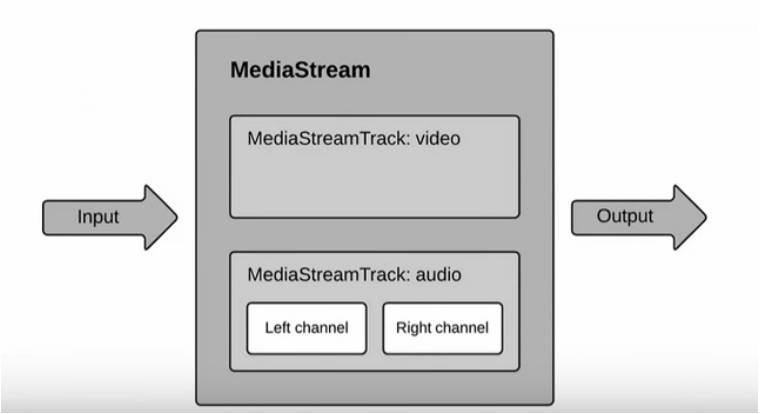

Figura 5.2: Diagrama da classe MediaStream

De posse do *stream* o desenvolvedor pode manipular e usar bibliotecas em *Javascript* para alterar ou mesmo exibir, gravar e reproduzir localmente os dados, e pode também utilizar o *stream* para enviá-lo a outros computadores pela rede.

5.2.2 RTCPeerConnection

A função do *RTCPeerConnection* é realizar a conexão entre o computador original com outro nó (*peer*). É responsabilidade dele abstrair do desenvolvedor as seguintes ações:

- Processamento de sinais
- Processamento de *Codec*
- Comunicação *Peer to peer*
- Segurança
- Administração de banda

- entre outros.

As responsabilidades do *RTCPeerConnection* são muitas, tornando mais simples para desenvolvedor realizar as conexões com outros nós, lidando com todas essas implementações relacionadas ao processo de conexão, que podem ser vistas na figura 5.3

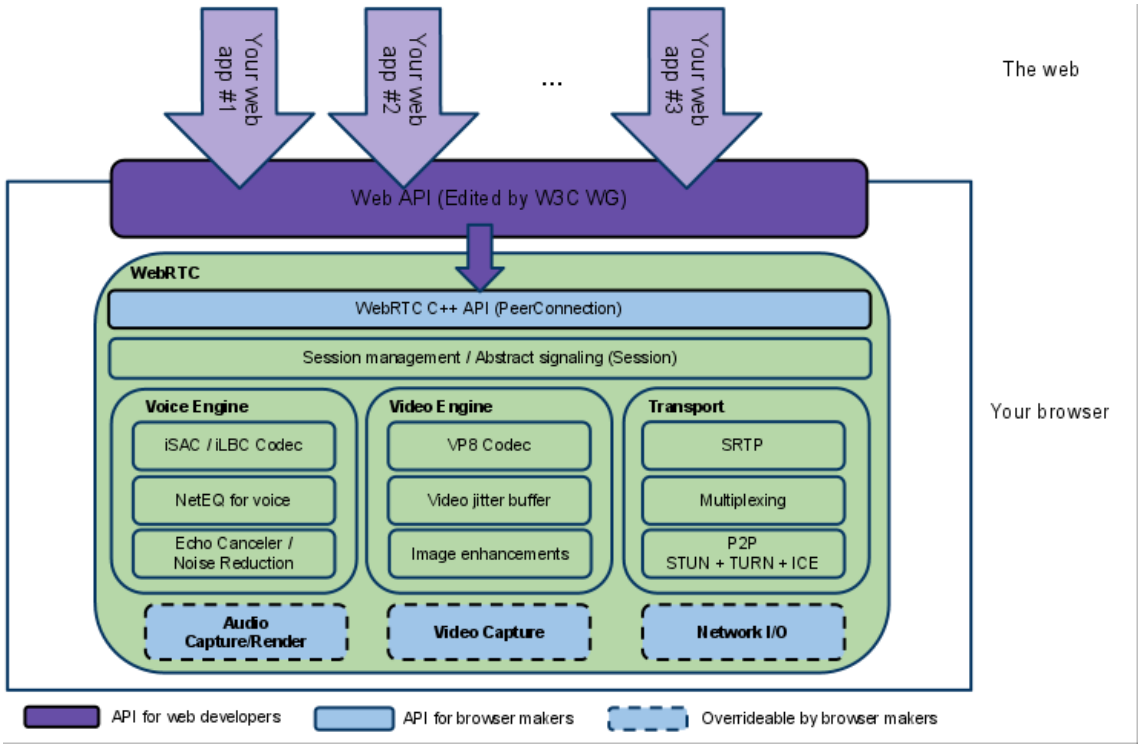

Figura 5.3: Arquitetura do WebRTC

5.2.3 RTCDataChannel

De posse da conexão com outros nós, basta agora uma maneira de enviar os dados. E é para isso que o *RTCDataChannel* serve. Algumas características :

- Mesma API que os *websockets*
- Latência baixa
- Modos de comunicação confiável/não confiável
- Segurança com encriptação DTLS

5.3 Sumário

Neste capitulo foram apresentadas as novas funcionalidades do HTML5 e do WebRTC e suas características importantes para este trabalho.

Capítulo 6

Proposta

Com a evolução da internet e a criação de diversas redes sociais onde o conteúdo é altamente distribuido e gerado pelos próprios usuários, os mecanismos tradicionais de *caching* estão se tornando cada vez menos eficientes [10]. Uma estratégia que consiga utilizar o poder computacional e os recursos dos usuários de um site , pode proporcionar uma cache adaptada a geração de conteúdo, de maneira distribuida e provida pelos próprios usuários.

A proposta é desenvolver uma estratégia de *web caching* utilizando uma rede *P2P* implementada sobre *WebRTC*. Com isso todo e qualquer cliente que esteja acessando um site, pode se tornar um distribuidor de seu conteúdo, provendo uma cópia local de arquivos de forma colaborativa, que podem compor uma *cache* distribuída cooperativa.

