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

Explorando Similaridade em URL de Conteúdo Dinâmico como Alternativa para Aumentar a Taxa de Acertos no Web Cache Institucional

Marcelo Monte Karam

Dissertação apresentada como requisito parcial
para conclusão do Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Jacir Luiz Bordim

Brasília
2014

Ficha catalográfica elaborada pela Biblioteca Central da Universidade de Brasília. Acervo 1018296.

Karam, Marcelo Monte.
K18e Explorando similaridade em URL de conteúdo dinâmico como alternativa para aumentar a taxa de acertos no web cache institucional / Marcelo Monte Karam. -- 2014.
xi, 86 f. : il. ; 30 cm.

Dissertação (mestrado) - Universidade de Brasília, Instituto de Ciências Exatas, Departamento de Ciência da Computação, 2014.
Inclui bibliografia.
Orientação: Jacir Luiz Bordim.

1. Web caching. 2. Memória cache. 3. Uniform Resource Locators. 4. Servidores proxy web. I. Bordim, Jacir Luiz. II. Título.

CDU 004.738.5

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

Explorando Similaridade em URLs de Conteúdo Dinâmico como Alternativa para Aumentar a Taxa de Acertos no Web Cache Institucional"

Marcelo Monte Karam

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Jacir Luis Bordim (Orientador)
CIC/UnB

Prof. Dr. Marcelo Ladeira
CIC/UnB

Prof. Dr. Laerte Peotta de Melo
Banco do Brasil

Prof. Dr. Marcelo Ladeira
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 08 de julho de 2014

Dedicatória

Dedico este trabalho a minha amada família:

- Ao meus pais, Telma e Ibrahim, pelo exemplo e pelo carinho recebido em todos estes anos;
- A minha esposa, Débora, pelo apoio, dedicação, paciência e coragem. Coragem esta que nos abençou com uma mais uma filha ao longo deste trabalho;
- As minhas filhas, Maria Eduarda e Ana Luíza, pela alegria e amor incodicionais. Mesmo quando eu não estava presente ou dando-lhes a atenção devida, sempre me deram motivação e energia para continuar.

Agradecimentos

Gostaria inicalmente agradecer a Deus por poder ter tido esta oportunida e vivido-a; Ao meu orientador prof. Dr. Jacir Bordim, pelas orientações, paciência, disponibilidade e companherismo, sempre me incentivando ir além; Ao prof. Dr. Marcelo Ladeira coordenador do programa pelos conselhos e ensinamentos; Ao professores(as) Dr.(a) do mestrado profissional, André, Letéia, Priscila, Marcelo Ladeira, Jacir Bordim, Jorge Fernandes, Joca, que nos ensiaram valiosas lições sobre os mais variados temas, sempre buscando um olhar mais crítico; Aos membros da Banca, em particular ao Prof. Marcelo e Laarte Peota pelos comentários e discussões que contribuíram para a evolução deste trabalho; Ao meu amigo e chefe Domingos Pereira pelo incentivo e amizade.

Resumo

Servidores *proxy web cache* apresentam vantagens significativas em termos do tempo necessário para recuperar objetos além de permitir uma melhor utilização da banda disponível. No entanto, sistemas convencionais de *proxy web cache* não dispõem de mecanismos para tratar URL (*Uniform Resource Locator*) dinâmicas, não conseguindo assim identificar URLs equivalentes que referenciam um mesmo objeto. No contexto da Universidade de Brasília, verifica-se uma grande demanda por conteúdo representado por URLs dinâmicas, em especial para conteúdo de vídeo. Com o intuito de permitir o reuso destes objetos, essa dissertação propõe a implementação e avaliação de um sistema de *proxy web cache* institucional. Para atingir este objetivo, técnicas de transformação de URL dinâmica em URL estáticas foram avaliadas e implementadas. O sistema *proxy web cache* resultante apresentou, nos testes realizados, taxas de reuso acima de 99% e significativa redução no tempo de recuperação destes objetos. Estes resultados denotam um aumento na capacidade do sistema de *proxy web cache* em comparação com abordagens convencionais no tratamento e recuperação de objetos representados por URLs dinâmicas.

Palavras-chave: *Web Caching*, *Cache*, Equivalência de URLs Dinâmicas, *Proxy*, Decodificação e Sumarização de URLs Dinâmicas e *Proxy Web Cache* Institucional.

Abstract

Web proxy cache servers provide significant advantages in terms of time required to retrieve objects while allowing better use of the available bandwidth. However, conventional web proxy cache systems are not able to deal with dynamic URLs (Uniform Resource Locator) as they cannot distinguish equivalent URLs that refer to the same object. In the context of the University of Brasília, there is a large demand for content represented by dynamic URLs, especially for video content. In order to allow the reuse of these objects, this dissertation proposes the implementation and evaluation of an institutional web proxy cache system. To achieve this goal, transformation techniques, from dynamic URL into static URLs, were evaluated and implemented. The resulting web proxy cache system attained, in the evaluated scenario, reuse rates above 99% and a significant object-retrieval-time reduction. These results show an increase in the ability of a web caching proxy system compared with conventional approaches in the treatment and recovery of objects represented by dynamic URLs.

Keywords: *Web Caching*, *Cache*, *Equivalence URLs Dynamic*, *Proxy*, *Decoding and Summarization URLs Dynamic* and *Proxy Web Cache Institutional.*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nos últimos 10 (dez) anos a popularização da Internet se intensificou, acomodando a cada ano em média 244 (duzentos e quarenta e quatro) milhões de novos usuários [1]. Em consequência deste aumento, se faz necessárias expansões consideráveis na infraestrutura de TIC's (Tecnologias da Informação e Comunicação), o que gera custos financeiros adicionais à instituições e provedores de serviços.

Buscando minimizar estes custos e racionalizar os recursos, foi desenvolvido em 1997 a primeira ferramenta de *proxy web cache*, o *Squid* [2]. Esta tecnologia foi inicialmente projetada para reuso de objetos pequenos de páginas estáticas acessadas através de URLs estáticas pelo protocolo HTTP (do Inglês, *Hypertext Transfer Protocol*) [3] a partir de uma rede local ou de um provedor de acesso [4].

Com evolução da Internet tanto o conteúdo quanto as URLs tornaram-se dinâmicos, sendo esta última normalmente composta por parâmetros passados por aplicações. No geral os parâmetros passados nas URLs dinâmicas são relativos à aplicações e ou informações de usuários.

Grandes provedores de serviço da Internet, tais como *YouTube*, *Facebook*, motores de pesquisas, entre outros, utilizam URLs dinâmicas agregadas com CDN (*Content Delivery Network*). O CDN [5] tem como principal característica disponibilizar o mesmo conteúdo em diversos locais diferentes, resultando em ganho considerável na disponibilidade e re-

dução no tempo de acesso a esses conteúdos, e consequentemente a melhoria na qualidade dos serviços ofertados. Normalmente as CDNs são compostas de servidores distribuídos geograficamente, atendendo as solicitações dos clientes, baseado em localização geográfica e carga.

É facilmente encontrando nos CDNs o uso de URLs dinâmicas. Apesar das técnicas de URLs dinâmicas e CDNs corroborarem bastante para disponibilização dos serviços de uma forma transparente e eficiente aos seus usuários, acabam por inviabilizarem o uso performático de sistemas de *proxy web cache* convencionais. As duas técnicas produzem URLs distintas, que apontam para o mesmo objeto, gerando assim as URLs equivalentes.

O serviço mais popular de vídeo pela Internet, o *YouTube*, repassa através das URLs dinâmicas informações sobre o cliente, tais como sistema operacional, navegador, velocidade da conexão, bem como as solicitações de vídeo, especificando a qualidade, parte do vídeo, entre outras caraterísticas. Por mais que milhares de pessoas assistam ao mesmo vídeo, nas mesmas condições de definição, tipo de navegador, entre outros, não será possível ao sistema *proxy web cache* convencional reutilizar os objetos destes vídeos.

O que se tem observado na Internet é o crescimento do uso de serviços de vídeos, tornando-os cada vez mais populares. Este fenômeno pode ser explicado em parte pela migração do formato da informação, sendo passada de texto para vídeos, acarretando um considerável consumo de banda na rede local. Outro ponto a ser levantado acerca deste serviço é a dificuldade em sua categorização, dificultando assim qualquer tipo de restrição.

Procedimentos convencionais para salvaguarda de banda, tais como bloqueio de conteúdo, técnicas de *Traffic Shape* e *proxy web cache* convencionais são ineficazes para serviços modernos de Internet, principalmente tratando-se de vídeos. *Traffic Shape* [4] é uma técnica que busca limitar em banda o uso de um dado serviço. O serviço de vídeo atualmente é utilizado tanto como entretenimento, quanto para difusão de conhecimento, com a imposição de limitações na banda, tem-se como consequência a queda na qualidade do serviço durante o uso legítimo. Serviços como *YouTube*, se tornaram desafios a qualquer administrador de redes, tendo a difícil missão de equalizar o provimento de serviços

de qualidade, em detrimento ao consumo de banda.

Neste trabalho foram levantadas propostas na literatura e em projetos de código aberto com foco em sistemas de *proxy web cache* capazes de tratar URLs dinâmicas, produzindo ganhos na economia do consumo de banda e na latência média. As propostas levantadas foram analisadas, de acordo com sua complexidade, e no caso dos projetos de código aberto a sua atividade.

A proposta escolhida foi pautada na necessidade de agregar soluções especializadas em serviços/aplicações que tratem URLs dinâmicas nos proxy web caches convencionais. Sendo para isto, utilizada as técnicas de decodificação, sumarização e agregação de URLs dinâmicas. Com usos destas técnicas se espera apontar as URLs equivalentes para um mesmo objeto, dando-lhe a oportunidade de ser reutilizado.

Para validação e mensuramento da eficácia da proposta escolhida foram executados testes baseados no serviço de vídeos do *YouTube* em navegadores distintos. O serviço de vídeos do *YouTube* foi escolhido devido à sua complexidade e popularidade. Os vídeos escolhidos para os testes se basearam na popularidade de parâmetros, como qualidade, duração e codificação, levantados em trabalhos anteriores. Foram escolhidos 28 (vinte oito) vídeos mais populares, mesclando os vídeos com maior volume de acesso desde 2007 e os mais populares no Brasil em abril de 2014. Foram escolhidos os três navegadores mais populares para exibição dos vídeos, objetivando assim identificar o impacto do uso de navegadores distintos.

A análise foi baseada em testes em 2 (dois) cenários diferentes, com uso ou ausência da implementação da proposta. Em ambos cenários os 28 (vinte e oito) vídeos foram exibidos nos três navegadores estudado, tendo o segundo cenário repetido seus testes três vezes a fim de consolidar os resultados. Os vídeos foram apresentados em 1 (uma) das 6 (seis) qualidades pré-definidas. Como métricas foram utilizadas as taxas de reuso de objetos baseada em requisição e em volume, além do tempo médio de recuperação dos mesmos.

Como será visto mais a frente observou-se da análise dos testes resultados animadores e bem promissores.

1.1 Motivação

Contribuir com o Centro de Informática da Universidade de Brasília, no provimento de serviços de conectivade de Internet com uso racionalizado da banda de Internet trazendo melhorias na experiência na navegação *web*.

1.2 Problema

Os sistema convencionais *proxy web cache* institucional em sua configuração padrão não dispõem de mecanismos para tratar URLs dinâmicas, não conseguindo assim identificar URLs equivalentes que referenciam um mesmo objeto. Como consequência tem-se taxas não significativas de acertos *hits* e dificuldade em manter ou diminuir a latência média durante a requisição dos objetos, de forma a limitar o sistema de *proxy web cache* convencional e diminuindo a qualidade de experiência do usuário.

1.3 Objetivos

O objetivo principal dessa pesquisa é identificar propostas eficientes de avaliação e implementação de *proxy web cache* institucional com foco em URLs dinâmicas, que proporcionem o aumento na taxa de acerto de *hits* e mantenha a latência média, avaliando a de melhor aderência. Como objetivos específicos, busca-se:

- Levantar o estado da arte dos trabalhos relacionados com sistema de *proxy web cache* com foco: em URLs dinâmicas e análise de métricas para avaliação da eficiência;
- Evidenciar as dificuldades do sistema de *proxy web cache* convencional em tratar URLs dinâmicas em um ambiente de produção;

- Levantar e identificar propostas de implementação de sistemas de *proxy web cache* institucionais, baseadas em código aberto, que tratem URLs dinâmicas permitindo *cache* de seus objetos;

- Efetuar implementação básica da proposta de melhor aderência explanando seu funcionamento, explicitando sua tecnologia e seus métodos de operação;

- Efetuar teste de eficiência na proposta selecionada, com base em métricas previamente levantadas a fim de demonstrar seu grau de eficiência.

1.4 Estrutura do Documento

Esta pesquisa está estruturada da seguinte maneira:

- **Capítulo 2:** Este capítulo apresenta uma breve revisão dos trabalhos diretamente relacionados com o escopo deste trabalho;

- **Capítulo 3:** Apresenta conceitos e explanações sobre protocolo HTTP e sistemas de *proxy web cache* para o entendimento da pesquisa;

- **Capítulo 4:** O funcionamento e composição de URLs, bem como o funcionamento do *YouTube* e trabalhos que abordam reescrita de URLs são detalhados neste capítulo.

- **Capítulo 5:** Apresenta uma análise preliminar de um sistema de *proxy web cache* com configuração padrão, em ambiente de produção de uma rede institucional, detalhando a taxa de *hits* e *miss* obtidas, bem como os principais serviços acessados;

- **Capítulo 6:** Neste capítulo são levantadas as propostas de implementação de um modelo de *proxy web cache* institucional focado em URLs dinâmicas, na literatura e em projetos de código aberto, sendo escolhida a de melhor aderência. A proposta de melhor aderência é explanada com uma implementação básica;

- **Capítulo 7:** Descreve os cenários e os testes de verificação de eficiência, bem como a análise dos resultados obtidos;

- **Capítulo 8:** Este capítulo conclui este trabalho relatando a eficiência da proposta de melhor aderência em face ao contexto de sua utilização e são sugeridos tópicos de interesse para trabalhos futuros.

Capítulo 2

Estado da Arte

Tendo em vista os objetivos deste trabalho, realizou-se uma ampla pesquisa sobre *proxy web cache* institucional com ênfase em URLs dinâmicas nos últimos 10 (dez) anos de 2004 à 2014, com objetivo de identificar trabalhos correlacionados. Esta pesquisa foi realizada em diversos períodos disponibilizaos pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) através da plataforma "Portal de Periódicos da CAPES" [6]. Identificando-se 2 (dois) grupos de trabalhos correlacionados, porém não utilizáveis nesta pesquisa, e os trabalhos correlatos que de fato contribuíram com a pesquisa.

Uma parte considerável das pesquisas estão em torno da área de *proxy web cache* reverso, sendo constituída pelos trabalhos: [7], [8], [9], [10], [11] que buscaram melhorar o acerto na taxa de *hits* e a diminuição do tempo de latência dos objetos disponibilizados pelos servidores *web* de aplicação, aliviando assim sua carga. Para atingir este objetivo foi utilizado um *proxy web cache* reverso com técnicas variáveis permitindo-lhe tratar as URLs e objetos dinâmicos. Apesar de, correlacionados estes trabalhos não corroboraram com esta pesquisa por terem focado em servidores *web*.

Outra parte considerável das pesquisa estão na área de *delta enconding*, sendo constituída dos trabalhos [12], [13], [14] e [12] que utilizaram-se desta técnica, conforme detalhada a RFC 3229 (do Inglês, *Request For Comments*) [15], na qual apenas parte do conteúdo dinâmico modificado seria solicitada, sendo analisadas diversas partes internas

dos objetos. Neste contexto é sugerido um *proxy web cache* na borda das redes institucionais com capacidade para interagir com as aplicações de serviços da Internet. A técnica de *delta enconding* ainda não está disseminada pois, necessita de mudanças na concepção da programação de páginas e aplicações da *web*, não sendo utilizada nesta pesquisa. Os trabalhos correlatos que corroboram com esta pesquisa foram: [16], [17] e [18]. Em [16] os autores analisaram o impacto da mudança da natureza das URLs estáticas para URLs dinâmicas nos servidores de *proxy web cache*. Foram analisadas 3 (três) redes, sendo 2 (duas) acadêmicas, nas redes acadêmicos identificou-se que 60% à 78% das requisições referiam-se a URLs dinâmicas e que as taxas de acerto de *hits* por requisição e por volume foram de 24,3% e 22,8%, respectivamente. Ao final é inferido que as URLs dinâmicas tem percentual significativo, porém são consideráveis não cacheáveis, sendo necessário o tratamento das mesmas para aumento da taxa de acerto de *hits*. Em [17] analisa um sistema de *proxy web cache* convencional com foco nos objetos e URLs do *YouTube*, detalhando assim as características dos vídeos solicitados. Da análise foram levantados os requisitos necessários para um *proxy web cache* especializado neste serviço, tendo como inferência que a natureza dinâmica destas URLs ocorre em função da fragmentação do vídeo em objetos, sendo necessária trata-las. Em [18] os autores discorrem sobre a problemática da natureza das URLs dinâmicas ofuscadas no processo de aquisição de dados para um sistema de base de conhecimento, buscando desofuscá-las e sumarizar as URLs equivalentes. Desta forma parte do trabalho pode ser aplicado a esta pesquisa na problemática das URLs dinâmicas, haja vista que através de análise de parâmetros da composição destas URLs os autores identificaram os parâmetros essenciais para decodificação e sumarização das URLs equivalentes. Porém, não será possível aplicar a proposta em sua plenitude, pois a técnica utilizada é baseada no diferencial entre varreduras de páginas *web* e utiliza-se para decodificação parâmetros presentes nos cabeçalhos HTTP tais com `cookies`.

Como resultado do levantamento do estado da arte não foram encontrados trabalhos que propusesse uma avaliação e implementação de um sistema de *proxy web cache* institucional com foco em URLs dinâmicas, de forma a melhora a taxa de *hits* dos objetos

armazenados, sendo este o tema proposto para esta pesquisa.

Capítulo 3

Fundamentação Teórica

Este capítulo apresenta os conceitos básicos necessários para o entendimento de um sistema de *proxy web cache* dinâmico. Com este intuito, serão apresentados, de forma contextualizada, a pilha de protocolo TCP/IP (do Inglês, *Transmission Control Protocol/Internet Protocol*) e o protocolo HTTP.

3.1 Pilha de Protocolos TCP/IP

Segundo [4], o modelo de referência TCP/IP foi criado para suprir a deficiência de conectar várias redes de maneira uniforme, sendo posteriormente aprimorado para ser robusto a ponto de funcionar quando alguns *hosts* ou ativos de redes falhassem. Foi inicialmente implementado na ARPANET (Rede da Agência de Pesquisa de Recursos Avançadas) do Departamento de Defesa Americano e atualmente na Internet.

Sua flexibilidade permite adaptar-se a uma grande gama de aplicações, desde envio de *e-mail* a sistemas de transmissão de voz e vídeo em tempo real. Possui 4 (quatro) camadas independentes conforme pode ser observado na Figura 3.1: (*i*) *host*-rede; (*ii*) inter-redes; (*iii*) transporte; e (*iv*) aplicação. Abaixo será feita uma breve descrição de cada uma das camadas.

Camadas	Suíte TCP/IP
Aplicação	**HTTP**, FTP, SMTP, DNS, ...
Transporte	**TCP** e UDP
Inter-Rede	**IP**
Hosts/Rede	Ethernet, Frame Relay, ...