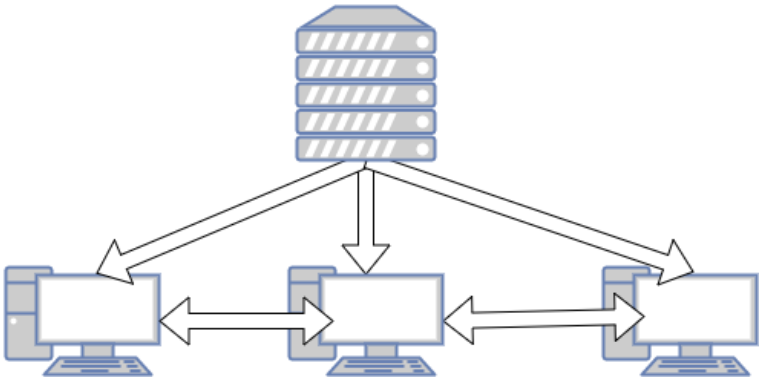

Figura 6.1: Arquitetura da solução proposta

6.1 Porque utilizar redes P2P

Como o conteúdo se encontra distribuído entre vários nós e a quantidade de nós tende a ser altamente variável, uma solução distribuida pode suportar tanto a quantidade de requisições quanto a alta rotatividade de nós. A estrutura proposta utiliza a implementação do protocolo bittorrent. Este estilo de arquitetura de *web proxy* já foi proposta[21] e é utilizada por servidores intermediários porém em nenhum dos casos é utilizada nas máquinas dos clientes. Nesta proposta a ideia é realizar a transformação de cada nó cliente que se encontra em um cenário cliente/servidor , em um nó também servidor, alocando parte de

seus recursos de disco , processamento e rede para agir como um servidor de arquivos, exatamente o que acontece quando o mesmo se encontra conectado em uma rede P2P. Outra vantagem do P2P é a utilização de mecanismos que tentam evitar que nós maliciosos ou que não contribuem com a rede, a utilizem da mesma forma que nós com bom comportamento, desmerecendo-os em seus algoritmos de preferência de nós. Além disso, como esses protocolos são voltados para a distribuição de arquivos, eles já possuem dentro de sua definição mecanismos que fazem a checagem do conteúdo baixado, e também a correta montagem de arquivos muito grandes que foram divididos em pedaços antes de serem transmitidos.

Como toda a troca de dados é feita entre os nós a descentralização comum em relações cliente/servidor também se mostra benéfica pois desta maneira o tráfego fica focado em regiões locais e não em pulos até o servidor de origem, com isso se o *tracker* for implementado utilizando a topologia dos acessos em conta, o tráfego pode pontencialmente permanecer restrito dentro das redes de cada provedor de acesso, tendo custos muito menores dos que os comparados com pulos entre *backbones*.

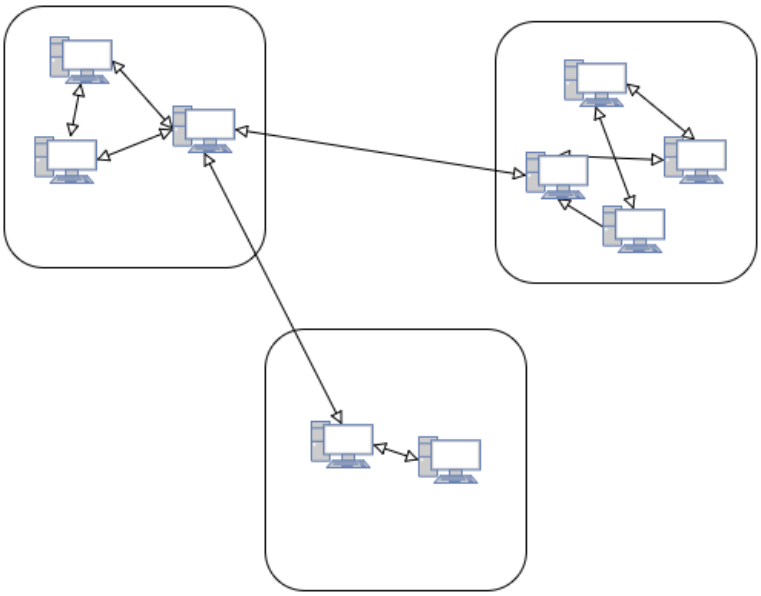

Figura 6.2: Cenário de agrupamento de nós por provedor de internet, privilegiando a conexão entre esses nós mas mantendo a possibilidade de conetividade entre redes, para evitar starvation

6.2 Arquitetura da solução

A solução utiliza a arquitetura proposta no protocolo bittorrent, porém existem papéis que seriam performados por programas distintos, que na arquitetura proposta serão realizados pelo mesmo software.

Segue a arquitetura proposta pelo protocolo bittorrent[1]

- Servidor web ordinário

- Arquivo de meta informações *MetaInfo* ou sua forma comprimida em url, o *magnet*.
- Bittorrent *tracker*
- Navegador do usuário final
- *Downloader/Uploader* do usuário final
- *Downloader/Uploader* original

Como o objetivo da proposta é definir uma estratégia de *web cache* , alguns papéis nessa arquitetura se sobrepõem.

- **Servidor web ordinário**: este é o servidor original do conteúdo
- **Arquivo de meta informações** *MetaInfo*: Será utilizada a forma comprimida na forma de uma url *magnet*.
- **Bittorrent *tracker*** - O tracker é um software que ficará hospedado em um servidor e fará o trabalho de distribuir as listas de nós que tem o arquivo desejado, e com isso realizar o trabalho de distribuir justamente a carga da *cache*.
- **Navegador do usuário fina**l : será também o cliente navegando no site junto com o *Downloader/Uploader*
- ***Downloader/Uploader* original** : será o próprio servidor original, com um cliente diferente , para evitar o travamento quando não houver nenhum nó de *cache* disponível.

Assim os componentes do sistema podem ser definidos da seguinte maneira:

Tabela 6.1: Componentes da solução proposta

Servidor original de conteúdo	Servidor com as alterações necessárias, que serão abordadas abaixo
Tracker bittorrent	Servidor simples que foi hospedado no mesmo servidor original do conteúdo
Imagens originais e cliente seeder	Também hospedado no mesmo servidor original de conteúdo
O navegador do cliente	Navegador que consiga executar javascript e tenha o WebRTC habilitado. Obs. Por padrão todos os navegadores no momento da escrita deste trabalho, vinham com suas configurações de WebRTC e Javascript habilitadas por padrão.
O servidor do script da solução	O código será provido por um servidor de aplicação feito em nodejs, e o mesmo estará rodando no mesmo servidor do tracker das imagens e do conteúdo original.

Alugns dos termos do protocolo bittorrent serão melhor explicados a seguir:

- **Seeder** é um nó que contribui com o compartilhamento, realizando *upload* do conteúdo que provê. Idealmente na solução todos os nós farão espontaneamente isso. O caso onde os usuários possam não realizar isto se dará quando o acesso for em celulares em conexão móvel, em um cenário real, isto não será abordado na solução.
- **Leecher** é um nó que não realiza upload, consome somente os dados de download. Idealmente na solução proposta isso não ocorrerá, mas em um cenário real é provavel que nós em acesso mobile se utilizem dessa estratégia para poupar dados do cliente.