Figura 3.1: Protocolo TCP/IP.

- ***Host*-Rede**: camada mais próxima do *hardware* responsável pelo acesso lógico e físico ao meio.

- **Inter-Rede**: camada que integra toda a arquitetura TCP/IP, tem como função interligar redes sem conexões, permitindo assim que pacotes gerados pelos *hosts* trafeguem por redes heterogêneas de forma independente e cheguem ao seu destino. O Protocolo presente nesta camada é o IP (Protocolo de Internet) [19] que tal como um sistema de endereçamento postal atribui endereços para redes e *hosts*.

- **Transporte**: camada localizada acima da camada de Inter-Redes responsável pela conversação entre *hosts* de origem e destino. Seus principais protocolos são TCP (Protocolo de Controle de Transmissão) [20] e o UDP (Protocolo de Datagrama ao Usuário) [21], sendo apenas o primeiro orientado a conexão. O protocolo TCP é confiável ao passo que verifica erros nos pacotes, tais como pacotes corrompidos ou faltantes, solicitando novo reenvio. Além de possuir o controle de fluxo de forma a dosar e equilibrar o recebimento e envio de dados entre 2 (dois) *hosts*. O UDP não é confiável e nem possui controle de fluxo repassando estas responsabilidades para a aplicação contudo, é um protocolo enxuto e bastante simples, sendo bastante usado em aplicações de tempo real, tal como voz e vídeo.

- **Aplicação**: camada localizada no topo do modelo TCP/IP logo acima da camada de Transporte, sendo a camada de mais alto nível, conectando-se diretamente as aplicações. Possui uma gama variada de protocolos tais como: SMTP (Protocolo de

Transporte de *Mail* Simples) [22], FTP (Protocolo de Transporte de Arquivos) [23], DNS (Serviço de Resolução de Nomes) [24], HTTP, entre outros.

No contexto deste trabalho utilizou-se os protocolos IP, TCP e HTTP correlacionado com suas respectivas camadas na Figura 3.1. Sendo os protocolos IP e TCP empregados para redirecionamento do tráfego *web* e o protocolo HTTP para *cache* de conteúdo *web*. Detalhou-se como foram empregados os protocolos no Capítulo 5.

3.2 Sistemas de *Cache Proxy Web*

Um sistema de *proxy web cache* tem como função o armazenamento dos objetos *web* mais acessados pelos usuários de forma a economizar recursos computacionais. Um *proxy web cache* bem configurado pode trazer os seguintes benefícios:

- economia do consumo de banda externa;
- diminuição da carga de processamento de servidores;
- redução do tempo médio de resposta das requisições *web*.

Um servidor de *proxy web cache* pode ser empregado de três formas distintas, sendo elas:(i) encaminhamento, tal como ilustrado na Figura 3.2; (ii) transparente, tal como ilustrado na Figura 3.4; e (iii) reverso, tal como ilustrado na Figura 3.3. Nos 2 (dois) primeiros modos o *proxy* é instalado na rede local de forma a intermediar as requisições web dos usuários para os diversos servidores localizados na Internet. No terceiro modo o *proxy* faz intermediação das requisições feitas pelos usuários da Internet ao servidor localizado na rede local.

A diferença básica entre os modos de encaminhamento e transparente reside na necessidade ou não de configurar os *hosts* dos usuários. No modo de encaminhamento é necessário configurar todas as máquinas da rede local para acessarem a Internet através do *proxy*, enquanto no modo transparente não é necessária nenhuma ação perante ao

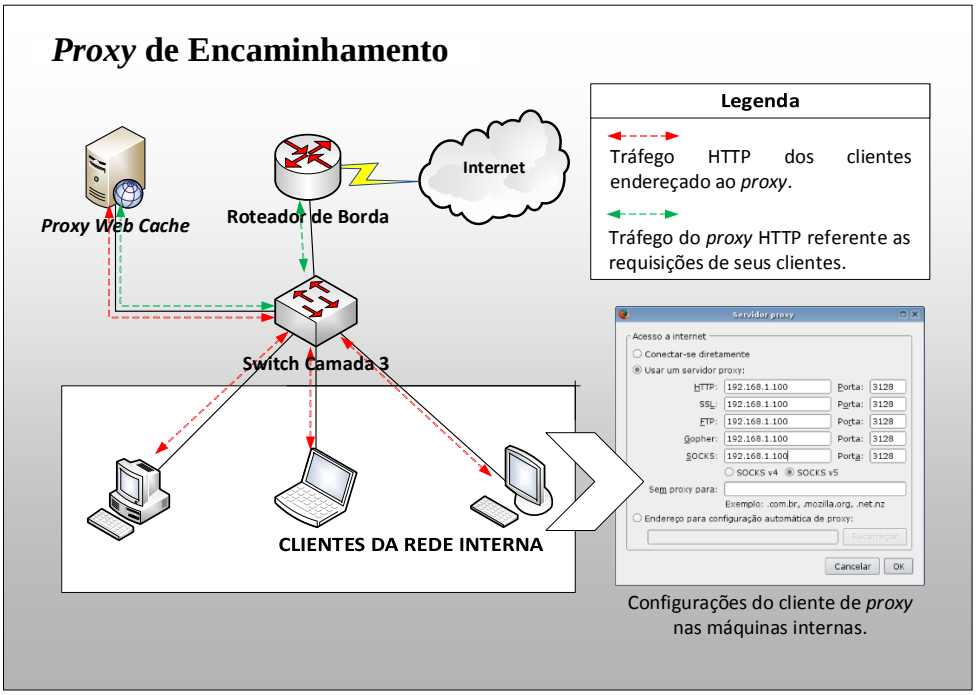


Figura 3.2: Topologia de *proxy* de encaminhamento.

usuário. Entretanto, o modo transparente requer configurações especiais em alguns ativos de rede de forma a interceptar o tráfego *web* e redirecioná-lo para o *proxy*.

A interceptação e o redirecionamento podem ser executados através de diversas técnicas, sendo as mais usadas: roteamento por política, NAT/PAT (do Inglês, *Port Address Translation*)/(do Inglês, *Port Address Translation*) [4] e WCCP (do Inglês, *Web Cache Communication Protocol*) [25]. O WCCP é um protocolo proprietário da *Cisco Systems, Inc.* [26], sendo suportado apenas pelos equipamentos deste fabricante.

As três técnicas trabalham com análise dos pacotes trafegados de forma a interceptar e redirecionar, apenas a que possuem como destino as portas 80, 8080, 443. Após a interceptação, os pacotes sofrem uma alteração do roteamento padrão para um alternativo redirecionando-os para o *proxy*. A diferença básica entre as três técnicas é onde ocorrerá a interceptação dos pacotes.

Na interceptação através de roteamento por política são utilizados equipamentos como roteadores e *firewalls*. No WCCP a interceptação é feita em roteadores, *swicths* de camada 3 e em *firewalls* da *Cisco Systems, Inc.* com suporte a este protocolo. Para o emprego da

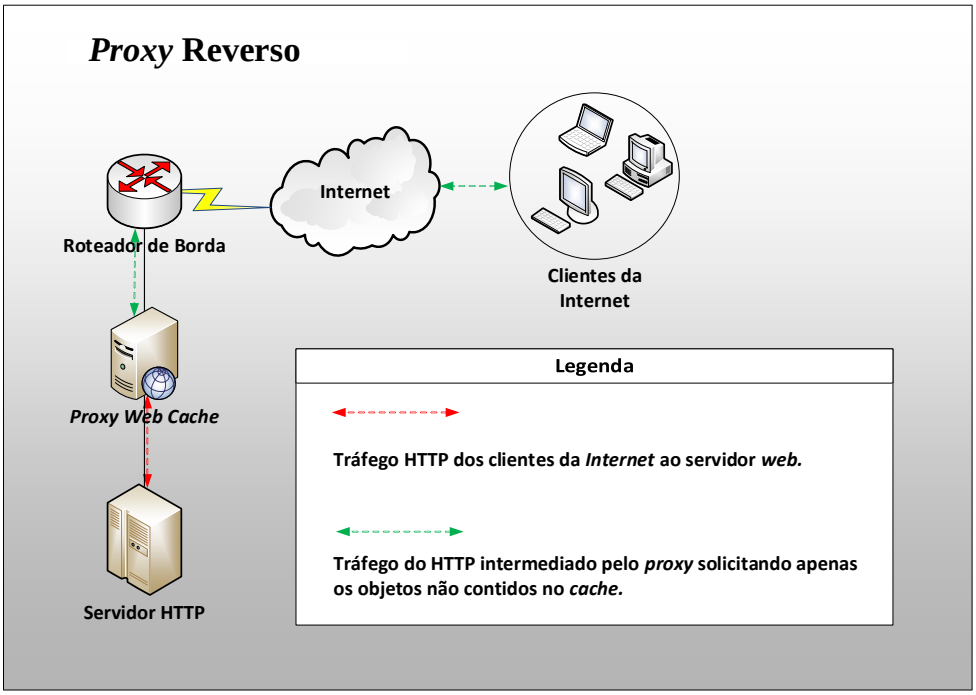


Figura 3.3: Topologia de *proxy* reverso.

técnica de NAT/PAT é necessário que o *proxy* tenha função de roteador.

Abaixo são descritas ferramentas de *proxy web cache* de código aberto de grande popularidade [4]. São elas:

- ***Apache* HTTP *Server*:** nativamente é um servidor de HTTP, porém, por ser bastante flexível possuí um módulo denominado de *mod_proxy* que permite funcionar como um *proxy* de encaminhamento. Possui também uma série de módulos e *plugins* permitindo trabalhar como balanceador de carga e *proxy* reverso. Ainda hoje é o servidor de HTTP mais usado no mundo [27] e [28]. É desenvolvido e mantido pela fundação *Apache*.

- ***Nginx*:** tal como *Apache*, foi construído inicialmente para ser um servidor *web*, porém, possui um módulo denominado de *ngx_http_proxy* que permite funcionar como um *proxy* de encaminhamento. Assim como *Apache*, possuí diversos módulos e *plugins*. Atualmente é considerado o segundo melhor servidor *web* do mundo [27] e [28].

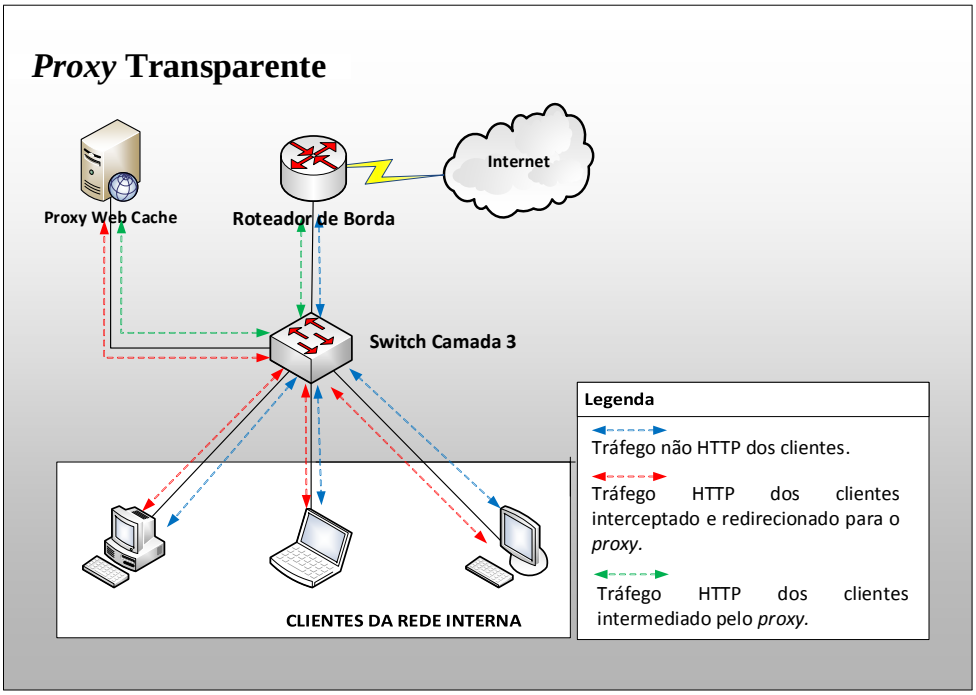


Figura 3.4: Topologia de *proxy* transparente.

- ***Squid*:** é considerado o 1º *proxy web* da história da Internet, sendo considerado até hoje um dos mais populares [4]. Dispõem através do site do projeto [2] e de fóruns uma vasta documentação. Diferencia-se basicamente dos demais por ser um servidor de *proxy* nativo.

- ***Traffic Server*:** trata-se de um projeto doado pelo *Yahoo* a fundação *Apache*, sendo mantido e desenvolvida pela mesma. Assim como o *Apache* e como o *Ngnix*, possui diversos módulos, *plugins* e funcionalidades.

Neste trabalho será empregado a ferramenta de *proxy web cache Squid* em modo transparente com interceptação e redirecionamento de pacotes baseado em roteamento por política, tal como será viso na Subseção 5.3. Utilizou-se a ferramenta *Squid* devido sua história, por ser na atualidade umas das mais eficazes e populares ferramentas do *proxy web cache* e por dispor de uma ampla documentação.

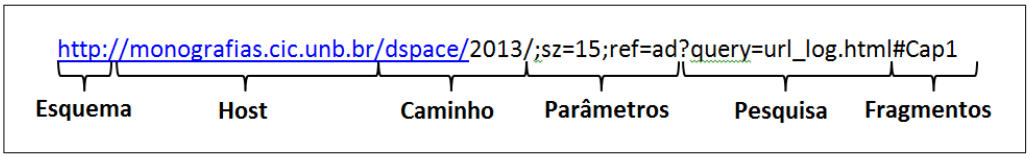


Figura 3.5: Sintaxe aplicada as URLs.

3.2.1 Protocolo HTTP

O HTTP é um protocolo da camada de aplicação da pilha do TCP/IP, baseado na arquitetura cliente servidor e em requisições e respostas. Foi definido por Tim Berners-Lee na década de 1990 motivado pela necessidade de disseminar informações. Em 1991 Tim Berners-Lee desenvolveu a primeira página com suporte ao protocolo HTTP, dando origem a grande teia mundial *World Wide Web*. O HTTP foi formalizado pela primeira vez em 1992 como HTTP/0.9 [29], tendo evoluído para a versão HTTP/1.0 [29] em 1996 e tendo como última versão o HTTP/1.1 [3]. A cada nova versão são incluídas novas funcionalidades, tendo na versão 1.0 as primeiras noções de *caching*. Para fins de convenção será abordada neste trabalho a versão HTTP/1.1.

Definida pela RFC [30], A URI é descrita como uma sequência de caracteres com sintaxe restrita que identifica ou referencia recursos de forma homogênea de maneira fácil e ao mesmo tempo extensível. A URI pode ser classificadas como URL, URN (do Inglês, *Uniform Resource Names*) [4] ou ambas. A URN referencia o nome do recurso enquanto a URL referencia o método para encontrar ou acessar este recurso. Abaixo são apresentados exemplos de URN e URL.

- **URN:** `urn:isbn:000134597`.
- **URL:** `http://monografias.cic.unb.br/dspace/mestrado.pdf`.

A URL possui uma sintaxe própria dividida em nove parâmetros, sendo eles: `esquema`, `usuário`, `senha`, `host`, `porta`, `caminho`, parâmetros de `consulta` e `fragmentos`. Na Figura 3.5 são ilustrados os 6 (seis) parâmetros que serão utilizados nesta pesquisa. Abaixo são detalhados cada um destes parâmetros.

- `Esquema`: esquema com qual o protocolo deve usar e acessar o recurso, sendo os mais usados: `http`, `malito` e `ftp`.
- `Host`: indica em que máquina o recurso está alocado.
- `Caminho`: indica em que local na máquina o recurso está alocado, remetendo a uma estrutura hierárquica de diretórios.
- `Parâmetros`: permite que aplicações passem parâmetros extras, utiliza o caractere "`;`" para este fim.
- `Consultas`: permite que consultas sejam feitas diretamente pelas URLs, utiliza o caractere "`?`" para este fim.
- `Fragmentos`: indica qual a parte do recurso deve ser solicita, utiliza o caractere "`#`" para este fim.

3.2.2 Funcionamento do HTTP

O protocolo HTTP funciona basicamente com requisições e respostas através de envio de massagens. Porém, antes é necessário que haja uma sessão TCP/IP estabelecida. A sessão TCP/IP é estabelecida a partir do IP do cliente para o IP do servidor, utilizando-se portas TCP altas no cliente e porta 80 no servidor.

O cliente incia a conexão com um pacote `syn`, logo depois o servidor envia um pacote com `syn`+`ack` e por fim o cliente devolve com um pacote `ack`. Após o triplo aperto de mãos a sessão esta estabelecida. A partir desta conexão o cliente utiliza um navegador para enviar e receber do servidor mensagens do protocolo HTTP.

Estas mensagens são padronizadas pela RFC [3], tendo a primeira linha denominada como linha de requisição, seguida linhas de cabeçalhos, sendo delimitados por uma linha em branco e terminado com o corpo da mensagem. A linha de requisição é formada pelo método, pelo caminho da URL e pela versão do protocolo.

Os métodos disponíveis são: `GET`, `HEAD`, `OPTIONS`, `POST`, `PUT`, `DELETE` e `TRACE`. Os métodos mais usados são: `GET`, `POST` e `HEAD`. O `GET` solicita um recurso, enquanto que o `POST` envia dados. O `HEAD`, assim como o `GET`, faz solicitações, entretanto, de metadados sem a necessidade do envio do corpo da mensagem. Como será visto mais a frente, o `HEAD` é utilizado para troca de informações sobre *cache* de um recurso.

Nas linhas de cabeçalhos são inseridas informações que qualificam a requisição com informações sobre a codificação e linguagem suportadas pelo cliente, e os tipos de objetos aceitos para transferência, além de dados sobre *cache*, `cookies`, entre outros. Os `cookies` são informações trocadas entre o servidor e o cliente referentes à conexão lógica da aplicação *web*, permitindo o armazenamento local das referências dos clientes e a persistência de uma autenticação até sua expiração.

A Figura 3.6 ilustra uma requisição com as seguintes características: (*i*) o recurso está sendo solicitado através do método `GET`, o recurso está implícito, sendo a página `index.html`, tendo como caminho referenciado o diretório local do servidor; (*ii*) a máquina que possui o recurso é `cic.unb.br`; (*iii*) o agente do cliente é o *Mozilla*; (*iv*) a máquina do cliente possui o sistema operacional *Windows NT* 6.1; (*v*) os formatos de objetos suportados são texto, HTML, XML e XHTML; (*vi*) as línguas suportadas são Português do Brasil e Inglês Americano; (*vii*) é suportado a compactação do objeto; (*viii*) é informado o `cookie` que o cliente utiliza para acessar este recurso; (*ix*) o estado da conexão; e (*x*) por fim, o navegador do cliente pergunta se há alguma alteração desde a última solicitação do recurso.

A resposta HTTP tem um formato bem parecido com a resposta, tendo como diferença primordial a primeira linha, sendo colocado o estado da resposta da requisição. O estado da resposta é um código numérico de 3 (três) números, sendo separado em 4 (quatro) classes tal como demonstrado abaixo:

- `1xx`: esta classe de codificação indica que a solicitação foi recebida e está sendo processada, representando uma reposta provisória ou transitória.

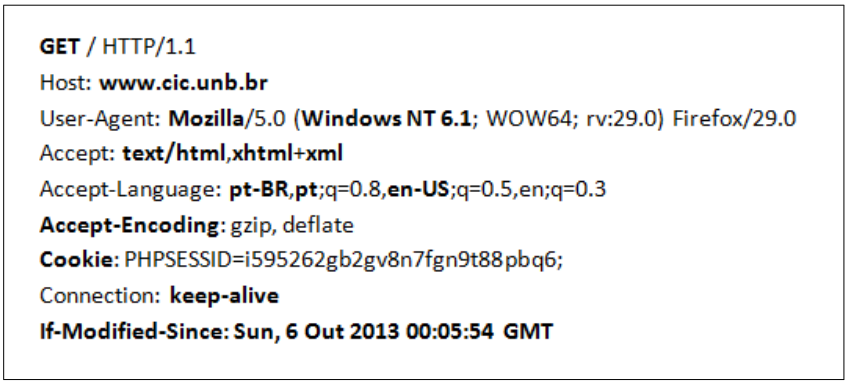


Figura 3.6: Cabeçalho de requisição HTTP.

- `2xx`: esta classe de codificação indica que a solicitação foi recebida e respondida com sucesso.

- `3xx`: esta classe de codificação indica que o cliente terá que efetuar outras ações para concluir a requisição, tal como ser redirecionado para outra página.

- `4xx`: esta classe de codificação indica que o cliente provavelmente solicitou erroneamente um recurso. O código `404` representa que houve comunicação porém, não foi encontra o recurso solicitado.

A Figura 3.7 apresenta uma mensagem de resposta HTTP, tendo na primeira linha o código de estado `200` e a versão do protocolo HTTP e nas demais linhas dos cabeçalhos as seguintes informações: tempo e hora do servidor durante o envio da mensagem; o *software* que está rodando como servidor *web*, bem como seus *plugins* ou módulos; informações do estado da conectividade e demais informações sobre o *cache* de objetos que serão vista posteriormente.

O HTTP possui embarcado um mecanismo de *cache* através de alguns cabeçalhos, sendo eles: `expires`, `cache-control`, `last-modified` e `if-modified-since`. Pode existir numa mesma mensagem informações conflitantes destes cabeçalhos, sendo assim existe uma precedência, sendo a menor precedência do `expires` e a maior do `if-modified-since`. Abaixo há um detalhamento de cada um destes cabeçalhos:

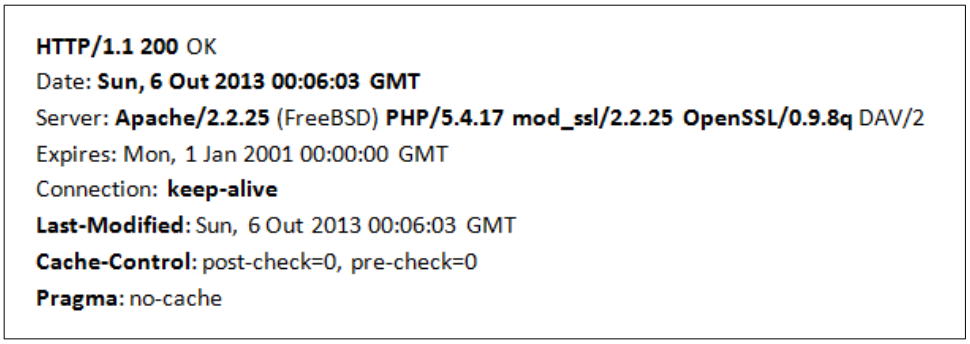


Figura 3.7: Pacote HTTP de resposta.

- `Expires:` fornece a data de expiração de um objeto, desta forma até a data informada o objeto não será alterado podendo ficar armazenado no *cache*, após esta data deverá ser feita nova requisição. Entretanto, quanto o servidor de origem não deseja que o objeto não seja armazenado em *cache* partir desta técnica, informa a data de expiração anterior a data atual da requisição, tal como na Figura 3.7;

- `Cache-Control:` é um dos cabeçalhos mais completos possuindo diretivas próprias tais como: `no-cahe`; `no-store`; `private`; `public`; `max-age`; `s-maxage` `must-revalidade`; `proxy-revalidation` e `no-transform`, sendo as cinco primeiras mais utilizadas. Abaixo detalha-se as diretivas mais utilizadas:

  - `No-Cahe`: o objeto não deve ser armazenado em *cache* antes que se verifique se o mesmo está obsoleto;
  - `No-Store`: o objeto não deve ser armazenado em nenhum tipo de *cache*, ou seja, nem no cliente nem no *proxy* público;
  - `Private`: o objeto deve ser apenas armazenado no *cache* do navegador do cliente;
  - `Public`: o objeto pode ser armazenado tanto no *cache* público quanto no *cache* do navegador do cliente;
  - `Max-age`: define o tempo relativo em segundos de expiração do objeto, desde a última requisição do cliente. Depois de expirado este tempo o objeto deverá ser retirado do *cache* e ser executada uma nova solicitação.

- `HeaderLast-Modified`: através deste cabeçalho o servidor informa a última data de modificação do objeto. Desta forma o cliente ou *proxy* pode comparar as datas e tempos do objeto armazenado em *cache* com o objeto disponível no servidor. Na Figura 3.7 o servidor informa que a última modificação do objeto solicitado foi em 6 outubro de 2013 às 00:06:03 GMT.

- `HeaderIf-Modified-Since`: com este cabeçalho o cliente ou *proxy* questiona ao servidor se o objeto armazenado localmente foi alterado desde a última requisição. Na Figura 3.6 o cliente questiona ao servidor se houve alguma modificação do objeto depois do dia 6 de outubro de 2013 às 00:06:03 GMT.

Os objetos somente serão armazenados em *cache* se alguns dos cabeçalhos acima apresentem informações ou condições para que isto ocorra, caso o contrário os objetos não serão armazenados em *cache* por um sistema de *proxy web cache*. Por fim, existem casos especiais que dificilmente o objeto será armazenado, salvo por configuração explícita, sendo eles: URLs dinâmicas; utilização dos métodos diferentes de `GET`, `HEAD` e `POST`; autenticação no cabeçalho; cabeçalho de `cookie` ou `set-cookie` na resposta.

Geralmente, nas URLs dinâmicas são identificadas por conterem os seguintes parâmetros:

- **?**: parâmetro relacionado com pesquisas.

- #: parâmetro relacionado com conteúdo dinâmico.

- `cgi-bin`: parâmetro relacionado com conteúdo dinâmico.

3.2.3 Políticas de Substituição de Objetos em *Cache*

A política de substituição de um objeto do *cache* é acionada toca vez que o limiar do espaço de armazenamento do *cache* atinge seu limite, sendo necessária a exclusão de objetos em *cache* para que haja condições de inserção de novos. As diversas políticas de substituições existentes podem se basear na popularidade dos objetos em relação a quantidade de

requisições, em tamanho e em ambos os casos. Neste trabalho serão usadas as políticas de substituições LRU (*Least Recently Used*); LFU (*Least Frequently Used*), GDSF (*Greedy-Dual Size Frequency*) e LFDA (*Least Frequently Used with Dynamic Aging*). Abaixo detalha-se cada uma destas políticas:

- **LRU**: trabalha com o conceito temporal, no qual os objetos mais recentemente utilizados serão mantidos no *cache* em virtude de sua maior chance em serem utilizados em um curto espaço de tempo [31].

- **LFU**: trabalha com um sistema de contador e incremento, no qual toda vez que um objeto é utilizado é incrementado um valor em seu contador, sendo excluído aquele com menor valor em seu contador [32].

- **GDSF**: busca otimizar a taxa de acerto, mantendo apenas os objetos menores mais populares, excluindo os maiores [33].

- **LFDA**: mantém em cache apenas os objetos mais populares, independentemente do tamanho [33].

Capítulo 4

YouTube

Este capítulo descreve o serviço de compartilhamento de vídeos pela Internet, *YouTube*, detalhando seu funcionamento, a composição de suas URLs, como suas informações são recuperadas, entre outras características. Na Seção 4.1 é fornecido um breve histórico do serviço, algumas estatísticas relevantes, trabalhos acadêmicos sobre o tema e informações técnicas sobre tipos de exibições dos vídeos. Na Seção 4.2 é feita uma breve descrição do funcionamento deste serviço que atende bilhões de usuários. A Seção 4.3 apresenta o formato de requisições dos objetos dos quais os vídeos são compostos.

Optou-se por reservar um capítulo exclusivo para o *YouTube* em função de sua popularidade, representatividade de tráfego na Internet, e pela complexidade em armazenar seus objetos em face da composição de suas URLs. Sendo assim, a análise do tráfego do *YouTube* torna-se importante a qualquer estudo de *proxy web cache*, que se propõe a tratar de URLs com foco na melhoria da taxa de *hits* e latência na recuperação dos objetos.

4.1 Visão Geral

O *YouTube*, serviço de compartilhamento de vídeos gratuito na Internet, teve uma grande aceitação por partes dos usuários desde seu início em 2006 [34]. Neste mesmo ano foi

considerado o 15º site mais acessado no mundo, sendo posteriormente vendido para a *Google Inc.* [34].

Atualmente o portal do *YouTube* é o 3º site mais acessado do mundo e o 4º mais acessado do Brasil, segundo a *Alexa* [35]. *Alexa* é um serviço de análise e mensuração de sites da Internet com foco em negócios. Estatísticas oficiais do *YouTube* [36] relatam que o serviço é acessado por mais de um bilhão de usuários únicos todos os meses, tendo em média cada pessoa do planeta assistindo quase uma hora de seus vídeos por mês. Por ser um serviço tão expressivo e inovador, acabou por chamar a atenção da comunidade acadêmica, desenvolvendo-se assim diversos trabalhos sobre o tema.

Os trabalhos levantados, com relevância a esta pesquisa, têm em comum a caracterização de aspectos do tráfego do *YouTube*. Contudo, abordam focos e aspectos diferentes do serviço do vídeo do *YouTube*. Abaixo são listados os trabalhos e as informações mais relevantes para esta pesquisa.

- **Trabalhos [34], [37], [38] e [17] relatam que:** (*i*) grande parte dos vídeos solicitados são abortados precocemente; (*ii*) a qualidade de vídeo mais popular foi 360p com container de *Flash*, com mais de 85% das exibições; e (*iii*) a qualidade padrão de alta definição encontrada foi de 720p em todos os casos.

- **Trabalhos [34] e [37] relatam que:** (*i*) 60% à 80% dos vídeos tiveram apenas 20% de sua duração exibida antes de serem abortadas pelos usuários; e (*ii*) a grande maioria dos vídeos não excederam 3 (três) minutos de exibição e uma parcela mínima ultrapassou os 10 (dez) minutos.

- **Trabalho [34] relata que:** a (*i*) os usuários raramente, menos de 5%, alteram a qualidade dos vídeos e a visualização para tela cheia; (*ii*) que 25% a 39% do tráfego de computadores pessoais são desperdiçados em virtude da política agressiva de *bufering* do *YouTube*; e (*iii*) apenas 10% dos vídeos tiveram 50% de sua duração exibida.

- **Trabalho [17] relata que:** (*i*) apenas 20% dos vídeos correspondem a 75% do total de tráfego do *YouTube*; (*ii*) 72% dos vídeos foram solicitados do início; (*iii*) um sistema de *proxy web cache* para *YouTube* para ser eficiente deve tratar os pedações dos vídeos, ou seja, e não somente o vídeo de forma integral; e (*iv*) 50% da latência de `RTT` calculada entre as requisições de vídeos e o primeiro pedaço do mesmo foi menor que 50 ms, tendo como valores de intervalos entre 3 e 300 ms.

- **Trabalho [39]:** (*i*) descreve o funcionamento do *YouTube*; (*ii*) efetuando caracterização do tráfego; e (*iii*) descrição de parâmetros de composição de suas URLs.

- **Trabalho [5]:** descreve o funcionamento da rede de distribuição de conteúdo do *YouTube*.

O *YouTube* trabalha com uma série de formatos de vídeos e áudio com diferentes resoluções e codificações, permitindo assim que o usuário doméstico envie cerca de 6 (seis) diferentes formatos com 5 (cinco) diferentes codificações. Os 6 (seis) formatos permitidos são [40]: Mov [41], MPEG4 (do Inglês, *Moving Picture Experts Group version* 4) [42], AVI (do Inglês, *Audio Video Interleave*) [42], WMV (do Inglês, *Windows Media Video*) [42], MPEGPS (do Inglês, *MPEG program stream*) [43], FLV (do Inglês, *Flash Video*) [43], 3GPP (do Inglês, *3rd Generation Partnership Project*) [44] e WebM (do Inglês, Web Multimedia) [45].

Na Tabela 4.1 são sumarizadas as codificações permitidas para submissão do vídeo por parte do usuário doméstico. Sendo descritos os parâmetros `tipos`, `resoluções`, `taxas de` *`bits`* `dos vídeos` e as `taxas de` *`bits`* `dos áudios em mono`, `estéreo` e `5.1`. O parâmetro `tipos` representa a categoria de resolução de vídeo. O parâmetro `resoluções` representa a resolução do vídeo. Os parâmetros `taxas de` *`bits`* `de vídeo` e `áudio` em geral representam a velocidade de transferência ou de processamento em *bits* por segundos necessários na codificação.

O *YouTube* oferece aos desenvolvedores *web* duas técnicas diferentes para embarque dos vídeos e customização do *player* de exibição, para serem utilizadas em diversas linguagens

Tabela 4.1: Tabela de codificação de vídeos usuários comuns do *YouTube*.

Tipo	**Resolução**	**Tx. *bits* Vídeo**	**Tx. *bits* Áud.Mono**	**Tx. *bits* Áud. Est.**	**Tx. *bits* Áud. 5.1**
1080p	1920 x 1080	8.000 *kbps*	128 *kbps*	384 *kbps*	512 *kbps*
720p	1280 x 720	5.000 *kbps*	128 *kbps*	384 *kbps*	512 *kbps*
480p	854 x 480	2.500 *kbps*	64 *kbps*	128 *kbps*	196 *kbps*
360p	640 x 360	1.000 kbps	64 *kbps*	128 *kbps*	196 *kbps*
240p	426 x 240	—	—	—	—

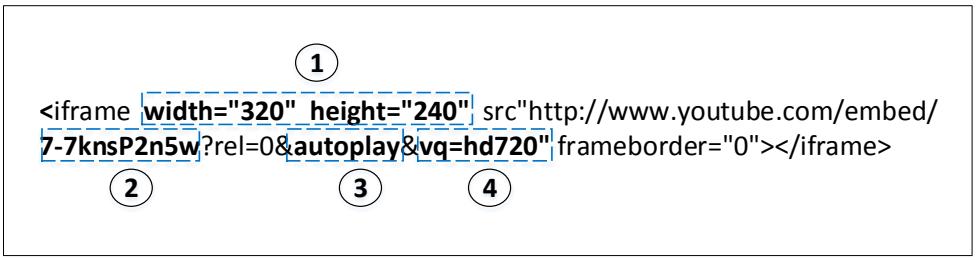


Figura 4.1: *Iframe* em HTML para vídeos do *YouTube*.

de programação, tais como: HTML, HTML5, *Java Script*, entre outros [46]. As duas técnicas disponibilizadas são: a "incorporação por objeto" e a "incorporação por *iframe*", sendo esta última mais nova e recomendada.

A Figura 4.1 ilustra a técnica de incorporação por *iframe* em HTML, com customização de 4 (quatro) parâmetros do *player* de exibição [47], sendo eles:

- `1`: refere-se ao tamanho da janela de exibição do *player*.
- `2`: refere-se ao `videoid`, identificador único do vídeo.
- `3`: refere-se exibição automática do vídeo ao carregá-lo.
- `4`: refere-se a qualidade sugerida para exibição do vídeo.

O parâmetro de `resolução de vídeo` permite ao desenvolvedor configurar previamente a qualidade do vídeo que deverá ser exibida ao usuário ao iniciá-lo. Caso os

controles do *player* estejam habilitados, o usuário poderá alterar a qualidade do vídeo após sua inicialização.

De acordo com o guia de desenvolvedor de referência para *API* (do Inglês, *Application Programming Interface*) do *YouTube* [46] o parâmetro de resolução `vq` pode receber os valores: `small`, `medium`, `large`, `HD720`, `HD1080`, `HD1440`, `default` e `highres` de forma individualizada. Os valores `small`, `medium` e `large` referem-se às resoluções de `240p`, `360p` e `480p`, respectivamente. O valor `default` refere-se à resolução de exibição definida automaticamente, tendo como valores `240p` ou `360p`, variando de acordo com qualidade de banda utilizada [48]. O parâmetro de `highres` refere-se a resolução máxima disponível.

Os usuários também podem acessar os vídeos através de pesquisas na Internet ou digitando diretamente a URLs dos vídeos no navegador. Em computadores pessoais o acesso é feito a partir dos navegadores, enquanto em dispositivos móveis ou eletrônicos, tais como televisores *smarts*, utiliza-se aplicativos específicos para *YouTube*.

Como será visto no Capítulo 7 utilizou-se da técnica de *iframe* com alteração dos parâmetros de qualidade dos vídeos, alterando-se o parâmetro `vq` para execução dos testes de validação da eficiência da proposta de um sistema de *proxy web cache* para URLs dinâmicas de melhor aderência. Na próxima seção será detalhado o funcionamento do *YouTube* para o provimento do serviço de visualização de vídeos.

4.2 Funcionamento do YouTube

O objetivo desta seção é descrever como o *YouTube* atende todos os seus usuários espalhados ao redor do mundo mantendo a qualidade de seus serviços.

Para atender a demanda mundial de acessos e postagens de vídeos o *YouTube* faz uso da tecnologia de CDN. CDN é uma rede de servidores geograficamente distribuída de forma a balancear a carga de requisições e minimizar a latência na entrega dos serviços de conteúdo. CDNs ampliam a capacidade de entrega de um serviço ao replicar a informação em vários servidores distintos [5]. Através de sua CDN o *YouTube* disponibiliza várias

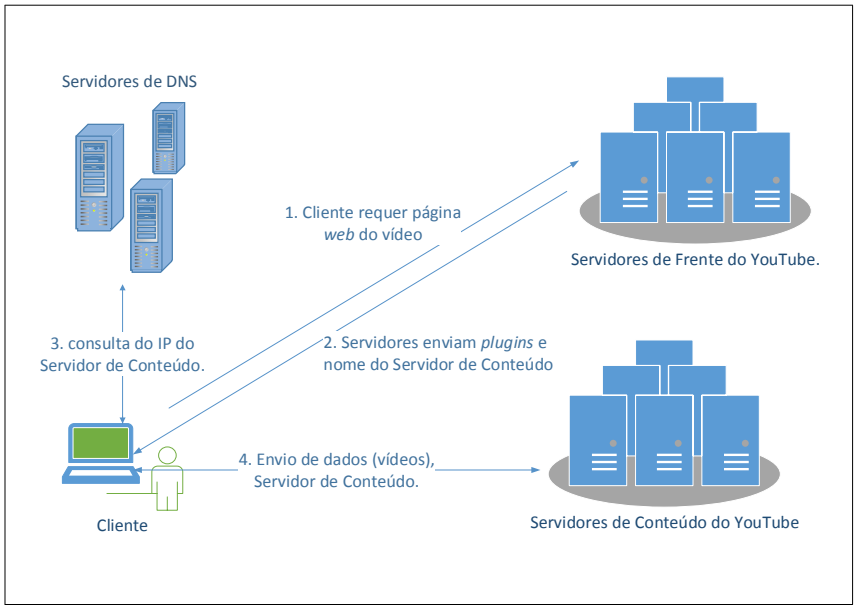


Figura 4.2: Passo a passo do serviço de entrega de vídeos - *YouTube*.

réplicas de seus vídeos e recebe os novos vídeos de seus usuários de forma distribuída [5]. Antes mesmo de acessar o site do *YouTube*, o serviço de resolução de nomes direciona o usuário para o servidor mais próximo geograficamente e de menor latência [17].