O processo de adoção da plataforma é bem simples e se dará da maneira a seguir:

- O responsável pelo site incluirá o script da solução WebRTCProxyCacheJS em seu site.
- A partir deste momento o site poderá prover seu conteúdo através da rede P2P estabelecida por seus clientes.
- A implementação sugerida será realizada apenas para imagens (por restrição de tempo), assim o desenvolvedor deverá alterar a marcação das imagens no código html , para que a imagem não seja automaticamente lida pelo navegador. Similar ao que já ocorre em sites que utlizam a estratégia de *lazy loading* de imagens. Este passo é simples porém mandatório para que o navegador não inicie o *download* da imagem por conta própria.
- Após realizar as alterações nas imagens o responsável deve prover um servidor que terá a cópia de todas as imagens referenciadas pelo site e nele instalará o cliente (implementado especificamente para nós sem navegador), que agirá como *seeder* inicial de todo o conteúdo.

O processo de execução da solução em um caso de uso normal de acesso de um usuário será o seguinte:

- O usuário acessa o site utilizando-se do serviço de DNS como faz normalmente.
- O site provê todo o conteúdo com as observações citadas na listagem acima.
- O cliente executa o código da implementação que foi adicionado ao conteúdo original, e faz a listagem de conteúdos passíveis de *cache* que devem ser exibidos (imagens abaixo da linha visivel não serão baixadas inicialmente para evitar conexões desnecessárias).
- Para cada item da lista o programa pergunta ao tracker a lista de peers.
- O tracker responde a lista de peers ideal para o usuário em questão. Junto com o hash do conteúdo e outras informações de meta dados.
- O programa abre as conexões com os peers que tem o conteúdo e iniciam o *donwload*.
- Após o download ter finalizado o nó checa a integridade do conteúdo através do hash passado pelo tracker , e avisa o mesmo de que finalizou o download, exibindo a imagem no local apropriado.
- A partir deste momento o cliente pode se tornar servidor para o conteúdo já baixado.

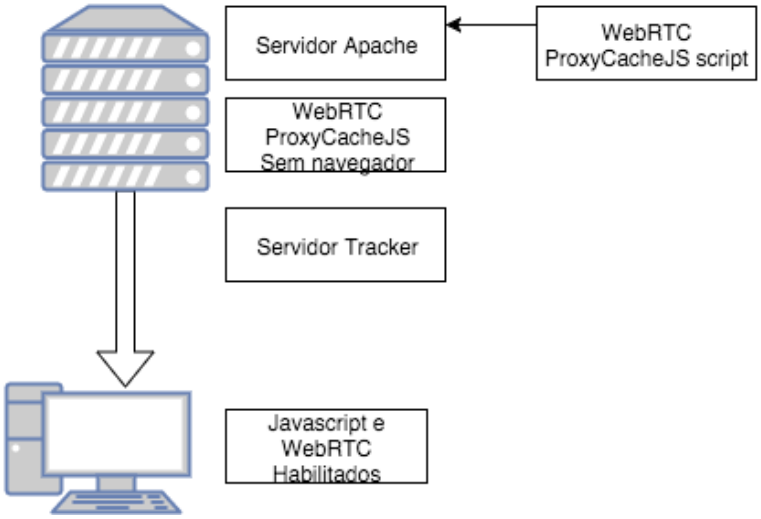

Figura 6.3: Descriçao dos componentes da solução. No servidor ficará o servidor de aplicação com a adição do script, o tracker e o aplicativo do cliente seeder de linha de comando. No cliente com javascript e WebRTC habilitados, o código da página será baixado como ilustra a seta grande

6.3 Solução WebRTCProxyCacheJS

A implementação de conexão direta entre nós dentro do navegador era impossível de ser realizada sem a utilização de mecanismos de *plugins* e *addons* em navegadores. Após a definição do nova especificação do HTML5 [12] e do WebRTC [2] esta implementação pode ser realizada em uma linguagem de script (Javascript) que é executada no cliente. O WebRTC define interfaces padrão, que podem ser executadas na maioria dos navegadores disponíveis, que expõe a camada de rede, no script sendo executado no navegador, com isso sockets, podem ser abertos e fechados, dados podem ser transmitidos de forma direta entre navegadores de usuários distintos.

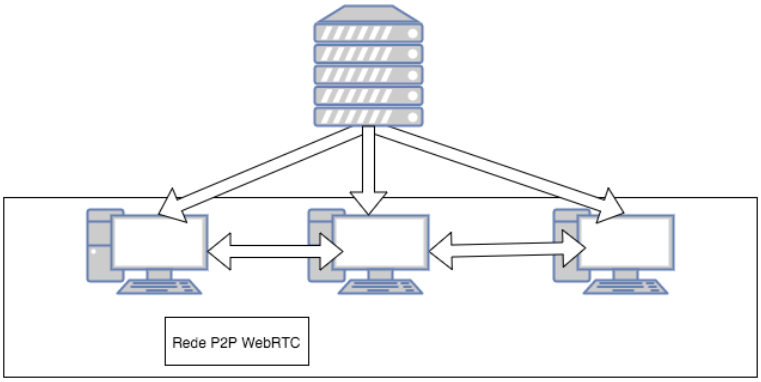

Figura 6.4: A rede P2P evidenciada na arquitetura, e implementada sobre WebRTC

Assim como foi detalhado anteriormente o processo de execução se dá de forma a inserir o nó que acaba de acessar a página na rede P2P. Com isso o nó abrirá canais de

RTCDataChannel com o peers que tiverem o conteúdo das *caches*. E ao mesmo tempo se registrará com o tracker, toda vez que o nó tiver o conteúdo completo, para que o mesmo se torne um possível *uploader*. O funcionamento detalhado de como o protocolo bittorrent foi utilizado na solu ção WebRTCProxyCacheJS será descrito na próxima seção

6.4 WebRTCProxyCacheJS

A solução proposta foi implementada sobre WebRTC, pois essa seria a única maneira de ter uma rede P2P rodando nos clientes sem a necessidade de instalação de extensões e plug-ins de navegadores. Além da notória performance e escalabilidade, o protocolo torrent foi uma das poucas opções de bibliotecas que se encontravam disponíveis para o desenvolvimento de aplicações sobre WebRTC, assim o mesmo foi escolhido, neste capítulo será detalhado o funcionamento do torrent rodando sobre o webrtc.