A Figura 4.2 ilustra uma solicitação de vídeo ao *YouTube* em quatro passos. No primeiro passo o usuário solicita uma URL da página de vídeo. No segundo passo o usuário recebe o conteúdo da página sem o vídeo, pequenas imagens, o *flash player* e ao solicitar o início do vídeo recebe os nomes dos servidores onde se encontram os objetos de vídeos propriamente ditos. No terceiro passo o usuário solicita o IP do servidor através do serviço de resolução de nomes, neste momento ele é direcionado aos servidores mais próximos e de menores latências. No quarto e último passo será solicitado ao servidor de conteúdo os objetos do vídeo e por fim, o usuário recebe os arquivos de vídeo e o assiste em seu *player*. A Figura 4.2 foi construída a partir de informações constante nos trabalhos [49] e [17].

Durante toda a comunicação entre o cliente e os servidores do *YouTube* são passados parâmetros do serviço pelas URLs, tornando-as dinâmicas e na sua grande maioria distintas, tal como será demonstrado na Seção 5.4. Isto ocorre devido *YouTube* utilizar da técnica de fragmentação de vídeos através de URI dinâmicas [50] para o *stream* de

seus vídeos. Desta forma, apesar dos vídeos serem considerados conteúdos estáticos, os sistemas de *proxy web cache* convencionais não conseguem efetuar *cache* deste conteúdo em função das URLs serem dinâmicas e distintas.

Na Seção 4.3 será feita uma análise mais detalhada das URLs do serviço do *YouTube* afim de analisar seus parâmetros objetivando identificar os mais relevantes.

4.3 Composição das URLs do *YouTube*

Nesta seção serão analisados os parâmetros mais relevantes na obtenção dos objetos de vídeo presentes nas URLs do *YouTube*. Apesar do *YouTube* não divulgar informações sobre os parâmetros de formação de suas URLs, utilizou-se informações dos trabalhos de [39], [50]. Em [39] foram analisados *logs* oriundos de comunicações reais entre clientes e servidores de conteúdo de mídia do *YouTube* com objetivo de identificar as características deste tráfego. Durante a análise os autores identificaram que as URLs de requisições de objetos de vídeos eram compostas pela expressão `videoplayback`. Tendo como sintaxe:

`http://<servidor-de- mídia>.youtube.com/videoplayback`

Os autores de [39] também identificaram nestas URLs, 7 (sete) diferentes parâmetros referentes a controle, *download* e exibição de vídeos. Na Tabela 4.2 são descritos todos os 7 (sete) parâmetros: `sparams`, `id`, `algorithm`, `fator`, `burst`, `begin` e `itag`. Os parâmetros destacados seguem sintaxe similar aos ilustrados no trabalho [50] que ilustra as estratégias para formação de URI dinâmicas de fragmentação de vídeos durante o *streaming*.

Ao observar os parâmetros apresentados na Tabela 4.2, identifica-se que apenas os parâmetros de `id`, `sparams` e `itag` são dinâmicos. O parâmetro `id` terá variabilidade para requisições de vídeos diferentes, enquanto `itag` e `sparams` poderão variar para requisições de um mesmo vídeo. Desta forma pode-se ter diversas versões de um mesmo vídeo para o mesmo `id`.

Apesar do trabalho [39] ter sido publicado em 2012, grande parte da coleta de dados foi executada em 2010, sendo constatado no Capítulo 6 que parte destes parâmetros estão

Tabela 4.2: Principais parâmetros de URLs para solicitação de objetos de vídeos.

Parâmetros	Descrição
`sparams`	Lista de parâmetros incluídos na requisição, separados por vírgulas.
`id`	Identificador único de vídeo.
`algorithm`	Algoritmo que o servidor de mídia pode utilizar para *stream* de vídeo. Sendo fixado como `throttle-factor`.
`factor`	Fator de velocidade, expressa o fator da taxa de codificação de um vídeo. Este valor é fixado em 1.25.
`burst`	A duração do vídeo que o servidor irá enviar para o *buffer* inicial medido em segundos. É fixado em 40 segundos.
`begin`	Tempo de início de reprodução expresso em milissegundos. Sendo fixado em 0 ms.
`itag`	Código de formato de vídeo, o equivalente a `fmt` (parâmetro de URL não documentado). *Clips* de vídeos baseadas em `flash` têm `itag` igual a `5` (de baixa qualidade, `240p`), `34` (qualidade normal, `360p`) e `35` (alto de qualidade, `480p`).

desatualizados. Estes parâmetros serão utilizados como base na Seção 6.2.1 para análise de proposta que tratem as URLs dinâmicas do *YouTube* de forma a permitir que os objetos referenciados por estas URLs tenham a possibilidade de armazenamento em *cache*.

O próximo capítulo apresenta uma análise preliminar de um *proxy web cache* convencional numa rede institucional, objetivando entender o comportamento e deficiências do *proxy* neste ambiente.

Capítulo 5

Caraterística do Tráfego HTTP da Rede da UnB

Este capítulo descreve uma análise preliminar das dificuldades enfrentadas por um sistema de *proxy web cache* institucional. Para a análise, foram coletados durante 4 (quatro) dias, dados do tráfego de borda da Rede de Comunicação da Universidade de Brasília (RedUnB), levantando e caracterizando a parte correlacionada ao protocolo HTTP e observando o comportamento do *proxy web cache* em sua configuração padrão.

O capítulo está dividido em 4 (quatro) seções: Rede de Comunicação da Universidade de Brasília (RedUnB), Levantamento do Tráfego HTTP, Pré-Análise e Discussão. Na Seção 5.1 é descrita a rede institucional utilizada durante a análise. A Seção 5.2 descreve o levantamento, a mensuração e posteriormente caracterização do tráfego HTTP presente na borda da RedUnB durante os 4 (quatro) dias de coleta. Na Seção 5.3 faz-se uma pré-análise dos dados obtidos. A Seção 5.4 apresenta uma discussão sobre os resultados obtidos e as dificuldades de implementação de um sistema de *proxy web cache*.

5.1 Rede de Comunicação da UnB

De forma a contextualizar um ambiente típico de uma rede institucional é feita uma breve descrição da Rede da Universidade de Brasília (RedUnB), apresentando sua topologia ló-

gica, física e o quantitativo estimado de usuários e ativos de redes. Todas estas informações foram embasadas em dados internos do Centro de Informática (CPD), unidade da UnB responsável pela área de Tecnologia de Informação e Comunicação (TIC).

A RedUnB foi projetada para interligar todos os centros, departamentos, faculdades, campus e áreas administrativas da UnB entre si e com a Internet de forma a suportar os mais variados serviços e ativos. Suportando desde serviços de simples como um envio de *e-mail* até serviços mais complexos como telefonia, teleconferência, computação em nuvem, entre outros, tudo numa mesma infraestrutura de rede.

A RedUnB possui uma rede classe B 164.41.0.0/16 válida, podendo endereçar até 65.536 (sessenta e cinco mil e quinhentos e trinta e seis) *hosts*, e um Número de Sistema Autônomo (ASN), 21506, tendo presença e anúncio próprio de sua rede na Internet. Na Tabela 5.1 é dada uma visão geral dos ativos de redes presentes na RedUnB, descrevendo o quantitativo e sua composição.

Os principais ativos e seus quantitativos estão listados na Tabela 5.1, na qual se pode destacar que atualmente a rede interconecta mais de 13.000 (treze mil) *hosts*. Este quantitativo de *hosts* é tão expressivo que quando comparado ao quantitativo de *hosts* levantados pela *CIA* (Agência Americana de Inteligência) [51] em 232 (duzentos e trinta e dois) países, contempla um número maior de *hosts* do que 102 (cento e dois) países individualmente listados.

Na Figura 5.1 é ilustrada a estrutura de agregação de tráfego de 4 (quatro) níveis da RedUnB, sendo eles: (*i*) acesso, (*ii*) distribuição, (*iii*) núcleo e, (*iv*) borda. Para um melhor entendimento destes níveis é dada uma abordagem de baixo para cima, no qual o tráfego flui das redes locais para a Internet. A explanação é iniciada com o nível de acesso que tem como função ingressar os *hosts* das redes locais na RedUnB através dos *swicths* e pontos de acessos das redes sem fio agregando e submetendo o tráfego gerado para o próximo nível. Na sequência tem-se o nível de distribuição que através de *swicths* com velocidades de 1 *Gbps* que interliga as redes locais agregando-as e repassando seu tráfego para o núcleo. No núcleo são utilizados 4 (quatro) roteadores conectados

Tabela 5.1: Tabela de estimativa de ativos da RedUnB.

Item	Ativos de Rede	Qtd.	Observações
1	*Hosts*	13000	Computadores, impressoras, servidores, dispositivos móveis entre outros.
2	*Swicths* de Acesso	400	Possui interfaces de 100 *Mps* e 1 *Gbps*.
3	*Swicths* para Telefonia	400	Difere-se dos demais *switchs* pelo fato ser alimentado por baterias ou energia DC de 48 *volts*.
4	*Acces Point* para redes sem fio	500	Conecta os dispositivos móveis à rede sem fio.
5	Controladoras para redes sem fio	10	Controla todos os *Access Points* da rede sem fio corporativa.
6	*Swicth* de Distribuição	130	Possui todas as suas interfaces a 1 *Gbps*.
7	Roteadores de Núcleo	5	Possui interfaces de 10 e 1 *Gbps*.
8	*Firewall*	40	Normalmente, é colocado no perímetro da redes locais e da própria RedUnB.
9	Roteadores	2	Interconecta a RedUnB a Internet.

entre si com enlaces de 10 *Gbps*, perfazendo um total de 40 *Gbps*, que interliga todas as sub-redes internas da RedUnB. Por fim, tem-se o nível de borda composto de um *cluster* de *firewalls*, cujo objetivo é descartar tráfego espúrio, e um *cluster* de roteadores conectados a Internet por meio da rede GigaCandanga por enlaces agregados de 1 *Gbps*. A GigaCandanga [52], é a rede metropolitana comunitária de educação e pesquisa do Distrito Federal cujo objetivo é propiciar o acesso a Internet de alta velocidade para as Instituições de Ensino Superior (IES) e instituições de pesquisa localizadas no Distrito Federal. Criada em 2007, possui toda sua infraestrutura baseada em fibra ótica, congregando atualmente 30 (trinta) instituições participantes [53], tendo a UnB sua maior demandatária com o consumo de mais de 70% da capacidade do enlace ofertado.

Esta seção descreveu a estrutura da RedUnB oferecendo assim uma visão da dimensão e da complexidade de uma rede institucional. Na Seção 5.2 será levantando e caracterizado o tráfego HTTP presente nesta rede e o comportamento de um sistema de *proxy web cache* institucional em sua configuração padrão.

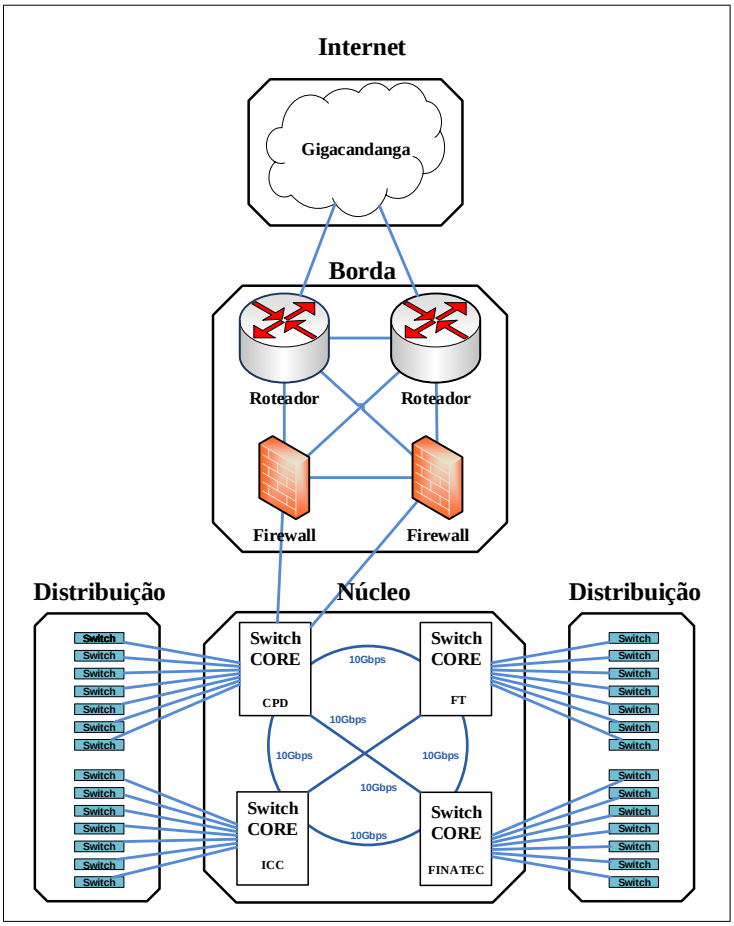


Figura 5.1: Topologia macro da RedUnB.

5.2 Levantamento do Tráfego HTTP da RedUnB

Esta seção tem como objetivo identificar e classificar os objetos mais demandados pelo uso do protocolo HTTP. Para tal, será realizado uma estimativa do volume de tráfego total da RedUnB, classificando-o percentualmente por protocolos da camada de aplicação. Após, será verificado o volume de objetos encapsulados pelo protocolo HTTP identificando por tipo de objeto. Como será apresentado nas seções subsequentes, identificou-se que do total de tráfego da RedeUnB, mais de 70% do tráfego corresponde ao protocolo HTTP, e deste, 70% corresponde a tráfego de URLs dinâmicas. Estes resultados demonstram que a utilização de um mecanismo que permita o reuso destes objetos via um sistema de *web cache* poderia reduzir significativamente a demanda por conexões externas uma vez que parte deste conteúdo estaria disponível no *cache* institucional.

5.2.1 Estimando o Volume do Tráfego HTTP na RedUnB

Visando alcançar o objetivo proposto neste trabalho, inicialmente foi realizada uma estimativa da banda total consumida e posteriormente identificação dos principais protocolos utilizados. Em especial, teve-se interesse em obter maiores informações sobre o volume de tráfego HTTP existente, comparando-o com os demais protocolos a fim verificar sua relevância.

Estas informações permitiram estabelecer parâmetros mínimos de rede para implementação de um sistema de *proxy web cache* em sua configuração padrão. Para este fim, foram utilizadas ferramentas de monitoramento de rede *Nagios* [54] e *Ntop* [55] que obtiveram informações de tráfego diretamente do *firewall* de borda da RedUnb conforme ilustrado na Figura 5.2. Ressalta-se que o *firewall* da UnB disponibilizou um conjunto de informações que foram acessíveis via SNMP (do Inglês, *Simple Network Management Protocol*) [56], tais como o volume, em *Mbps*, da banda externa consumida, condições gerais do equipamento como uso de processamento e memória.

Neste trabalho, utilizou-se o *Nagios* para obter o volume de tráfego a partir do *firewall* usando o protocolo SNMP. O *Ntop* foi utilizado para identificar o percentual de consumo de cada protocolo presente neste tráfego, utilizando-se a técnica de espelhamentos de portas, que será detalhada mais a frente. Por fim, foram correlacionadas as informações oriundas das duas ferramentas de forma a identificar o consumo médio e máximo, em *Mbps*, do tráfego HTTP. A seguir são descritas as ferramentas *Nagios* e *Ntop* com suas principais características:

- **Nagios** [57] e [54]: O *Nagios* teve seu lançamento em 1999 como projeto de código aberto baseado em sistema de monitoramento de infraestrutura de TI. Possui tanto o monitoramento proativo quanto reativo para qualquer tipo de ativo de rede, sistema operacional, aplicação, entre outros que disponha de recurso de SNMP ou que permita execução de seu cliente. Congrega a visualização, administração e gestão do monitoramento de toda a infraestrutura de TI em uma única interface *web*, além de possuir outras características tais como:

- **Alerta e mitigação de problemas**: permite configurar valores como limiares para checagens contínuas dos parâmetros de *hardware* e *software* dos ativos de rede de forma contínua identificando possíveis problemas de infraestrutura de TI. Fornece alerta aos técnicos responsáveis sempre que um limiar é atingido.
- **Flexibilidade**: através de seus *plugins*, *addons* e *front-ends* dispõem de diversos tipos de checagem com diversas forma de apresentações em diversos tipos de sistemas operacionais *windows, Unix, MacOSX, Android* entre outros. Possui mais de 50 (cinquenta) *plugins* para monitorar desde *hosts*, serviços até aplicações.
- **Relatórios**: dispõem de diversos relatórios que permite uma melhor gestão da infraestrutura de TI. Tem como destaque os relatórios de: disponibilidade, no qual descreve de forma individualizada o percentual no qual cada ativo ficou disponível; acordo de níveis de serviços, calcula o nível de serviço dos serviços prestado pelo agrupamento de ativos de rede e o de inventários, que relaciona todos os ativos de rede monitorados.

- **Ntop** [58] e [55]: é um projeto de código aberto desenvolvido com ajuda da comunidade de *software* livre, sendo iniciado em 1998. Entretanto, difere-se do *Nagios* ao focar-se no monitoramento de tráfego buscando identificar suas características tais como protocolos, aplicações e sistemas operacionais. Possui portabilidade para diversos sistemas operacionais, *Unix*, *MacOSX* e *Windows* 32 bits, além de utilizar a *libcap*, uma das mais populares bibliotecas para escuta de tráfego. Suporta diversos métodos para aquisição do tráfego, desde protocolos de amostragem [4] *sFlow*, NetFlow, IPFIX até escuta de interfaces em modo promíscuo. São ditos protocolos de amostragem por fornecerem parte das informações do pacote de dados ou no caso do sFlow fornecer pacotes completos em caráter de amostragem. Estes protocolos possuem uma arquitetura baseada em *probes*, coletores e analisadores. Os ativos de redes que possuem o suporte a estes protocolos enviam geralmente, via UDP, os

fluxos de dados selecionados ou padronizados para um dispositivo coletor. Os dispositivos coletores normalmente, são equipamento do tipo servidor que recebem e armazenam os fluxos. Posteriormente as máquinas analisadoras consultam os dados brutos presente nos coletores e geram informações úteis sobre a rede. No método de escuta deve-se colocar uma ou mais interface do ativo de rede em modo promíscuo de forma a receber todos os pacotes que cheguem à interface sem nenhum tipo de descarte, inclusive processando os pacotes que não são endereçados aos *hosts*. Para que uma interface seja colocada em modo promíscuo são utilizados *software* que manipulam os *drivers* e bibliotecas presentes na pilha TCP/IP dos sistemas operacionais, de forma a alterar o comportamento nativo da pilha. O *Ntop* é uma ferramenta bastante robusta, sendo capaz de analisar em tempo real tráfego com velocidades de até 10 *Gbps*. Para analisar tráfego com velocidade tão altas é requerido a utilização da plataforma *PF_RING DNA* para placas *Intel*. A tecnologia de *PF_RING DNA* permite o acesso direto da memória e registradores através do processador da placa de rede sem nenhum tipo de cópia. Abaixo são listadas as demais características do *Ntop*:

- **Analisa e Identifica os IPs mais Ativos na Rede**: identifica e correlaciona de forma ordenada por consumo de tráfego os IPs de origem e destino mais atuantes na rede de forma a levantar seus sistemas operacionais, sua geolocalização, portas de origens e destinos, suas aplicações de rede utilizadas e possíveis anomalias;
- **Identificação e Mensuração Protocolos e Aplicações**: identifica e ordena de forma percentual os principais protocolos e aplicações existentes na rede, inclusive identificando o protocolo HTTP não somente pela porta 80.
- **Visibilidade**: Fornece interface *web* dinâmica e moderna construída com HTML5 e AJAX. Permite numa única interface a visualização de estatísticas e a administração da ferramenta de forma integrada.

Tabela 5.2: Configurações do ambiente de monitoramento e coleta de dados.

Ferramenta	**Configurações de *Hardware***
Nagios	Processador *Intel Xeon* de 2.40 *GHz*, disco rígido de 36 *GB*, memória RAM de 2 *GB* e 2 (duas) placas de redes de 1 *Gbps*.
Ntop	Processador de 4 núcleos, disco rígido de 119 *GB*, memória RAM de 2 *GB* e 2 (duas) placas de redes de 1 *Gbps*.
Squid/Calamaris	Processadores *Xeon Dual Processors* - X5460 de 3.1 *GHz*, disco rígido de 146 *GB*, memória RAM de *32 GB* e 2 (duas) placas de redes de 1 *Gbps*.

As duas ferramentas acima descritas foram integradas a RedUnB de forma paralela, ou seja, no mesmo período porém, com técnicas diferentes, conforme ilustra a Figura 5.2. O *Nagios* mensurou o tráfego consumido e o *Ntop* identificou o percentual de cada protocolo presente neste tráfego. Para tanto utilizou-se os recursos de *hardware* e *software* listados na Tabela 5.2 para implementação destas ferramentas. O *Nagios* coletou o quantitativo de tráfego de entrada e saída da interface interna do *firewall* de borda através do comando *GET* do protocolo SNMP, como ilustrado na Figura 5.2, num período de 24 horas em um dia típico de aula na universidade. Como o *Nagios* trabalha como amostragem de tráfego para geração de seus gráficos, escolheu-se o período de 24 horas objetivando a identificar com maior precisão os valores de pico máximo e mínimo. Contudo, o gráfico gerado pelo *Nagios* apresentou valores similares aos das ferramentas de monitoramento do CPD da UnB, tendo como diferença os valores de picos e média.

Os dados coletados estão consolidados na Figura 5.3, tendo em seu eixo vertical o tráfego consumido em *Mbps* e o no eixo horizontal o tempo decorrido em intervalos de 15 minutos. O gráfico superior corresponde ao período de 24 horas de coleta enquanto o gráfico inferior ao período em que o tráfego é mais representativo, das 09h00 às 18h00. Apesar dos gráficos oferecerem os tráfegos de entrada e saída, analisou-se apenas o primeiro. Esta decisão tem respaldo no fato de que um sistema de *web cache* raramente armazena conteúdo de saída. Por fim, infere-se da análise dos 2 (dois) gráficos presentes na Figura 5.3 que o comportamento do tráfego total consumido na RedUnB é bastante sa-

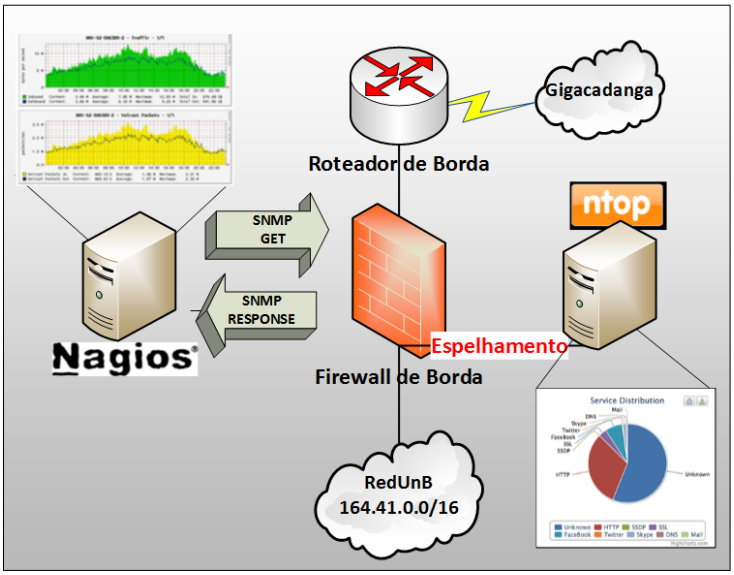
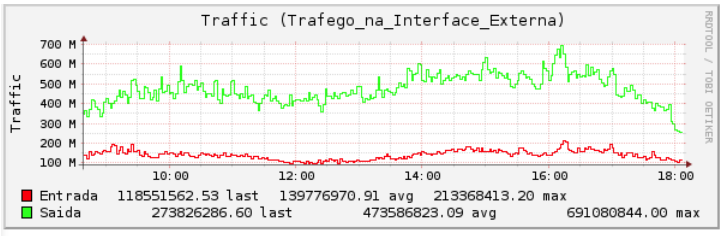
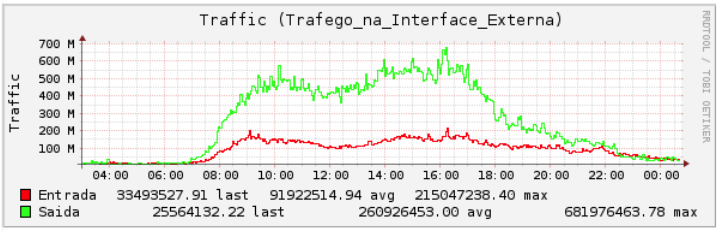


Figura 5.2: Topologia usada para estimar o tráfego HTTP.



(a) Volume de tráfego entre as 9h às 18h.



(b) Volume de tráfego entre as 0h e 23h59.

Figura 5.3: Mensuração do tráfego da RedUnB.

zonal, passando de 100 *Mps* durante a madrugada para picos de 700 *Mbps* durante o dia, apresentando uma média durante o período de maior representatividade de 473 *Mbps*. Esta sanzonalidade é corroborada com informações dos sistemas de monitoramento do Centro de Informática da UnB, apontando o mesmo formato de tráfego quando analisado anualmente.

Para que o *Ntop* estimasse de forma percentual os protocolos presentes no tráfego,

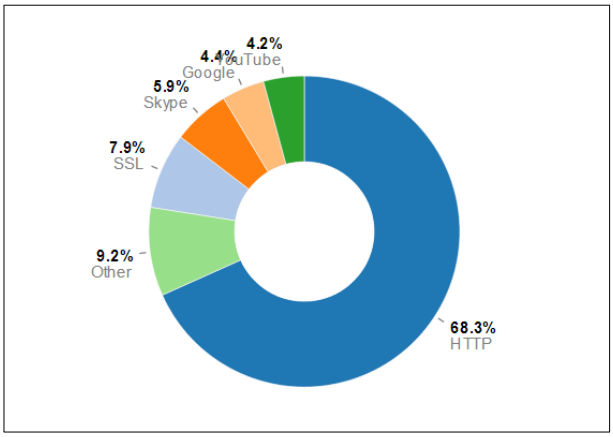


Figura 5.4: Protocolos utilizados.

levantado pelo *Nagios* fez-se necessário configurar uma de suas interfaces em modo de escuta e conectá-la logicamente ao espelhamento da interface interna do *firewall* de borda. Para tal, a interface do roteador de núcleo com o *firewall* de borda foi espelhada de forma a replicar todo o tráfego de entrada e saída da RedUnB para o *Ntop*. Esta configuração permitiu utilizar o *Ntop* para visualizar em tempo real o percentual de consumo dos protocolos presentes na borda, conforme ilustrado na Figura 5.4. Como pode ser observado nesta figura, pode-se confirmar a predominância do protocolo HTTP (HTTP, *YouTube* e *Google*) com cerca de 77% do consumo de banda total da RedUnB.

Ao consolidar as informações coletadas pelas ferramentas *Nagios* e *Ntop*, obteve-se uma estimativa de consumo do tráfego HTTP de entrada da RedUnB de 539 Mbps (77% de 700 *Mbps*) em período de picos. Esta informação será utilizada na próxima seção para planejar a implementação controlada do sistema de *proxy web cache* institucional experimental em sua configuração padrão dentro da RedUnB, objetivando identificar seu comportamento tipificando e caracterizar os objetos mais consumidos pelos seus usuários. As análises destes objetos e o comportamento do *proxy web cache* irão apontar as características necessárias para um bom modelo de *proxy web cache* institucional com boa taxas de *hits* com manutenção da latência média de forma a salvaguardar a banda de borda.

5.3 Caracterizando o Tráfego HTTP

Conforme citado na Subseção 5.2.1 o objetivo desta seção é caracterizar os objetos *web* encapsulados no tráfego HTTP consumidos pela RedUnB. Com esse objetivo, criou-se um cenário controlado para efetuar a interceptação, redirecionamento, inspeção e análise do tráfego HTTP. Para este fim, serão utilizadas as ferramentas de *proxy web cache Squid*, o *firewall* de borda e o analisador de *logs Calamaris* [59]. Os equipamentos utilizados para este experimento estão descritos na Tabela 5.2. Entretanto, em face do grande volume do tráfego *web*, pela forma que será implementado o *Squid* [2], intermediando as conexões entre clientes e servidores em tempo real, e por se tratar de um ambiente em produção, optou-se por realizar o experimento em um subconjunto de equipamentos (*hosts*) da RedUnB. Mais precisamente, selecionou-se 10 (dez) sub-redes com maior volume de tráfego HTTP. Estas sub-redes foram identificadas por meio da ferramenta *Ntop*, como visto na Subseção 5.2.1. Em virtude do sigilo institucional não serão descritos, com exceção da rede do CPD, os nomes dos departamentos nem os segmentos de redes escolhidas. O tráfego destas redes serão paulatinamente submetidos ao *Squid*, cercando-se de cuidados para que não ocorra um aumento considerável na latência de acesso ou indisponibilidade ocasionada por uma sensível elevação da carga. Para interceptação e redirecionamento do tráfego utilizou-se no *firewall* de borda a técnica de roteamento baseado em política. Esta técnica irá interceptar todas as requisições originárias da rede interna com destino a porta 80, porta padrão de protocolo HTTP, redirecionando-as para o IP do *Squid* através roteamento alternativo. O *Squid* por sua vez irá intermediar todas as requisições HTTP redirecionadas buscando armazenar o maior número possível de objetos e gerando registros de todas as solicitações. Utilizou-se de forma complementar a ferramenta *Calamaris* para analisar e consolidar as informações destes registros. Por fim, serão apresentados de forma comentada os principais resultados obtidos. A seguir será descrito as ferramentas *Squid* e *Calamaris* com suas principais características:

- ***Squid*** [60]e [2]: foi criado na década de 1990 com financiamento da NSF (Fundação

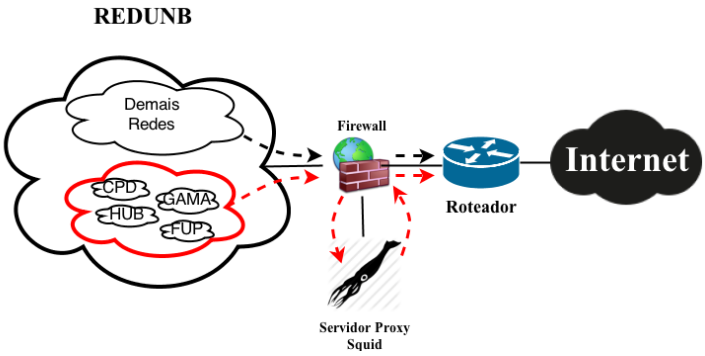


Figura 5.5: Interceptação e redirecionamento do tráfego *web*.

de Ciência Nacional) Americana como fomento para investigação de tecnologias de armazenamento em *cache*, tornando-se o primeiro *proxy web cache* da história da Internet. Aos longo dos anos tornou-se um dos mais respeitáveis e utilizados projetos de código aberto de sua categoria. Possui 2 (dois) níveis básicos de armazenamento, sendo um na memória RAM e outro no disco rígido. Os objetos mais demandados são armazenados na memória RAM, enquanto os demais objetos são armazenados no disco rígido. O *Squid* implementa diversas políticas de substituição de objeto em *cache*, entre elas o LRU, GSFD e LFUDA. Além de suportar diversos tipos de protocolos *web*, tais como FTP, HTTP, HTTPS entre outros. Permite a expansão de suas funcionalidades com utilização de módulos, *plugins* e até mesmo programas externos. Permitindo redirecionar em tempo real URLs para módulos externos, para fins tratamento, e posteriormente armazenando-as.

- **Calamaris** [59]: é um dos mais antigos e referenciados analisadores de *logs* e gerador de relatórios de sistemas de *proxy web cache*. É compatível com os servidores de *proxy web cache*: *Squid*, *NetCache*, *Inktomi Traffic Server*, *Oops! proxy server*, entre outros. Possui uma grande flexibilidade, gerando desde relatórios sumarizados até estendidos, permitindo diversos tipos e graus de visibilidade. Através de seus relatórios pode-se identificar o quantitativo de requisições: por protocolo da camada de transporte (UDP/TCP); por método (`ICP-QUERY`, `GET`, `HEAD`, etc); por tipo de conteúdo dos objetos (vídeo, áudio, dinâmico); por tipo de protocolo da camada de

aplicação (HTTP, HTTPs, FTP); por tipo de extensão dos objetos (`rar`, `zip`, `swf`, etc); por tamanho de objetos; por domínios acessados; por URLs mais acessadas; por objetos mais acessados, além de fornecer informações que permitem inferir sobre a latência média.

Durante a implementação do *Squid* no cenário ilustrado na Figura 5.5, fez-se necessárias algumas alterações na sua configuração padrão, almejando armazenar o maior número possível de objetos, identificando assim os possíveis *hits* e *miss*. Estas alterações permitiram alocar 10 *GB* para *cache* em memória RAM, 100 *GB* para *cache* em disco rígido, de forma a armazenar objetos com tamanhos entre 0 (zero) a 204.800 (duzentos e quatro mil e oitocentos) *KB*.

O modo de operação também foi alterado de *proxy* explícito para *proxy* transparente. A alteração para modo transparente permitiu que fossem aceitas requisições sem autenticação, não sendo necessárias configurações adicionais nos clientes nem tampouco alterações significativas nas configurações dos ativos de rede. Com intuito de validar as configurações definidas, submeteu-se durante uma semana parte do tráfego HTTP da rede do CPD para o *Squid*.

Após validação dos ajustes realizados no *Squid*, submeteu-se o tráfego HTTP das demais sub-redes selecionadas durante 4 (quatro) dias registrando em *logs*, todas as URLs e objetos acessados, bem como tempo de latência e taxa de *hits*. Ressalta-se que as sub-redes selecionadas correspondem as 10 (dez) sub-redes com maior consumo de tráfego HTTP. Todos estes *logs* foram analisados e copilados pela ferramenta *Calamaris*, entretanto, considera-se para este estudo apenas o período compreendido entre 09h00 às 18h00, por ser o período de maior representatividade do tráfego conforme visto na subseção anterior.

Com os dados destes relatórios gerou-se as Figuras 5.7 e 5.6 que ilustram o quantitativo de requisições e o volume, em *Gbps*, das solicitações atendidas pelo *Squid* ao longo deste período. Observa-se em ambos os gráficos que os valores são incrementados ao longo dos dias, tendo os valores mais baixo no 1º dia e os mais altos no 4º e último dia. Estes incrementos são justificados, em partes, pela adição sequenciada das redes no *Squid*, tal

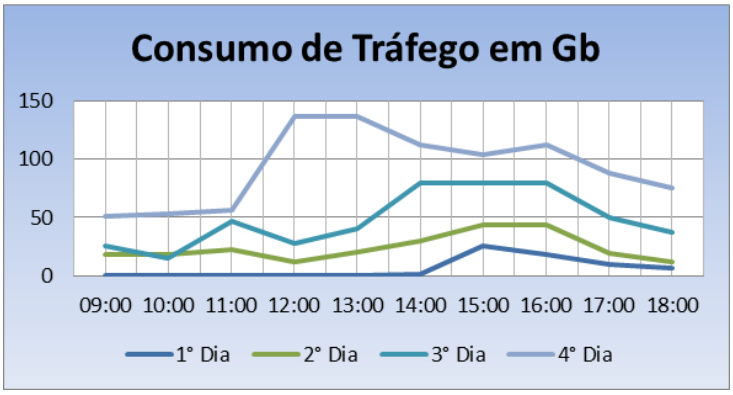


Figura 5.6: Tráfego *web* em *Gbps*.

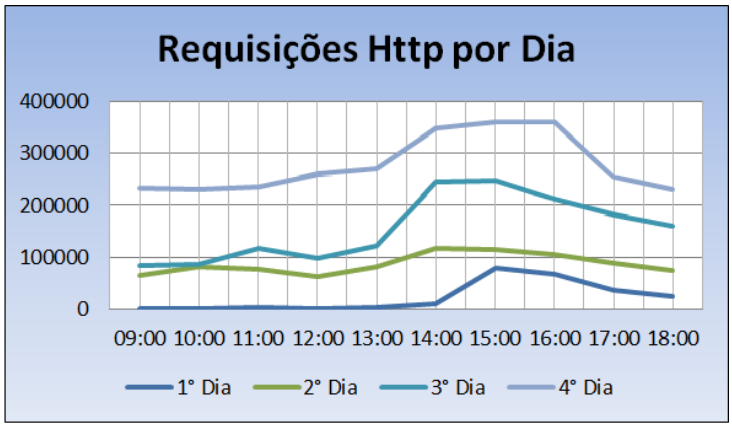


Figura 5.7: Total de requisições HTTP.

como descrito na Seção 5.2.1. Entretanto, o maior quantitativo de requisições, cerca de 380.000 (trezentos e oitenta mil), ocorreu no período das 12h00 às 13h00. Pode-se verificar também que o maior volume consumido, cerca de 148 *Gbps*, ocorreu no período das 14h00 às 16h00. Abaixo são colocadas as principias informações de destaque constantes nos relatórios:

- Mais de 64% do tráfego *web* interceptados referiram-se a *octet-stream* e vídeos;
- Em média 68,5% do tráfego *web* interceptado possuíam extensões consideradas dinâmicas entretanto, apenas 2% destas receberam *hits*;
- Em média 55% do tráfego *web* teve como destino sites de vídeos;
- A maior taxa de *hits* alcançada foi de 10% porém, representou apenas 1,95% de todo o tráfego *web* daquele dia;

- A maior parte dos objetos em *cache* são imagens e com tamanhos inferiores a 100 *bytes*;
- Em nenhum momento o *cache* chegou a 15% de sua capacidade, desta forma o espaço em *cache* não foi um limitador para *cache* de objetos.

5.4 Discussão

A partir dos resultados obtidos na Seção 5.2 pode-se inferir acerca das características de consumo do tráfego HTTP da RedUnB que o mesmo é inerentemente dinâmico, tendo uma parcela significativa composta por vídeo sob demanda. Com relação ao armazenamento de seus objetos por um sistema de *proxy web cache* pode-se inferir que ao submeter este tráfego ao sistema de *proxy web cache* convencional, obteve-se menos de 2% de *hits*.