6.5 A visão geral

O protocolo torrent visa o compartilhamento de arquivos, desta maneira toda a comunicação é otimizada para conseguir realizar o processo de baixar o conteúdo utilizando poucas mensagens e de maneira altamente distribuida. O protocolo será detalhado na ótica de como foi utilizado na solução proposta:

6.5.1 O arquivo e seus meta dados

O arquivo a ser compartilhado deve ter um .torrent associado, este é o arquivo de meta dados que contém o *hash* do conteúdo do arquivo utilizando o algoritmo *SHA1*, e também a lista dos pedaços do arquivo. Segue abaixo os detalhes de um arquivo .torrent:

```
{
    name: 'Arquivo de imagem 1',
    pieces: ['1hj1hyiuhkh8u89j91ij', 'i89iko372uoijew7uw8yi3ho'],
    announce: 'http://tracker.carlosbpf.com'

}
```

6.5.2 Descobrir os peers que possuem o arquivo solicitado

De posse do arquivo de meta dados, o cliente utiliza o hash para perguntar ao servidor *tracker* utilizando a informação definida no campo *announce*.

O servidor responde com uma lista de IPs e portas que possuem o arquivo, como pode ser visto abaixo:

```
{
        'i89iko372uoijew7uw8yi3ho' : [
                '12.83.30.30:3000',
                '54.87.93.30:3040'
        ]
}
```

6.5.3 Baixando o conteúdo desejado

Ao receber a lista de IPs, o cliente começa a baixar cada pedaço do arquivo. Para isso o mesmo envia uma requisição e espera pela resposta definidas abaixo:

```
BitTorrent request: REQUEST 0  0 12392103
```

Onde o primeiro campo é o método (similar a um request HTTP), o segundo o índice do pedaço sendo baixado. O terceiro é o *offset* em bytes. O quarto é o tamanho a ser baixado.

```
BitTorrent response: PIECE 0 0 \textless DATA\textgreater
```

Onde o primeiro campo é o método definindo que o dado chegando é um pedaço do arquivo. O segundo é o índice do pedaço, o terceiro o offset de bytes.

Lembrando que cada pedaço do arquivo tem seu indice e o hash associado para a verificação de integridade do dado transferido.

6.6 O tracker

O tracker é o servidor que informa quais nós possuem quais pedaços do arquivo a ser baixado. Ele faz o trabalho de guardar os ips e portas de cada nó na rede. Este trabalho apesar de ser uma centralização pode trazer benefícios como por exemplo realizar agrupamento de nós em clusters, para beneficiar ou privilegiar o trafego interno nos *ISPs*.

Mas outra abordagem possível não implementada é a de se utilizar de DHTs. Estas tabelas eliminam a necessidade de haver o nó central. Estas estruturas são complexas e ainda não foram implementadas para funcionarem sobre WebRTC.

6.7 A página

Na solução proposta a página html com as imagens disponibilizadas no campo "data-src"faz na verdade uma coleção de arquivos .torrent.

```
<img data-src="http://www.site.com.br/image.jpg">
```

Para cada "data-src"encontrado no código, um novo .torrent com apenas 1 pedaço será criado, e assim o usuário baixará o conteúdo de cada um através da rede WebRTC instalada na máquina. Em uma expansão da solução se poderia incluir informações de hash diretamente no lugar das URLs das imagens, assim o passo de tradução da URL em um arquivo .torrent pode ser pulado, salvando tempo e processamento.

6.8 O servidor seed inicial

Como a página deve estar disponível mesmo que os clientes WebRTC não estejam presentes, se faz necessário a utilização de um cliente não ligado ao navegador, para disponibilizar as conexões e prover os arquivos ao primeiro usuário vindo do navegador. Ele também será o responsável por traduzir as URLs de cada imagem em hashes após baixar o conteúdo das mesmas. Desta maneira a rede fica assegurada, pois o textitseeder sempre estará presente.

6.9 A jornada de dados na solução proposta

6.9.1 O setup inicial da solução

Como dito anteriormente para uma página utilizar a solução proposta duas ações são necessárias sobre o conteúdo da mesma:

- Adicionar o script da solução ao código HTML da página. Isto é feito utilizando-se o seguinte comando:

```
<script src="http://javascript.carlosbpf.com.br/WebRTCProxyCacheJS
    .js"></script>
```

- Alterar **todos** os tags de imagens para não utilizarem o campo *src* mas sim o campo *data-src* como definido e explicado anteriormente

```
<img data-src="http://www.site.com.br/image.jpg">
```

  No primeiro acesso ao site após as mudanças serem salvas, a URL do script será chamada. Esta URL não é para um arquivo estático, e sim para um servidor de aplicativos web, que:

  1. Verifica se o servidor já possui um arquivo de índice para o site, o endereço do site é descoberto pelo campo de origem *referal* do protocolo HTTP.
  2. Se já tiver o índice, retorna este índice em uma variável no script concatenada com o resto da solução.
  3. Se não tiver o índice definido, o servidor inicia uma thread que irá rastrear o código do servidor de origem da chamada, para encontrar todas as referencias a imagens externas.
  4. Para cada imagem, esta thread irá criar um arquivo *.torrent*, e deste arquivo será extraído o *magnet*, uma representação do arquivo de metadados em formato de URL, a dupla do caminho completo da imagem e seu *magnet* são salvos no arquivo de índice.
  5. Após esta lista estar pronta o servidor sempre retornará no passo 2. o código acrescentado da lista de mapeamento das imagens do site.

Após esta etapa inicial o algoritmo pode utilizar-se da solução proposta. Até que esta lista seja compilada corretamente os acessos ao site terão a estratégia de contingência de utilizar-se do contúdo servido de maneira tradicional, via HTTP.

Para criar o torrent de cada imagem o cliente de linha de comando da solução é utilizado. Ao fazer isso os arquivos ficam automaticamente disponíveis para compartilhamento, e o cliente se torna o *seeder* original. Isto é importante pois no acesso do primeiro visitante, o conteúdo não teria nenhum outro nó para baixar o conteúdo. Desta forma sempre que um visitante entrar na rede, este nunca estará sozinho, o cliente seeder sempre estará disponível e com todo o conteúdo baixado e apto a compartilhamento.