Para o sistema de *proxy web cache* convencional as URLs foram consideradas distintas ou seus objetos foram identificados como não armazenáveis. Porém, ao analisar o arquivo de *logs* `access.log` identificando-se que a grande maioria das URLs do *YouTube* foram consideradas distintas para o sistema de *proxy web cache* convencional. Foi utilizado para o teste as URLs do *YouTube* pela sua representatividade em relação ao tráfego *web* dinâmico. Apenas 2 (duas) URLs foram consideradas iguais.

A análise apontou que grande parte das URLs do *YouTube* são distintas, entretanto, não explica a causa deste fenômeno, ficando assim uma lacuna em aberto. Para fechar esta lacuna o arquivo de *log* `access.log` foi analisado com uma maior profundidade, inferindo-se que a grande maioria das URLs do *YouTube* são longas e na maioria das vezes contém informações dos usuários, tornado-as únicas. Na Figura 5.8 foram destacadas algumas das informações dos usuários, tais como endereço de IP e sistema operacional. Todas as análises foram realizadas utilizando-se os comandos do *Shell* do *Linux*.

Das análises realizadas inferiu-se que o sistema de *proxy web cache* institucional convencional não consegue tratar as URLs dinâmicas do *YouTube* em função da passagem de diversos parâmetros nas próprias URL, tornado-as únicas na maioria das vezes. Sendo

```
GET/http://s.youtube.com/api/stats/
watchtime?adformat=2_2_1&cos=Windows&Ytimg.cvfl2Y2Exx%2F
ad3.swf&rtn=28&cbrver=29.0&cbr=Firefox .. . . . . .

GET/http://r19---sn-gpv7en7z.googlevideo.com/
videoplayback?c=web&clen=4609159&ip=164.41.219.1&signatu
re=D19D533096C1FE42C40106F370474814275A41D7.06E3 . . .
```

Figura 5.8: URLs do *YouTube*.

demandado um outro modelo de *proxy web cache* institucional capaz de tratar URLs dinâmicas.

Capítulo 6

Cache de URLs Dinâmicas

O Capítulo 5 teve como propósito a análise do comportamento de um sistema de *proxy web cache* institucional em sua configuração padrão com caracterização do tráfego HTTP de borda. Ao caracterizar o tráfego HTTP observou-se que 68.5% das URLs requisitadas eram dinâmicas e que 55% de todo o tráfego analisado estavam relacionados a objetos de vídeos. Apesar das URLs dinâmicas terem tido um percentual representativo na análise, a taxa de reuso de seus objetos não passou de 2% de *hits*. Por fim, a análise concluiu que o sistema de *proxy web cache* institucional convencional não está preparado para efetuar *cache* de objetos acessados a partir de URLs dinâmicas, principalmente no tangente a objetos de vídeos.

O objetivo deste capítulo é levantar as principais propostas de um sistema de *proxy web cache* institucional capazes de tratar as URLs dinâmicas e apontar a de melhor aderência. O capítulo foi dividido em 2 (duas) seções. Na Seção 6.1 são levantadas propostas correlatas abordando propostas acadêmicas e projetos em código aberto. Na Seção 6.2 é descrita e analisada uma proposta de um sistema de *proxy web cache* para URLs dinâmicas.

6.1 Estado da Arte em *Cache* de URLs Dinâmicas

Esta seção tem como objetivo levantar e analisar propostas de um sistema de *proxy web cache* capaz de tratar URLs dinâmicas permitindo *cache* de seus objetos. Nas Subseções 6.1.1 e 6.1.2 são levantados e analisados propostas correlacionadas a problemática em relação a literatura e a projetos de códigos abertos, respectivamente. Optou-se por também levantar e analisar propostas de projetos de código aberto, devido ao fato de não ter sido encontrado na literatura nenhuma proposta de boa aderência. Todavia, os trabalhos correlacionados levantados na literatura apresentaram técnicas e tecnologias que possibilitaram nortear a pesquisa por projetos promissores de código abertos.

6.1.1 Propostas da Literatura

No levantamento do estado da arte no Capítulo 2 elencou-se alguns trabalhos relacionados com sistema de *proxy web cache* institucional. Entretanto apenas três tiveram alguma correlação com *cache web* de conteúdo dinâmico. Os trabalhos [61] e [62] discorrem sobre o *cache* de vídeos, sendo o primeiro focado em *prefetching cache*, enquanto o segundo propõem um algoritmo de substituições e recuperação em *cache* específico para objetos de vídeos. Os 2 (dois) trabalhos tomam como pressuposto que as URLs de vídeos equivalentes referenciam os mesmos objetos. Entretanto, como visto na Seção 5.4 as URLs dinâmicas do *YouTube* apresentaram-se como únicas ao sistema de *proxy web cache*, inclusive para os mesmos objetos.

Como os trabalhos [61] e [62] não tratam as URLs dinâmicas, utilizaram como pressuposto que todas as URLs dinâmicas são cacheáveis, não sendo utilizado nesta pesquisa. Em [18] os autores discorrem sobre a natureza das URLs dinâmicas ofuscadas no processo de aquisição de dados para um sistema de base de conhecimento, buscando desofuscá-las e sumarizar as URLs equivalentes.

Desta forma, parte dos resultados obtidos em [18] pode ser aplicado para solucionar a problemática das URLs dinâmicas, haja vista que através de análise de parâmetros da

composição destas URLs os autores identificaram os parâmetros essenciais para decodificação e sumarização das URLs equivalentes. Porém, não será possível aplicar a proposta em sua plenitude, pois a técnica utilizada é baseada no diferencial entre varreduras de páginas *web* e utiliza-se para decodificação parâmetros presentes nos cabeçalhos HTTP, tais com `cookies`.

Um servidor de *proxy web cache* institucional trabalha com um número significativos de usuários e com parâmetros distintos presentes nas URLs dinâmicas, não sendo possível aplicar o recurso de diferenciação controlada de acesso a objetos previamente determinados. Todavia, o trabalho [18] apontou a técnica de reescrita de URLs dinâmicas para URLs estáticas através da decodificação e sumarização bem promissora, tornando-se assim um norte para a busca de projetos de código aberto com este fim.

6.1.2 Projetos de Código Aberto

Partindo-se do modelo de reescrita de URLs dinâmicas, buscou-se propostas e projetos de código aberto para tratamento de objetos referenciados por URLs dinâmicas, com viés para os serviços de vídeos do *YouTube*. A especificidade dado ao tratamento do *YouTube* deu-se pela sua popularidade, como visto no Capítulo 4, e pelo consumo significativo de banda, como visto no Capítulo 5. Durante a pesquisa identificou-se algumas projetos com tecnologias baseados em reescritas de URLs com base de padrões predefinidos dos serviços mais populares da *web*. Dentro das propostas identificadas destacaram-se *Yt-cache(YouTube cache Project)* [63], InComun [64] e *Projekt YouTube* [65]. Abaixo são detalhados cada um dos projetos:

- ***Yt-cache(YouTube cache Project)***: Segue o mesmo princípio de reescrita de URLs com definições de padrões previamente identificados, sendo um *fork* do projeto *YouTube-Cache - Cache YouTube videos* [66]. É disponibilizado através de um módulo que se conecta ao *Squid* 2.7.x, necessita do servidor *web* com *PHP* (do Inglês, *Hypertext Preprocessor*), banco de dados *MySQL* (do Inglês, *Structured Query Language*) e pacote para desenvolvedores da linguagem *Ruby*. O projeto parou de

receber atualizações em março de 2012, tendo suas assinaturas da base de padrões dos serviços da Internet desatualizados, como pode ser visto no repositório do código do projeto em [67];

- **InComun**: Tem como objetivo criar regras de equivalência para URLs distintas porém, equivalentes. Utiliza-se do mesmo princípio de reescrita de URLs, trabalha com o *Squid* e *Lusca*. Tem seu módulo escrito em *PHP*, necessitando da instalação de um servidor *web* com *PHP*. Apesar de ser um projeto ativo, em sua lista de *e-mail* [68] são apresentados dificuldades pelos desenvolvedores em efetuar *cache* do conteúdo do *YouTube* desde novembro de 2013.

- ***Projekt YouTube***: Projeto mantido pela Universidade *Lausitz* na Alemanha, utiliza o mesmo princípio de reescrita de URLs dinâmicas. Disponibiliza documentação básica, porém eficaz, módulo de reescrita em linguagem *Perl* e as alterações necessárias para o arquivo de configuração do *Squid*. Tem como princípio básico a interceptação das URLs dinâmicas de diversos serviços populares da Internet, inclusive do *YouTube*, e o redirecionamento ao módulo decodificador. Desde de 2012 vem sendo atualizado diretamente no site do projeto [65] e pela comunidade de código aberto em [69].

Dentre os projetos acima, escolheu-se o *Projectkt YouTube* para testes e avaliação, tendo em vista ser um projeto bastante ativo. Na Seção 6.2 serão detalhados e analisados o funcionamentos dos componentes do projeto escolhido com uma implementação básica. Por fim, no Capítulo 7 serão feitos testes de validação da eficiência.

6.2 *Projeckt YouTube*

O projeto é composto por dois componentes: (*i*) uma base de padrões conhecidos para os serviços mais populares; e (*ii*) um *módulo* de decodificação e sumarização de URLs dinâmicas equivalentes. O componente base de padrões é construída com *regex* do *Squid*

e inserida no arquivo de configuração `squid.conf`, tendo como função a identificação das URLs dinâmicas dos serviços de Internet e seu redirecionamento para o módulo de decodificação. O componente *módulo* foi escrito na linguagem *Perl* com objetivo de decodificar e sumarizar as URLs dinâmicas de forma rápida e simples. A decodificação foi criada baseada na engenharia reversa das URLs originais dos serviços de Internet, de forma a identificar apenas os parâmetros relevantes. Após a decodificação dos parâmetros relevantes é gerado uma URL única, sumarizando as equivalentes.

A Figura 6.1 ilustra o fluxo das requisições HTTP com o *módulo* implementado. As requisições chegam ao *Squid* e são conferidas com os padrões conhecidos, em caso de batimento a URL é repassada para o *módulo*, que verifica à existência dos parâmetros conhecidos. Ocorrendo o batimento dos parâmetros conhecidos a URL é decodificada, sumarizada e posteriormente referenciada ao objeto. As URLs sumarizadas torna-se única para indexação do objeto referenciado pelas URLs equivalentes, permitindo assim, a ocorrência de *hits*. No caso de não haver batimento da URL com os padrões da base, a mesma seguirá seu fluxo normal dentro do *Squid*. Entende-se como fluxo normal do *Squid* a análise da solicitação, verificação da existência dos objetos no *cache* e a posterior entrega ao usuário. Em caso dos objetos não existirem no *cache*, será feita um solicitação externa antes de responder ao usuário. Na Subseção 6.2.1 será utilizada uma implementação básica para detalhar o funcionamento dos componentes do projeto.

6.2.1 Disponibilização de Ambiente de Avaliação

Nesta subseção serão descritos e analisados os componentes do projeto *Projekt YouTube* a partir de uma implementação básica, de forma a detalhar seus funcionamentos no tratamento de URLs dinâmicas do *YouTube*. Para esta análise utilizou-se um ambiente com duas máquinas, sendo uma cliente e outra servidor de *proxy*. A máquina cliente possuía o sistema operacional *Windows* 7 e o navegador para páginas *web Firefox*. A máquina servidor possuía o sistema operacional do *Debian* 7, a ferramenta de *proxy web cache*, *Squid* na versão 2.7.9, o pacote da linguagem *Perl* e o módulo decodificador de URLs

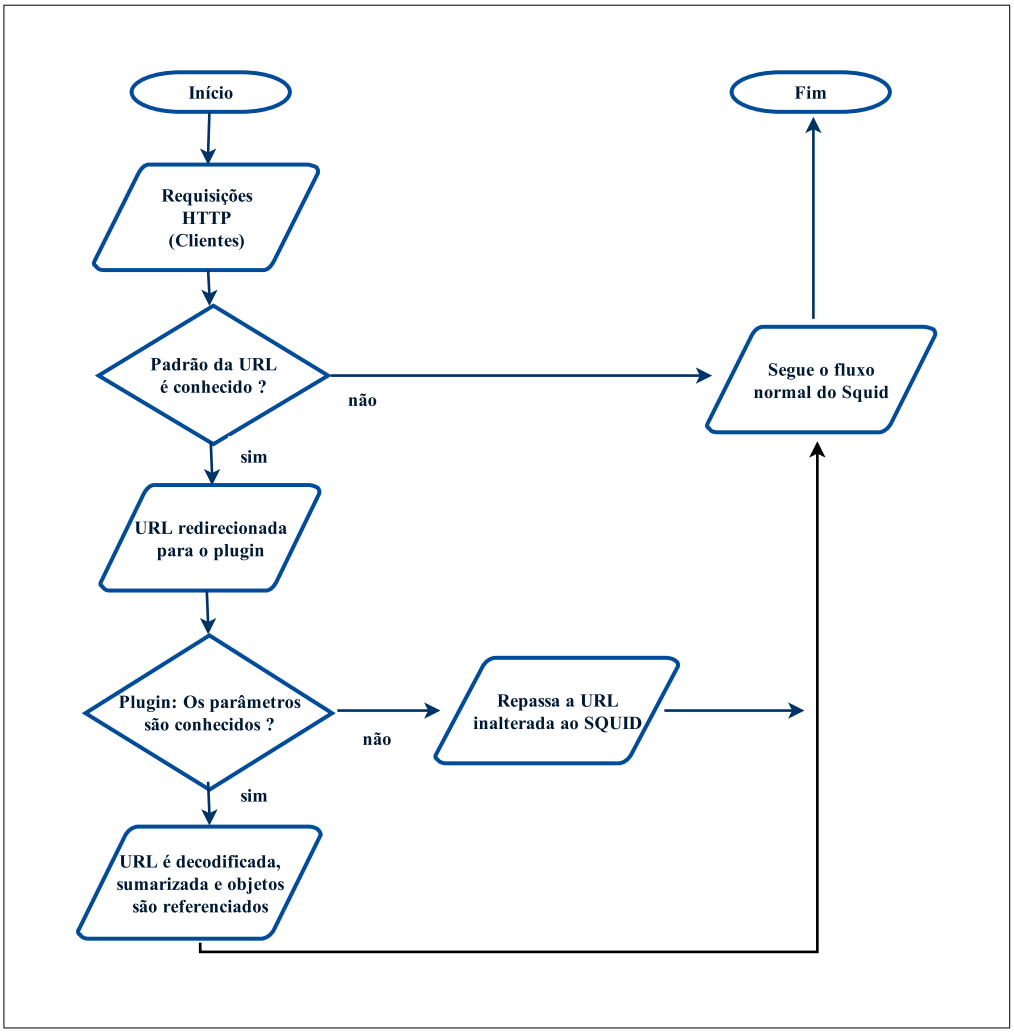


Figura 6.1: Fluxo de requisições HTTP no *Squid* com o módulo decodificador.

dinâmicas. Além do módulo decodificar também inseriu-se a base de conhecimento de padrões do projeto *Projekt YouTube* no arquivo de configuração `squid.conf`. Todo o tráfego da máquina cliente foi interceptado e redirecionado para a máquina servidor de *proxy*.

Acessou-se a partir da máquina cliente em dois momentos distintos o vídeo mais popular do *YouTube* desde 2007 - *Psy - Gangnam Style*. O primeiro acesso, que será denominado de 1º (primeira) fase, teve como objetivo popular o *cache* como os objetos do vídeo escolhido, enquanto que no segundo acesso, denominado de 2º (segunda) fase, teve como objetivo analisar a eficiência do módulo decodificador em relação ao reuso destes objetos. Para detalhar o funcionamento do módulo durante o tratamento das URLs

Regex com Padrão de Identificação de URLs do *YouTube* - Objetos de Vídeos

```
acl store-rewrite-list urlpath-regex
\/(get-video\?|videodownload\?|videoplayback.*id)
```

Diretiva para Redirecionamento de URLs

```
storeurl-rewrite-program /etc/squid/storeurl.pl
```

Figura 6.2: Padrões de URLs do *YouTube* e redirecionamento.

dinâmicas foram analisados durante as duas fases os arquivos de *logs*: (*i*) `access.log`, (*ii*) `storeurl.log` e (*iii*) `store.log`. O `access.log` registrou todas as URLs recebidas pelo *Squid*, o `storeurl.log` registrou todas as URLs que foram redirecionadas para módulo decodificador e o `store.log` registrou as entradas em *cache* dos objetos solicitados.

Como ilustrado no diagrama de fluxo Figura 6.1, o primeiro processo executado quando o servidor de *proxy web cache* possui implementado o módulo decodificador de URLs dinâmicas e a base de padrões é o batimento da URL requisita pelo protocolo HTTP com os padrões conhecidos. Tendo encontrado algum padrão conhecido a URL é redirecionada para o módulo de decodificação, caso contrário a requisição segue seu fluxo normal dentro do *Squid*. Como visto na Figura 6.2, o padrão para identificação das URLs do *YouTube* é composto pelos parâmetros: `get-video\%?`, `videodownload\?` e `videoplayback.*.id`. Sendo identificado pelo menos um destes parâmetros, a URL será redirecionada. Na mesma figura observa-se que o redirecionamento deve ser feito para o *plugin*, nessa pesquisa denominado de módulo decodificador, `/etc/squid/storeurl.pl`.

Após a URL ser enviada para o módulo decodificador, tenta-se obter os parâmetros relevantes previamente conhecidos e posteriormente sumarizar as URLs. Na Figura 6.3, os parâmetros relevantes da URL do *YouTube*, tais como `itag` e `range`, são decodificados de forma direta, enquanto os parâmetros `cpn` e `id` são utilizados de forma indireta para decodificar o parâmetro de `video-id` - `9bZkp7q19f0`. Tendo como saída uma URL sumarizada compostas pelos parâmetros da decodificação.

O processo de sumarização resultam em URLs construídas com parâmetros mínimos

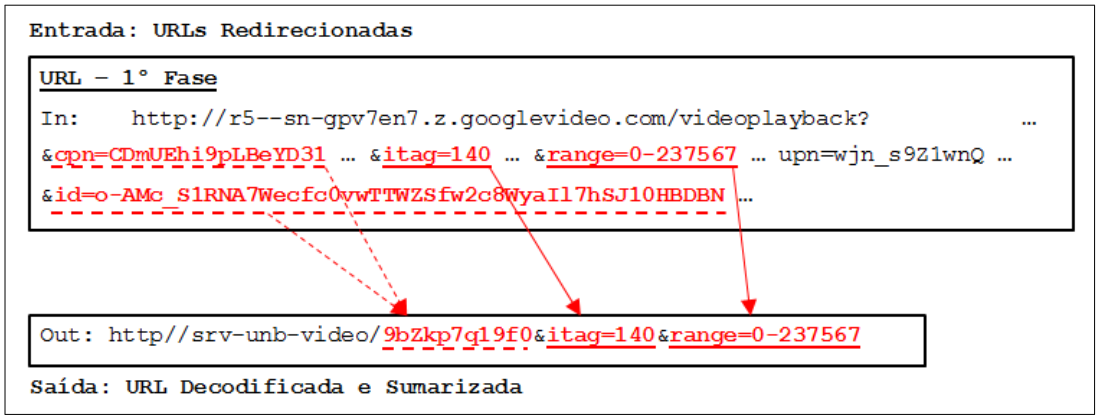


Figura 6.3: Fluxo de entrada e saída de URLs no módulo de decodificação de URLs dinâmicas.