6.9.2 A jornada para cada novo visitante no site

1. O script da solução é executado antes de qualquer outro script na página.
2. O script le calcula as imagens que devem ser exibidas na posição gráfica atual da página, e inicia o seu download através da rede P2P
3. Para cada imagem o mapeamento URL/magnet é feito e o torrent é aberto e a sinalização do protoloco bit torrent se inicia
4. O nó do visitante é adicionado a rede, através da troca de sinalização.
5. O nós que possuem a imagem são descobertos com o auxílio do tracker.
6. Assim que um *peer* aceita que o conteúdo seja transferido o *download* se inicia
7. A imagem é baixada, e o evento de finalização do *download* é lançado
8. O script sabe a tag de origem da imagem, e a insere diretamente na página usando a notação de base 64
9. A partir desse momento o nó pode começar a enviar a imagem baixada para outros nós caso houver a requisição.
10. Após um movimento da posição gráfica da página mais imagens podem ter seu *download* iniciado.
11. Enquanto a aba do navedor deste visitante se mantiver aberta, o mesmo estará provendo as imagens que já tenha baixado.

Mesmo que ocorra um problema com o *seeder* a implementação cai para a utilização de HTTP como estratégia de contigência.

O próximo capítulo demonstra as características de performance da solução contrastadas com a tecnologia tradicional.

Capítulo 7

Caso de uso

7.1 Caso de uso

Como teste para a plataforma foi utilizado duas versões de três portais de grande acesso no Brasil o uol.com.br, g1.com.br, r7.com. A escolha de portais se dá pelo fato de portais terem uma grande quantidade de imagens e a requisição da página completa ter um tamanho grande, em média de 2 a 6 Mb se todas as mídias forem somadas. E também porque seu tempo total de abertura, até o lançamento do evento *ready* ser lançado, é considerável, em torno de 1 a 2 minutos.

O conteúdo completo da página foi salvo em disco e disponibilizado através de um servidor *web* padrão *Apache*.

O tracker foi implementado em *nodejs* e foi executado em um servidor de aplicação *node*.

O código para o cliente sem navegador , o *seeder* inicial foi executado na mesma máquina do servidor de dados original.

Os três aplicativos foram executados em uma mesma máquina virtual hospedada na plataforma de computação em nuvem da *Amazon*.

Os clientes foram computadores conectados na mesma rede, com acesso via internet ao servidor hospedado na *Amazon*, onde foi usado para a execução o navegador google chrome. O site uol.com.br foi salvo no dia 21 de Julho de 2016 . O site g1.com.br foi salvo no dia 15 de Janeiro de 2017. O site r7.com foi salvo no dia 15 de Janeiro de 2017.

Todo o seu conteúdo dos portais foi disponibilizado no servidor de aplicação na *Amazon*. Como pode ser visto nas figuras 7.1 os sites são totalmente funcionais no navegador Chrome.

- PC1**Macbook pro 2011**
- PC2**Macbook pro 2015**
- PC3**Notebook Asus eepc**
- PC4**Notebook Dell Vostro**
- PC5**Notebook Toshiba**

Foram executado dois testes: Ambos foram realizados de forma automatizada com 1000 acessos para evitar flutuações de performance da rede.

Figura 7.1: Captura de tela do site uol.com.br hospedado no servidor dedicado para os testes

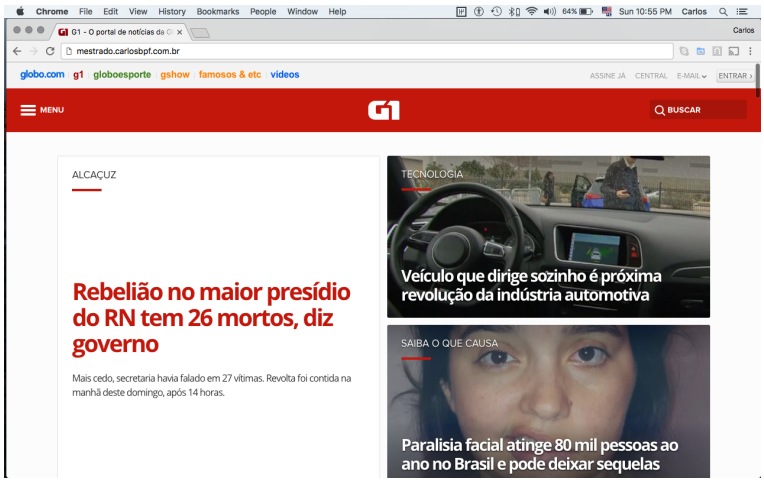

Figura 7.2: Captura de tela do site g1.com.br hospedado no servidor dedicado para os testes

- **Teste 1 :** O site sem alteração nenhuma em seu conteúdo será submetido ao acesso de 1 navegador e 5 navegadores. Foi medido no cliente a quantidade de dados trafegados e tempo total de download do conteúdo. Foi medido no servidor a quantidade total de dados trafegados. A figura 7.4 ilustra o teste 1, o servidor a direita foi submetido ao acesso de apenas 1 computador, Teste 1.1 . NoTeste 1.2 o mesmo servidor foi submetido ao acesso de 5 computadores.

  O site do uol salvo em disco possui 301 imagens. Como pode ser visto na figura 7.5 capturada do aplicativo *Developer Tools* do chrome no PC1. Além disso neste teste a página foi totalmente processada em 13.73 segundos.

  O site do g1 salvo em disco possui 62 imagens. Como pode ser visto na figura 7.6 capturada do aplicativo *Developer Tools* do chrome no PC1. Além disso neste teste a página foi totalmente processada em 6.26 segundos.

Figura 7.3: Captura de tela do site r7.com hospedado no servidor dedicado para os testes

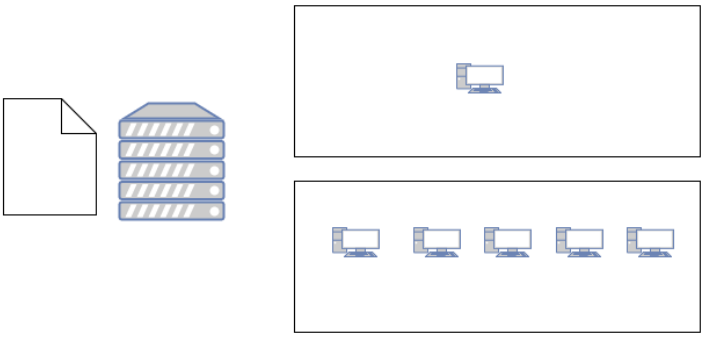

Figura 7.4: Configuração do teste 1, e suas variações, acesso de 1 , 5 e 10 clientes, sem a rede P2P

O site do r7 salvo em disco possui 399 imagens. Como pode ser visto na figura 7.5 capturada do aplicativo *Developer Tools* do chrome no PC1. Além disso neste teste a página foi totalmente processada em 27.99 segundos.

- **Teste 2 :** O site com a alteração das urls das imagens, e com a inclusão do script de *p2pwebcaching* em seu conteúdo foi submetido ao acesso de 1 navegador e 5 navegadores, as abas nos navegadores foram mantidos acessando a página e provendo o conteúdo durante todo o teste. Foi medido no cliente a quantidade de dados trafegados e tempo total de download do conteúdo. Foi medido no servidor a quantidade total de dados trafegados. Como pode ser visto na figura 7.8

A figura 7.9 abaixo exibe o painel de estatísticas e debug do PC5 após o final da aquisição de todo o conteúdo do uol. Nele é possível ver os outros nós listados , incluindo o servidor seed original. Abaixo pode-se ver a pilha de eventos que ocorreram no decorrer da execução das chamadas feitas pela solução.

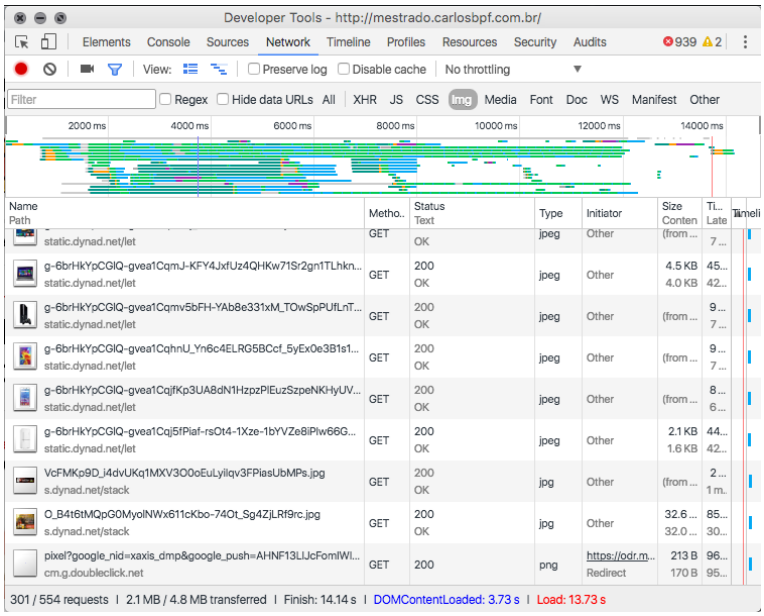

Figura 7.5: Captura de tela do site uol.com.br hospedado no servidor dedicado para os testes

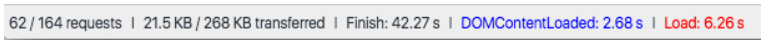

Figura 7.6: Captura de tela do resumo das imagens do site g1.com.br hospedado no servidor para testes

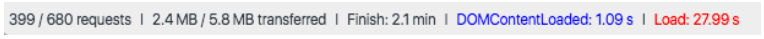

Figura 7.7: Captura de tela do resumo das imagens do site r7.com hospedado no servidor dedicado para os testes

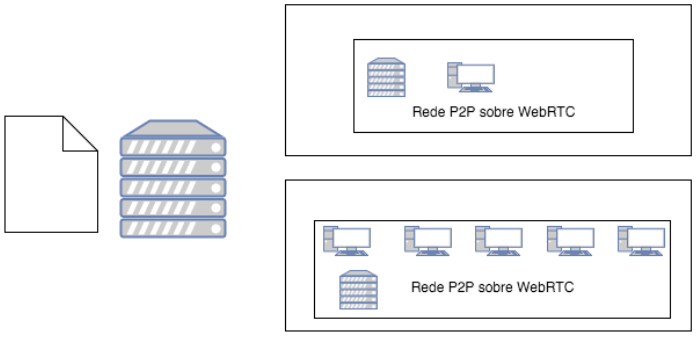

Figura 7.8: Configuração do teste 2, e suas variações, acesso de 1 e 5 clientes , com a rede P2P de cada nó adentrando assim que tiver o conteúdo baixado

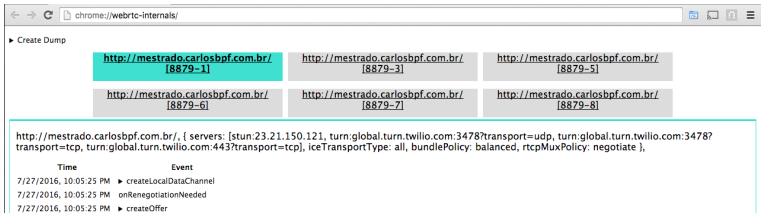

Figura 7.9: Página de estatísticas e debug do webrtc no navegador chrome

7.2 Resultados

Com os dados colhidos do teste pode-se avaliar a eficiência da estratégia comparada a um site tradicional nos seguintes aspectos:

- **Tempo total de abertura da página** : Para avaliar se houve impacto na percepção de velocidade para o usuário final. Ao se comparar os 2 cenários de quantidade de usuários, pode-se também extrair como o número de nós afeta a performance da solução proposta.
- **Numero total de dados trafegados no cliente** : Para avaliar se o número de dados trafegados no cliente foi maior em qual cenário da solução, e com quanto número de nós associados a rede P2P.
- **Número total de dados trafegados no servidor** : Para avaliar como foi impacto da solução no servidor original.
- **Custo total para o dono do site** : Será computado o valor de custo (que em geral está associado a quantidade de dados transferidos, poder computacional gasto pelas máquinas virtuais e suas operações em disco e memória utilizados). Esse valor além de ser computado para cada um dos casos será também confrontado com a média de todos os acessos ao portal. Pois existe a suposição do sistema evoluir de forma linear, assim pode-se ter uma ideia de como a solução performaria em um cenário real de maior utilização.

7.2.1 Teste 1

No teste 1 o servidor se comporta de maneira padrão, servindo o conteúdo a cada requisição. Da requisição HTML , *scripts* e planilhas de estilo originais a cada uma das imagens.

Análise dos dados para o portal uol.com.br

Para o uol, 301 imagens, O teste 1.1 ilustra o acesso de apenas 1 consumidor. A tabela abaixo 7.1 exibe os resultados do teste 1 para o site uol que serão usados como referência.

Análise dos dados para o portal g1.com.br

Para o G1, 62 imagens, O teste 1.1 ilustra o acesso de apenas 1 consumidor. A tabela abaixo 7.2 exibe os resultados do teste 1 para o site G1 que serão usados como referência.