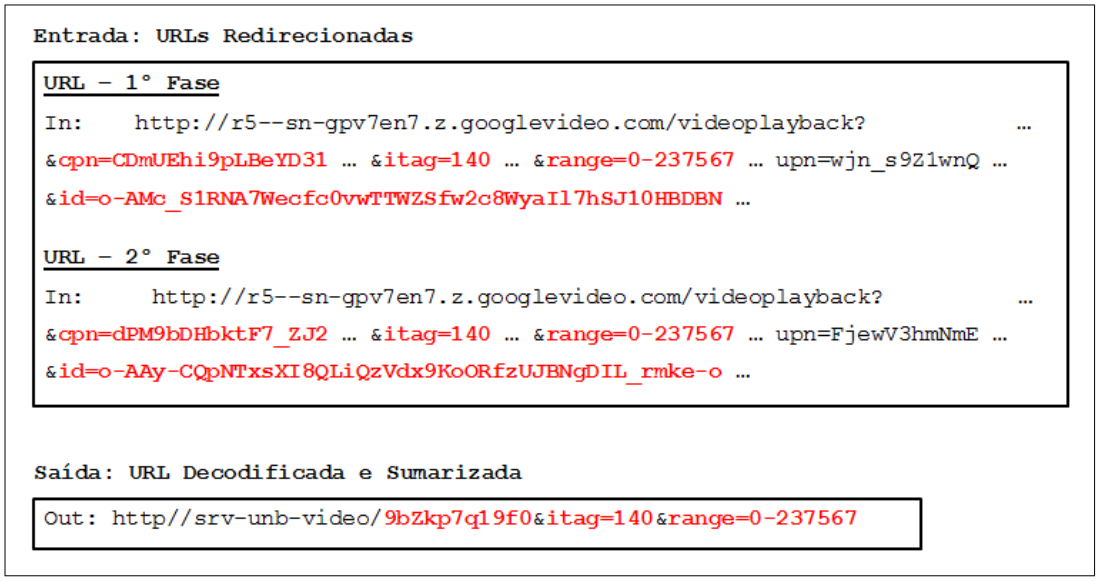


Figura 6.4: Decodificação e sumarização de URLs equivalentes do *YouTube*.

que referenciam todas as outras URLs equivalentes. A Figura 6.4 ilustra 2 (duas) URLs aparentemente distintas acessadas em momentos diferentes, sendo uma na 1º (primeira) fase e a outra na 2º (segunda) fase. Logo em seguida são mostrados os parâmetros `range` e `itag` decodificados e posteriormente sumarizados, tendo como saída uma única URL. Esta nova URL é equivalente à quaisquer requisições de vídeos com `videoid=9bZkp7q19f0`, `itag=140` e `range=0-237567`, mesmo em URLs distintas.

O mais importante da sumarização é a possibilidade de fazer reuso dos objetos a partir da URL sumarizada. A Figura 6.5 ilustra as entradas do `access.log` informado o reuso do objeto em *cache*, mesmo as URLs sendo distintas. Os parâmetros em destaques `cpn`

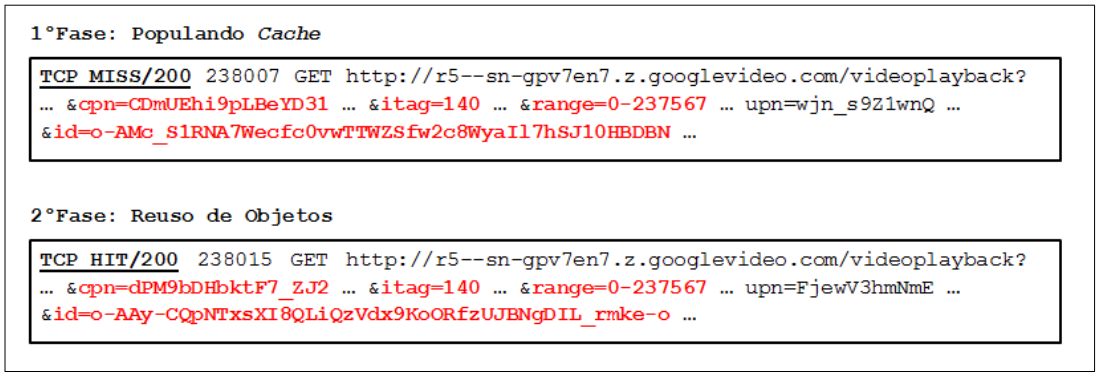


Figura 6.5: *Hits* de objetos para URLs distintas, mas equivalentes.

```
SWAPOUT 00 00000012 3925B7FDF8974F0668BA531034F6899 200 … 237568/237568
GET http://r5--sn-gpv7en7.z.googlevideo.com/videoplayback?
… &cpn=CDmUEhi9pLBeYD31 … &itag=140 … &range=0-237567 … upn=wjn_s9Z1wnQ …
&id=o-AMc_S1RNA7Wecfc0vwTTWZSfw2c8WyaIl7hSJ10HBDBN …
```

Figura 6.6: Rastreando gravação de objeto no *cache*.

e `id` são diferentes em cada uma das URLs tornando-as distintas. Na primeira fase não houve *hits* apenas *miss*, enquanto na segunda fase ocorreu *hits*. Apesar de distintas, as URLs eram equivalentes, pois possuíam os mesmos parâmetros de `range`, `itag` e `videoid`.

O parâmetro de `videoid` é obtido indiretamente a partir dos parâmetros `cpn` e `id`, em um dado momento é passado uma URL onde são correlacionados os três parâmetros. Utiliza-se o parâmetro `videoid` em detrimento dos parâmetros `cpn` e `id` para sumarização das URLs. Este parâmetro é considerado único e invariável, independente da sessão HTTP, ao contrário dos demais parâmetros.

Para confirmar se o objeto foi realmente armazenado em *cache*, analisou-se os registros do `store.log`. Na Figura 6.6 é ilustrada apenas uma entrada de um objeto em disco, referente a URL da primeira fase. Isso ocorre pois a segunda solicitação de objeto foi dada como existente, sem a necessidade de gravar mais objetos em discos. O objeto gravado em disco foi armazenado no diretório `00` e dado o nome de `00000012`, a fim de constatar se o arquivo realmente existe foi listado o arquivo e seus respectivo tamanho, tal como pode ser visto na Figura 6.7

```
root@proxy-unb:/# ls -l /cache/00/00/00000012
-rw-r---- 1   proxy proxy   238732                /cache/00/00/00000012
                    (Tamanho do Objeto no SA)   (Localização do Objeto no SA)
```

Figura 6.7: Confirmado gravação de objetos no sistema de arquivos.

6.2.2 Discussão

Nesta seção foi detalhado o funcionamento dos componentes do projeto *Projectk YouTube* através de uma implementação básica. Durante a implementação tanto o componente do módulo de decodificar de URLs dinâmicas, quanto o componente da base de padrões mostraram-se funcionais inclusive com obtenção de *hits* para os objetos de vídeos do *YouTube*. Porém, a implementação básica não foi suficiente para comprovar a eficiência do projeto. Desta forma, identificou-se a necessidade de criação de um ambiente mais adequado para validação dos módulos. Para tal, um cenário contendo uma maior quantidade de vídeos e como alternância de alguns parâmetros de forma a simular os diversos tipos de acessos feitos pelos usuários, serão apresentados e avaliados no Capítulo 7.

Nesta seção também observou-se parâmetros básicos relevantes descritos em trabalhos anteriores [49], que não foram encontrados ou tiveram seu significado alterado ao longo do tempo. Um dos parâmetros que teve seu significado alterado foi o `id`, que conforme descrito na Seção 4.3, referia-se a identificação única do vídeo. Observa-se nas Figuras 6.5 e 6.4 que o parâmetro `id` variou com valores distintos para cada URL, apesar de tratar do mesmo vídeo. O parâmetro `id` atualmente é utilizado de forma indireta para obter o parâmetro de `videoid`, variando de acordo com as sessões. Já o parâmetro `cpn` não chega a ser citado e têm as mesmas características do parâmetro `id`. Estas alterações da significância e na existência dos parâmetros de construção das URLs do *YouTube* inferem que este serviço está em constante alteração. Os fóruns do *Squid* [70] e InCommum [68] corroboram com esta afirmação. Em particular, há registros nestes fóruns sobre alterações nos parâmetros das URLs do *YouTube* em 2013 e 2014.

Capítulo 7

Avaliação do Módulo Decodificador de URLs Dinâmicas

Para mensuração dos testes utilizou-se como métricas as taxas: `HRR` (do Inglês, *Hit Request Rate*), `MRR` (do Inglês, *Miss Request Rate*), `HBR` (do Inglês, *Hit Byte Rate*), `MBR` (do Inglês, *Miss Byte Rate*) e o tempo de TCP. As métricas `HRR`, `MRR`, `HBR` e `MBR` estão relacionadas com as taxas de acertos e perdas de objetos encontrados em *cache*, sendo a `HRR` e a `MRR` relacionadas com o quantitativo de requisições e a `HBR` e a `MBR` com o volume. O tempo de TCP se refere ao tempo transcorrido nas seções de TCP para recuperação dos objetos solicitados. Todas as métricas foram escolhidas baseadas em trabalhos anteriores sobre *proxy web cache*, tal como visto no Capítulo 3.

Os testes executados buscaram mensurar a eficiência do módulo em relação as URLs dos objetos de vídeos, com alternância dos parâmetros de qualidades de vídeo e dos navegadores. Foram escolhidos para os testes 28 (vinte oito) vídeos populares, sendo cada um deles requerido uma única vez em uma das 6 (seis) qualidades predefinidas. As qualidades foram escolhidas baseadas no guia de *API* do *YouTube* [46], sendo elas: `default`, `small`, `large`, `medium`, `HD720` e `HD1080`.

Para escolha dos vídeos foram utilizados os critérios de popularidade, restrição e duração. Os vídeos escolhidos estavam figurando na lista do mais acessados a partir do Brasil

desde 2007 [71] e no mês de abril de 2014 [72], sem restrição de exibição e com duração inferior a 9 minutos. Os critérios adotados para escolha dos vídeos buscaram refletir as preferências dos usuários brasileiros e ao mesmo tempo viabilizar a automação dos testes.

Os navegadores também foram escolhidos com base na popularidade, elencando os três mais populares em uso durante o período de abril de 2013 a abril de 2014, sendo utilizado como fontes os sites especializados em mercado de sistemas operacionais e navegadores. As fontes utilizadas foram: *Netshare* [73], *Clicky Web Analytics* [74], *W3counter* [75] e *Stat Counter Global Stats* [76], todos eles utilizam como metodologia agentes embarcados em servidores *web* para identificação dos sistemas operacionais e navegadores utilizados pelos usuários. Em todas as fontes pesquisadas os três navegadores mais populares foram o *Chrome*, *Firefox* e o *Internet Explorer*, tendo como sistema operacional mais popular o *Windows* 7.

Como será visto na Seção 7.1, os testes foram executados em 2 (dois) cenários, sendo distintos entre si em função da existência do módulo decodificador de URLs dinâmicas.

A Seção 7.2 descreverá os resultados obtidos, destacando a eficiência do módulo decodificador de URLs dinâmicas. Alcançou-se nos testes, com uso do módulo, taxas de *hits* acima de 99% e valores de tempo de TCP menores, em relação aos testes sem implementação do módulo. Os resultados dos testes indicam que o tipo de navegador e qualidade do vídeo não possuem influência significativa nos resultados. Em relação às taxas de *miss*, identificou-se que as mesmas ocorreram em função de erros no decodificador e devido a troca automática de qualidade. Entretanto, em ambos os casos não ocorreram em quantidade significativa.

Na Seção 7.3 será relembrada a natureza dinâmica das URLs do *YouTube* e o *modus operandi* do módulo decodificador de URLs dinâmicas, de forma a justificar os resultados obtidos. Por fim, é ressaltado que os erros eram esperados, pois se trata de um trabalho de engenharia reversa.

7.1 Descrição do Ambiente

Nesta seção é descrito o ambiente de testes, detalhando os cenários bem como os recursos computacionais e topologias utilizadas. O ambiente é composto por 2 (dois) cenários que utilizam os mesmos recursos computacionais e topologia, diferindo entre si apenas pela implementação do módulo de decodificação de URLs dinâmicas. No primeiro cenário não foi implementado o módulo decodificador, tendo como objetivo simular um ambiente convencional de um sistema *proxy web cache* institucional. Sendo executado o teste denominado `Linha Base` com a finalidade de mensurar as métricas sem a implementação do módulo. No segundo cenário foi implementado o módulo de forma a simular o mesmo ambiente presente no cenário anterior, agora capaz de tratar URLs dinâmicas. Para isso realizou-se 3 (três) testes, `T1`, `T2` e `T3`, sendo exatamente iguais para que os resultados fossem consolidados e verificado a existência de alguma anomalia provocada pelo uso do módulo.

Em todos os testes utilizou-se um computador pessoal com o sistema operacional *Windows* 7 e navegadores para acessar uma sequência de vídeos, previamente determinada, passando por um servidor *proxy web cache* com *Squid*. Na Tabela 7.1 são descritos os recursos computacionais utilizados em ambos cenários, detalhando o *hardware* e *software* utilizados, na qual são elencados os equipamentos `M1` e `M2`.

O `M1` foi utilizado como servidor de *proxy web cache*, com a utilização da ferramenta *Squid*, e como servidor de análise de *logs*, com a ferramenta *Calamaris*. Além de analisar os *logs* do *Squid* a ferramenta, *Calamaris*, também confeccionou diversos relatórios.

O `M2` foi utilizado como cliente de acesso aos vídeos do *YouTube*, para tanto contou com um sistema operacional *Windows* 7 e com os navegadores: *Chrome*, *Firefox* e *Internet Explorer*. O `M2` também contou com um servidor de páginas *web Apache* utilizado para automatizar o acesso aos vídeos.

A Figura 7.1 ilustra a disposição dos recursos utilizados bem como o fluxo de tráfego existente. Observa-se que o `M2` dispunha de um servidor *web* para disponibilizar páginas HTML customizadas com a listagem dos vídeos do *YouTube* escolhidos de forma

Tabela 7.1: Recursos computacionais.

Equip.	***Hardware***	***Software***
`M1`	Processador de 2 núcleos 3.1 Ghz - (X5460); disco rígido de 146 GB e memória RAM de 32 GB.	S.O. *Debian* 7; *proxy web cache Squid* 2.7.9 e o analisador de *logs Calamaris* 2.9
`M2`	Processador *Intel* I5; disco rígido de 500 GB e memória RAM de 4 GB	S.O.: *Windows* 7; navegadores *Chrome* 35.x, *Firefox* 29.x e *Internet Explorer* 11.x e servidor de páginas *web Apache* 2.x.

automatizada. Para acessar as páginas HTML o `M2` utilizava-se de três navegadores diferentes, gerando tráfego para Internet de saída e entrada, caracterizados pelos fluxos `A` e `B`, respectivamente.

O `M1` por sua vez exerceu o papel de servidor de *proxy web cache* e interceptava todo o tráfego de Internet do `M2` tentando atendê-la localmente. Quando o objeto não era encontrado no *cache* era gerado um fluxo `C` de tráfego externo para Internet, tendo como resposta o fluxo de entrada `D`. Por fim, o servidor atendida as solicitações de seu cliente com o fluxo `B`.

As páginas HTML geradas para automatização dos testes continham códigos de *Java Script* embarcados, com instruções para acesso sequencial dos 28 (vinte e oito) vídeos selecionados com as qualidades pré-determinadas. As qualidades de vídeos utilizadas foram: `default`, `small`, `medium`, `large`, `HD720` e `HD1080`.

Conforme já descrito no início do Capítulo 7 os vídeos escolhidos seguiram critérios de popularidade, duração e restrição. Sendo selecionados vídeos com até 9 minutos de duração, grande popularidade de acesso a partir do Brasil e sem restrições de visualizações.

Por fim, ao término de cada teste utilizou-se um *script*, desenvolvido em *shell script*, que recolhia os *logs* brutos do *Squid* dando-lhes tratamento. Primeiro os *logs* passavam por uma sanitização com objetivo de separar apenas as requisições dos objetos de vídeos, excluídos imagens e outras solicitações. Posteriormente gerava-se relatórios com a utilização da ferramenta *Calamaris*, extraindo os dados relevantes para criação de gráficos. Os

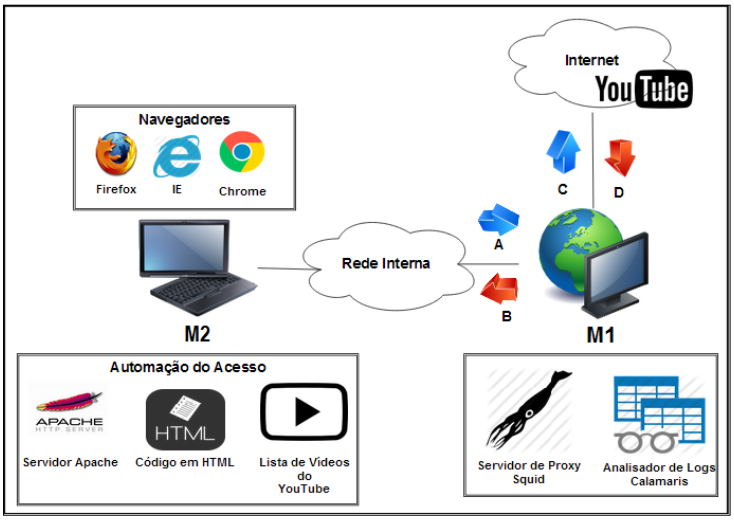


Figura 7.1: Topologia dos cenários de testes.

gráficos apresentam, de forma sumarizada, os resultados obtidos nos testes.

Durantes os testes mensuraram-se as métricas de recuperação de objetos em *cache* `HRR`, `MRR`, `HBR`, `MBR` e tempo de TCP. Sendo `HRR` e MRR as métricas de recuperação de objetos em *cache* em relação as requisições dos clientes, enquanto que as métricas `HBR` e `MBR` estão relacionadas com o volume. A métrica tempo de TCP refere-se aos tempos das sessões TCP durante as requisições dos objetos.

Como visto no Capítulo 2, para que um sistema de *proxy web cache* seja considerado eficiente é necessário que os valores das taxas `HRR` e `HBR` sejam significativas, bem como os tempos médios de TCP relativamente baixos.

7.2 Resultados dos Testes de Eficiência

A Seção 7.1 descreveu o ambiente de testes. Em cada um dos testes foram feitos acessos a 28 (vinte e oito) vídeos em umas das 6 (seis) qualidades pré-definidas, através dos navegadores *Chrome*, *Firefox* e *Internet Explorer* e tendo o tráfego intermediado por um *proxy web cache*. Para popular o *cache* do *proxy* foram feitos acessos aos vídeos não computados antes do teste de `Linha Base` e do teste `T1`.

Nesta seção serão comparados os resultados obtidos durantes os testes buscando identificar a eficiência do módulo de decodificação. Sendo descrito inicialmente uma visão geral dos resultados e os motivos do *miss*, depois são detalhados as taxas de `HRR`, `MRR`, `HBR` e `MBR` e finalizando com o tempo médio de TCP. Em todos os testes sumarizou-se as taxas `HRR`, `MRR`, `HBR` e `MBR` calculando-as a partir da média de todas as exibições por navegador e teste, optando por apresentar os dados desta forma em virtude da grande similaridade dos mesmos.

Já na métrica de tempo de TCP trabalhou-se com as médias dos testes `T1`, `T2` e `T3` em função da qualidade de vídeo e do tipo de navegador não apresentarem similaridade dos dados, ao contrário das demais métricas. Para as métricas de *hits* e *miss* não foi identificado variações significativas ao alterar o tipo de navegador qualidade do vídeo, diferentemente da métrica de tempo TCP.