Tabela 7.1: Resultados para o Teste 1.1 e 1.2 para o uol.com.br

	Teste 1.1	Teste 1.2 total	Teste 1.2 médio
Tempo total de abertura da página	14.14s	69.21s	13.84s
Total de dados trafegados no cliente	2.1 MByte	11.12 MByte	2.25 MByte
Total de dados trafegados no servidor	2.18 MByte	11.1619 MByte	2.243 MByte
Custo total para os dados trafegados	USD $ 0.12	USD $ 0.65	USD $ 0.13

Tabela 7.2: Resultados para o Teste 1.1 e 1.2 para o g1.com.br

	Teste 1.1	Teste 1.2 total	Teste 1.2 médio
Tempo total de abertura da página	6.26s	32.5s	6.5s
Total de dados trafegados no cliente	268 KByte	1.13 MByte	260. KByte
Total de dados trafegados no servidor	290 KByte	1.19 MByte	265.43 KByte

Análise dos dados para o portal r7.com

Para o R7, 399 imagens, O teste 1.1 ilustra o acesso de apenas 1 consumidor. A tabela abaixo 7.3 exibe os resultados do teste 1 para o site R7 que serão usados como referência.

Tabela 7.3: Resultados para o Teste 1.1 e 1.2 para o r7.com

	Teste 1.1	Teste 1.2 total	Teste 1.2 médio
Tempo total de abertura da página	27.99s	133.85s	26.77s
Total de dados trafegados no cliente	2.4 MByte	12.5 MByte	2.5 MByte
Total de dados trafegados no servidor	2.5 MByte	13.16 MByte	2.64 MByte

O preço total para o UOL é baseado no valor de dados transferidos cobrados pela plataforma AWS da Amazon. Como o preço é baseado numa unidade muito maior do que a que foi testada os valores exibidos são multiplicados por 100000. O valor para 50 TB é o de $0.0290 USD.Ele foi calculado apenas para o uol pois o mesmo foi o unico a divulgar publicamente o número de acessos ao site, que será descrito nas próximas seções.

7.2.2 Teste 2

Para o teste 2 foi utilizado o conjunto de computadores descritos no início deste capítulo. Cinco computadores no total, onde cada computador acessou a página e permaneceu após o fim do *donwload* total da página. Assim a tabela 7.4 exibe os resultados a medida que o computador acessou a página do uol e entrou na rede P2P. No primeiro acesso apenas o servidor está na rede P2P , por isso o valor da coluna PC1 tem o valor de dados trafegados apenas para a linha do servidor. Para o PC2 ambas as linhas PC1 e Servidor tem dados trafegados . A mesma lógica se aplica aos outros PCs. No PC5 todos os computadores prévios estão contribuindo para a rede menos o próprio PC5 que ainda está se comportando apenas como *leecher* .

Apesar de haver uma pequena variação entre o total de dados trafegados, que muitas vezes tem a ver com a variação das rotas entre os computadores na rede P2P, a média fica bem proxima.

Como pode ser observado pelas tabelas 7.5 7.6 7.7, temos uma perda de performance em total de abertura da página. E há também um aumento na quantidade de dados

Tabela 7.4: Tabela de dados trafegados a medida que o site uol foi acessado

	PC1	PC2	PC3	PC4	PC5
Servidor	2543.61	1165.82	846.31	582.85	478.90
PC1		1254.09	846.23	635.32	475.18
PC2			775.57	635.32	502.76
PC3				594.15	563.69
PC4					469.91
PC5					
Total	**2543.61**	**2419.91**	**2468.11**	**2447.64**	**2490.46**

Tabela 7.5: Resultados para o Teste 2.1 e 2.2 para o uol.com.br

	Teste 1	Teste 2.1	Teste 2.2 médio
Tempo total de abertura da página	14.14s	15.58s	15.40s
Total de dados trafegados no cliente	2.1 MByte	2.54 MByte	2.47 MByte
Total de dados trafegados no servidor	2.18 MByte	2.54 MByte	1.12 MByte
Custo total para os dados trafegados	USD $ 0.12	USD $ 0.15	USD $ 0.07

Tabela 7.6: Resultados para o Teste 2.1 e 2.2 para o g1.com.br

	Teste 1	Teste 2.1	Teste 2.2 médio
Tempo total de abertura da página	6.26.14s	7.42s	7.28s
Total de dados trafegados no cliente	268 KByte	296 KByte	286 KByte
Total de dados trafegados no servidor	290 KByte	296 KByte	286 KByte

Tabela 7.7: Resultados para o Teste 2.1 e 2.2 para o r7.com

	Teste 1	Teste 2.1	Teste 2.2 médio
Tempo total de abertura da página	27.99s	32.36s	30.16s
Total de dados trafegados no cliente	2.4 MByte	2.64 MByte	2.51 MByte
Total de dados trafegados no servidor	2.5 MByte	2.64 MByte	2.51 MByte

trafegados no cliente, cerca de 16% . Todavia a quantidade de dados no servidor caiu vertiginosamente. Um diminuição de 51% Assim é possível dizer que a solução realizou uma economia no servidor, mas com a contra partida de ter aumentado a quantidade de dados e também do tempo total de abertura da página.

Apenas para se ter uma idéia do volume da economia da solução, o próprio site do uol afirma que tem um número de acessos mensais de 1 bilhão de acessos. Ao se multiplicar pelo valor que a AWS *Amazon* de USD 0.09 para cada GByte de dados trafegados. O custo estimado mensal do UOL é de aproximadamente : USD 570000 . Caso a solução se comportasse de maneira semelhante ao teste 2.2 no ambiente real de produção do UOL, a economia total seria de 250 mil dólares. Claro que é esperado que a economia não se mantenha na mesma proporção mas esta conta aproximada mostra o potencial de economia desta solução.

Capítulo 8

Conclusão

Conforme apresentado no Capítulo 2 o processo de *caching* é bastante importante para a diminuição de custos e para a velocidade da internet como um todo. E conforme trabalhos correlatos confirmam, sua eficiência vem diminuindo com a mudança da origem de conteúdo, assim ,novas abordagens se fazem necessárias.

A proposta deste trabalho incorpora conceitos que já estão sendo utilizados para *cache* utilizando redes P2P. Porém realiza este trabalho em uma camada muito mais numerosa de nós, no próprio cliente.

E como a implantação dessa tecnologia não demanda nenhuma alteração nos protocolos e navegadores atuais, a mesma pode ser utilizada por quase todos os usuários da internet atual.

Os testes visaram estudar o impacto da implementação atual comparados com a abordagem atual, e esta performance evidenciou que a solução proposta consegue executar a *cache* diminuindo muito a quantidade de dados no servidor mas para os clientes a quantidade de dados aumentou mas não de forma muito grande. Ao mesmo tempo foi óbvio que no teste de apenas 1 pc na rede e com 5 pcs realizaram um impacto muito maior na rede, mostrando que potencialmente quanto maior a quantidade de nós maior será a economia no servidor, respeitando o princípio de Paretto.

8.1 Trabalhos futuros

O trabalho apresentado pode ser estendido para utilizar-se da mesma implementação para prover conteúdos de vídeo, que devem beneficiar-se bastante da redução de custo visto o tamanho que costumam ter é significantemente maior que o de imagens. E o impacto geral na percepção de velocidade pode se tornar ainda maior pois o conteúdo de vídeo costuma frequentemente ser o gargalo em uma navegação de conteúdos em sites. Outra extensão possível é a de se inserir na lógica de compartilhamento não somente as midias tradicionais mas também todo o conteúdo provido pelo site. Como pontencialmente todo o conteúdo, seja ele textual ou em formato HTML pode ser *cacheado*, pode-se em teoria ter um site completamente disponível sem a necessidade de um servidor central, como é feito hoje, apenas com a implementação da rede P2P se poderia solicitar o conteúdo do portal, e ele seria montado pelos nós que teriam seu conteúdo altamente distribuido, tornando assim sites altamente difíceis de serem bloqueados ou censurados.

Referências

[1] The bittorrent protocol specification. Technical report, Bittorrent Inc., 2009. 33

[2] Cullen Jennings Anant Narayanan Adam Bergkvis, Daniel C. Burnett. Webrtc specification. W3C Working Draft 10 February 1.5610, World Wide Web Consortium - W3C, 2015. http://www.w3.org/TR/webrtc/. 36

[3] Bernhard Ager, Fabian Schneider, Juhoon Kim, and Anja Feldmann. Revisiting Cacheability in Times of User Generated Content. pages 0–5, 2010. 1, 10, 23

[4] Mudashiru Busari and Carey Williamson. On the sensitivity of web proxy cache performance to workload characteristics. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1225–1234. IEEE, 2001. 11

[5] John W Byers, Michael Luby, and Michael Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 275–283. IEEE, 1999. 11

[6] Kideok Cho, Munyoung Lee, Kunwoo Park, Ted Taekyoung Kwon, Yanghee Choi, and Sangheon Pack. Wave: Popularity-based and collaborative in-network caching for content-oriented networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 316–321. IEEE, 2012. 22

[7] Manal El Dick, Esther Pacitti, Reza Akbarinia, and Bettina Kemme. Building a peer-to-peer content distribution network with high performance, scalability and robustness. *Information Systems*, 36(2):222–247, 2011. 11

[8] Manal El Dick, Esther Pacitti, Reza Akbarinia, and Bettina Kemme. Building a peer-to-peer content distribution network with high performance, scalability and robustness. *Information Systems*, 36(2):222–247, 2011. 23

[9] Anja Feldmann, Ramon Caceres, Fred Douglis, Gideon Glass, and Michael Rabinovich. Performance of web proxy caching in heterogeneous bandwidth environments. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 107–116. IEEE, 1999. 11

[10] CACERES R. DOUGLIS F. GLASS G. FELDMANN, A. and RA-BINOVICH. Performance of web proxy caching in heterogeneous bandwidth environments. *In Proc. IEEE INFOCOM (1999)*, pages 0–5, 2010. 32

[11] Shen Fuke and Zhu Yuhong. A hybrid p2p-cdn based on locality-awareness and interest-awareness. *Information Technology Journal*, 12(3):403, 2013. 23

[12] Ian Hickson. Html5 specification. first Edition of a Recommendation 1.5610, World Wide Web Consortium - W3C, 2012. http://dev.w3.org/html5/spec/single-page.html. 36

[13] Sandvine Inc. Sandvine. Global internet phenomena report - africa, middle east north america, Sandvine Inc., 2016. https://www.sandvine.com/trends/global-internet-phenomena/. 2, 16

[14] Zhimei Jiang and Leonard Kleinrock. Web prefetching in a mobile environment. *Personal Communications, IEEE*, 5(5):25–34, 1998. 11

[15] Shudong Jin and Azer Bestavros. Popularity-aware greedy dual-size web proxy caching algorithms. In *Distributed computing systems, 2000. Proceedings. 20th international conference on*, pages 254–261. IEEE, 2000. 11

[16] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker. Web caching and zipf-like distributions: evidence and implications. *NFOCOM '99. Eighteenth Annual Joint Con- ference of the IEEE Computer and Communications Societies*, 1:126–134, 1999. 11

[17] Wei-Kuo Liao and Chun-Ta King. Proxy prefetch and prefix caching. In *Parallel Processing, 2001. International Conference on*, pages 95–102. IEEE, 2001. 11

[18] Anirban Mahanti, Carey Williamson, and Derek Eager. Traffic analysis of a web proxy caching hierarchy. *Network, IEEE*, 14(3):16–23, 2000. 11

[19] Kazuki Matsushita, Masashi Nishimine, and Kazunori Ueda. Cooperative cache distribution system for virtual p2p web proxy. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 3, pages 646–647. IEEE, 2015. 22

[20] Sergey Mavrody. *Sergey's HTML5 & CSS3 Quick Reference*. Belisso Corporation, 2012. 27

[21] Masashi Nishimine and Kazunori Ueda. Design and Implementation of a Cache-less P2P Web Proxy. *2013 Eighth International Conference on Broadband and Wireless Computing, Communication and Applications*, pages 500–505, October 2013. 32

[22] M. Nottingham R. Fielding and J. Reschke. RFC 1866. Technical report, RFC Editor. 25

[23] M. Nottingham R. Fielding and J. Reschke. RFC 7234. Technical report, RFC Editor. 7, 9, 11

[24] Jan Seedorf, Sebastian Kiesel, and Martin Stiemerling. Traffic localization for p2p-applications: the alto approach. In *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pages 171–177. IEEE, 2009. 23

[25] Guobin Shen, Ye Wang, Yongqiang Xiong, Ben Y Zhao, and Zhi-Li Zhang. Hptp: Relieving the tension between isps and p2p. In *IPTPS*. Citeseer, 2007. 23

[26] Zheng Zhao, Jie Yang, Song Wang, Gang Zhang, and Yantai Shu. Measurement based intelligent prefetch and cache technique in web. In *Electrical and Computer Engineering, 1999 IEEE Canadian Conference on*, volume 1, pages 168–173. IEEE, 1999. 11