- **Visão geral:** Com exceção do teste de `Linha Base` que não contou com o módulo de decodificação de URLs dinâmicas, praticamente não houve *miss* significativas, como pode ser observado na Figura 7.2, tendo-se taxas de *hits* por requisição e por volume acima de 99% demostrando assim a eficiência do módulo decodificador. Ao analisar os *logs* do *Squid* que obtiveram *miss*, identificaram-se duas causa distintas sendo elas: (i) erro do decodificador e (ii) mudança arbitrária na qualidade dos vídeos pela aplicação do *YouTube*.

  Da análise identificou-se que o decodificador apresentou erros no *parse* de duas URLs, porém, esta valor não é significativo quando comparado ao total das requisições que foram mais de nove milhões. Todavia, na alteração deliberada de qualidade de vídeos houve uma taxa mais significativa, alcançando o valor máximo em testes individualizados de 11,64% por volume, ao considerar o valor total dos *miss* obtidos nos testes este valor deixa de ser significativo.

  Na Figura 7.2 é ilustrada o comparativo percentual das taxas de *hits* e *miss*, em relação ao quantitativo de requisição e de volume, obtidas em cada um dos testes.

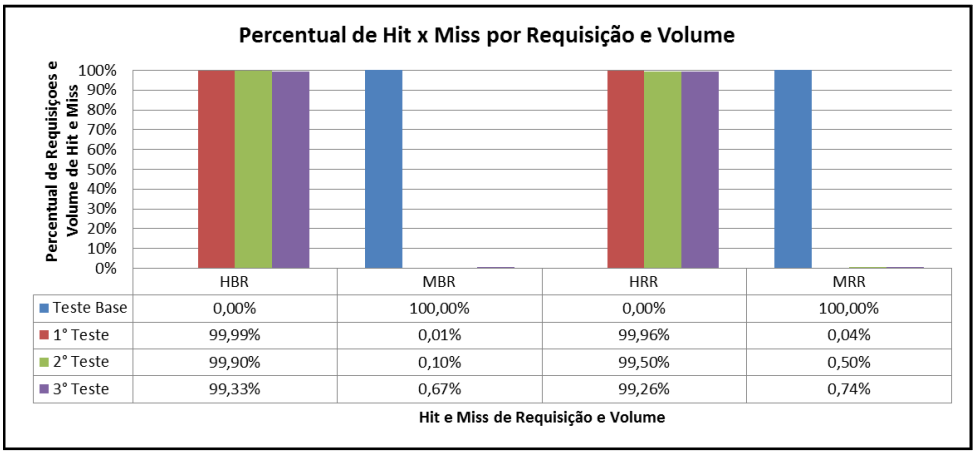


Figura 7.2: Percentual de requisições e volume de *hits* x *miss*.

No eixo horizontal tem-se os resultados agrupados em `HBR`, `MBR`, `HRR` e `MRR`. No eixo vertical, tem a mensuração do percentual das taxas de `HBR`, `MBR`, `HRR` e `MRR`.

Da análise da Figura 7.2 observa-se que as taxas de *miss* só são significativas no teste de `Linha Base`, com 100%, sendo os demais testes envoltos por taxas irrisórias, não ultrapassando 1%. Com relação às taxas de *hits* tem-se uma inversão com valores acima de 99% nos testes `T1`, `T2` e `T3`, enquanto no teste de `Linha Base` teve-se 0%. Vale ressaltar que o teste de `Linha Base` não contou com o decodificador de URLs dinâmicas ao contrário do demais. O teste `T1` apresentou o melhor resultado com percentual mínimo de *hits* em 99,9%. O teste `T3` apresentou o pior resultado com percentual mínimo de *hits* em 99,2%. O teste `T3` teve o pior desempenho em função da mudança automática de qualidade ocorrida durante exibição de alguns vídeos. A mudança foi provocada pelo próprio *YouTube* sem motivo aparente.

- **Resultados do percentual de *hits* e *miss* em relação ao volume:** Na Figura 7.3 é detalhado o percentual do volume de *hits* e *miss* obtidos durantes os testes. No eixo horizontal tem-se o agrupamento dos testes por `HBR` e `MBR`. No eixo vertical tem a mensuração do percentual do volume de `HBR` e de `MBR`. Da análise da Figura 7.3 identifica-se que foram alcançadas taxas mínimas de 99% de *hits* por volume em todos os testes que dispunham do módulo decodificador de URLs.

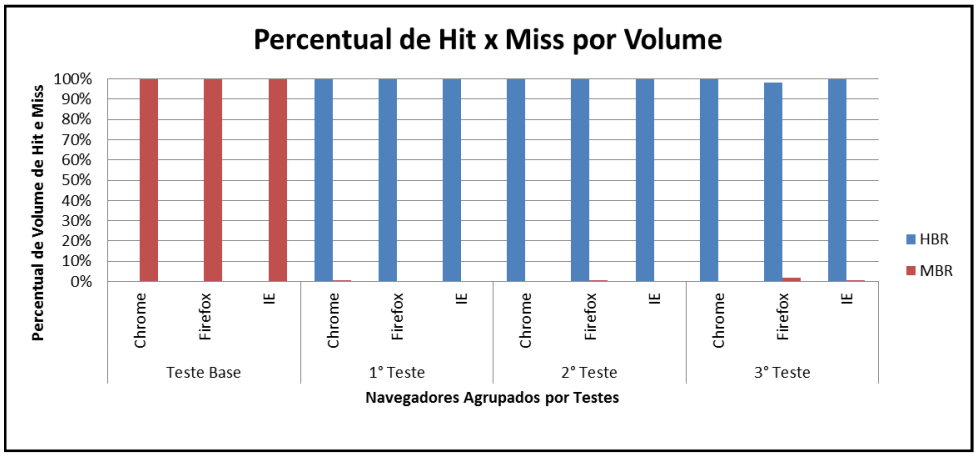


Figura 7.3: Percentual de volume de *hits* x *miss*.

Apesar de não haver diferenças significativas entre o uso dos navegadores durante os testes, o *Firefox* teve uma pequena quantidade de *miss* no teste `T3`, tornando-o menos eficiente. O *Chrome* e o *Internet Explorer* foram bem similares, tendo baixíssimas taxas de *miss*. Os *miss* observados no *Firefox* foram causadas pela mudança automática de qualidade de vídeo, já no *Chrome* e *Internet Explorer* foram causados por erro de decodificação do módulo. A mudança automática de qualidade foi gerada pelo *YouTube* sem motivação aparente, quanto ao erro do módulo de decodificação ocorre devido a necessidade de refinamento do mesmo. Ao final dos testes houve uma salvaguarda de 6,45 GB de objetos de vídeos, este número refere-se ao total do volume de tráfego que seria consumido sem o uso do módulo decodificador durante os testes.

- **Resultados do percentual de *hits* e *miss* em relação as requisições:** Na Figura 7.4 é detalhado o percentual de requisições *hits* e *miss* de forma a comparar os resultados obtidos em cada um dos testes. No eixo horizontal tem-se o agrupamento dos testes por `HRR` e `MRR`. No eixo vertical, tem-se a mensuração percentual das requisições. Observa-se que em todos os testes em que foram implementados o módulo decodificador as taxas de *hits* mínimas foram de 99%, enquanto no testes sem o decodificador, não houve taxas de *hits*.

Apesar das taxas de *miss* não serem significativas em relação a diferença do uso dos

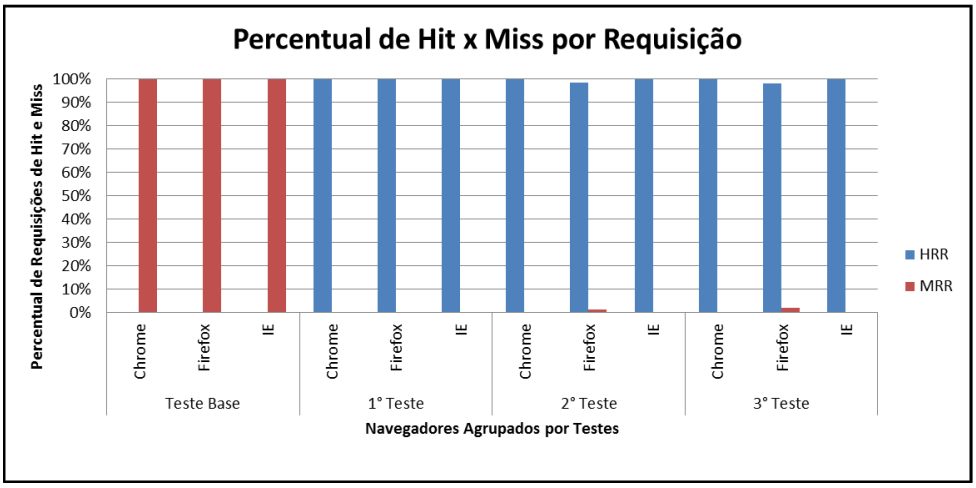


Figura 7.4: Percentual de requisições de *hits* x *miss*.

navegadores, existiu *miss* nos testes `T2` e `T3` com uso do *Firefox*, tornando-o menos eficiente. Os *miss* no teste `T2` foram relativos a erro do módulo, enquanto que no teste `T3` a causa foi a troca automática de qualidade de vídeo. A troca automática de qualidade foi provocada pelo *YouTube* sem motivação aparente, enquanto o erro provocado pelo módulo de decodificação exige um maior refinamento do mesmo.

- **Resultados do tempo de TCP:** A Figura 7.5 abaixo ilustra o comparativo entre os testes do tempo médio das sessões TCP em relação aos navegadores e as qualidades utilizados. No eixo horizontal tem-se o agrupamento das qualidades de vídeos em função dos navegadores utilizados e no eixo vertical, os valores médios das sessões TCP em (ms). As barras representam os resultados do teste de `Linha Base` e a linha os resultados médios dos testes `T1`, `T2` `T3`, em relação a cada uma das qualidades e de acordo com o navegador específico.

  Nota-se que em todas as situações os valores médios das sessões TCP dos testes realizados com o módulo decodificador de URLs dinâmicas foram menores em relação ao teste de `Linha Base`, que não teve a implementação do módulo. Também notou-se que durante o teste de `Linha Base` os vídeos na qualidade `HD1080` e `HD720` tinham exibições extremamente lentas, inclusive com travamentos. Este comportamento não foi notado durante os testes `T1`, `T2` e `T3`.

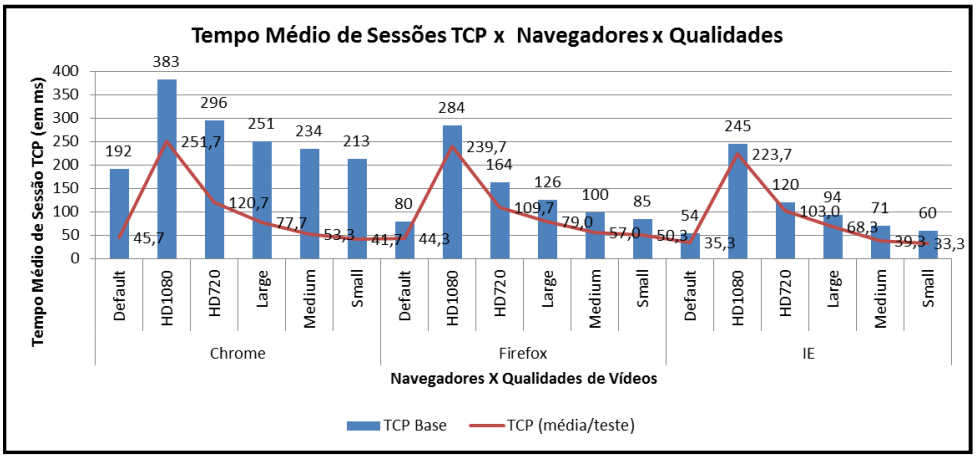


Figura 7.5: Tempo de sessões TCP.

No tempo médio das sessões de TCP houve diferenças significativas quanto a qualidade dos vídeos e o navegador utilizado, ao contrário do que foi observado com as métricas de *hits* e *miss*. A maior diferença encontrada entre os testes `T1`, `T2` e `T3` e o teste de `Linha Base` foi de 131,3 ms (383 – 251,7) durante a execução do teste `T1`, com a utilização do navegador *Chrome* para visualização de vídeos em qualidade de `HD1080`. Entretanto, o *Chrome* apresentou-se menos eficiente entre os navegadores com as maiores médias. O navegador que apresentou as menores médias nas sessões TCP foi o *Internet Explorer* seguido pelo *Firefox*.

Nesta seção pode-se constatar através dos resultados obtidos que a utilização do módulo de decodificação de URLs dinâmicas aumentou significativamente as taxas de *hits* e diminuiu o tempo de recuperação dos objetos de vídeos do *YouTube*, quando comparado os testes `T1`, `T2` e `T3` em relação ao teste de `Linha Base`. A taxa de *hits* mínimas foram de 99%, tanto em requisições, quanto em volume, tendo um total de volume salvaguardado de 6,45 GB.

Tendo os melhores resultados em relação as taxas de *hits* alcançados no teste `T1`. Os navegadores que tiveram as melhores taxas de *hits* foram *Chrome* e *Internet Explorer* praticamente empatados e o navegador que teve o menor tempo médio de TCP foi *Internet Explorer*. O navegador mais eficiente durante os testes foi o *Internet Explorer*.

7.3 Discussão

Os resultados apresentados na Seção 7.2 com taxas de *hits* acima de 99% e o com tempo das sessões TCP menores quando implementado o módulo decodificador de URLs dinâmica, indica que o mesmo é promissor. Mesmo com acesso feito de três diferentes navegadores e de qualidades distintas, o módulo tratou a maioria das solicitações.

Como descrito no Capítulo 6, o módulo decodifica e sumariza as URLs de um mesmo vídeo do *YouTube* que esteja na mesma qualidade e com o mesmo *range*, tornando-as equivalentes. Ao decodificar as URLs o módulo excluiu todos as informações irrelevantes e ao sumarizar gera a indexação dos objetos em *cache* para os parâmetros mínimos e relevantes. Sem o módulo as URLs equivalentes são tidas como diferentes, não tendo gerando *hits* de seus objetos.

Apesar, do módulo apresentar alguns erros, sendo a taxa não significativa, os mesmos já eram esperados, haja vista que sua construção foi baseada na engenharia reversa da troca de tráfego entre os clientes e servidores do *YouTube*. Os erros apresentados pelo módulo não causaram danos a exibição dos vídeos, apenas fez com que algumas requisições não tivessem *hits*.

Por fim, os resultados obtidos demonstram a viabilidade de aplicação deste módulo para o tratamento de URLs dinâmicas em um ambiente institucional que possui um tráfego intenso de conteúdo dinâmico, tal como universidades. Melhorando a experiência de navegação do usuário e ao mesmo tempo salvaguardando banda para outras atividade, tais como uso de laboratórios e supercomputadores externo a estas instituições.

Capítulo 8

Conclusão e Trabalhos Futuros

Esta pesquisa teve como foco as dificuldades encontradas pelos sistemas de *proxy web cache* institucionais, em atingir taxas de acertos de *hits* significativa com a manutenção média da latência de objetos referenciados por URLs dinâmicas. Buscando delimitar melhor a problemática, foi analisado o comportamento de um sistema de *proxy web cache* na rede da Universidade de Brasília, caracterizando o tráfego HTTP por ele tratado. Da análise preliminar, mais de 70% das requisições se referiam a URLs dinâmicas, outro ponto interessante levantado foi o percentual de volume consumido com *YouTube*, cerca de 32% de todo o tráfego *web* da RedeUnB. Apesar de possuírem taxas significativas, os objetos referenciados pelas URLs dinâmicas, tiveram apenas 10% de acertos na taxa de *hits* por requisição e apenas 2% por volume, valores considerados relativamente baixos.

Visando identificar os motivos de taxas tão baixas, analisou-se com maior profundidade as URLs do serviço que teve o maior consumo individual, o *YouTube*. Dessa análise constatou que a maioria das URLs eram compostas por diversos parâmetros, inclusive de usuários, sendo consideradas distintas e dinâmicas. Ao pesquisar na literatura e em projetos de códigos abertos propostas para a problemática, identificou-se na literatura uma proposta de reescrita de URLs dinâmicas para estáticas, o projeto *Projekt YouTube*, apoiado pela Universidade *Lausitz* na Alemanha, tendo como proposta a interceptação, decodificação e sumarização de URLs dinâmicas, com foco no *YouTube*. Para avaliação

da proposta foram elencadas 5 (cinco) métricas, sendo elas: (i)*hits* por requisição, (ii)*hits* por volume, (iii)*miss* por requisição, (iv)*miss* por volume e (v)tempo de TCP.

Para um maior entendimento do funcionamento da proposta e a fim de avaliar sua eficácia foi feito uma implementação básica, com resultados satisfatórios. Posteriormente, analisou-se sua eficiência com testes mais complexos com variabilidade dos parâmetros de qualidade de vídeos e com utilização de três navegadores, obtendo-se resultados bem promissores. Objetivando reproduzir o comportamento dos usuários, foi utilizada como metodologia a escolha dos vídeos mais acessados a partir de redes brasileiras, em todas qualidades disponibilizadas pelo *YouTube*, utilizando-se os três navegadores mais populares do mundo: *Chrome*, *Firefox* e *Internet Explorer*.

Como colocado anteriormente, os resultados foram bem promissores alcançando taxa de acertos de *hits* acima de 99% em relação ao percentual de requisições e volume, e tendo um tempo médio de TCP abaixo de um sistema convencional de *proxy web cache* institucional. Identificou-se também que a variabilidade dos parâmetros de qualidade de vídeo e de tipo de navegadores não influenciaram nas métricas de *hits* e *miss*, ao contrário da métrica de tempo TCP. O navegador que apresentou o melhor desempenho durante os testes foi o *Internet Explorer*, pois contou com o menor tempo de TCP em relação aos demais.

A pesquisa aponta a viabilidade, alcançada em testes de laboratório, do uso da proposta do projeto *Projekt YouTube* no uso de *proxy web cache* institucionais para o tratamento de URLs dinâmicas, aumentado assim sua eficiência e desempenho, de forma a salvaguardar a banda da instituição, corroborando assim na melhoria da qualidade de experiência de navegação dos usuários. Sinalizando de forma positiva para a implementação no ambiente de produção da Universidade de Brasília. Contudo, em função do escopo e de questões operacionais não foram abordados no trabalhos alguns tópicos interessantes, ficando para trabalhos futuros. Seria de interesse verificar o comportamento da proposta sugerida em diferentes sistemas operacionais, inclusive em dispositivos móveis, identificando suas diferenças. Outro ponto a ser amplicado é testar a solução em ambiente

operacional e verificando sua eficácia e desempenho. Este estudo denotou a possibilidade de economia de banda. No entanto, dado a constante evolução serviço fornecido pelo *YouTube*, é necessário que haja atualização e monitoramento constante destas ferramentas de forma a garantir seu bom desempenho.

Referências

[1] Miniwatts Marketing Group. World internet usage and population statistics - 2012 q2. Disponível em: <http://www.internetworldstats.com/stats.htm>, Acessado em: 10 de Fevereiro de 2014. 1

[2] The Squid Software Foundation. Squid: Optimising web delivery. Disponível em: <http://www.squid-cache.org/>, Acessado em: 30 de Agosto de 2013. 1, 15, 41

[3] Internet Engineering Task Force (IETF). Rfc 2616 - hypertext transfer protocol http/1.1. Disponível em: <http://www.rfc-base.org/txt/rfc-2616.txt>, Acessado em: 10 de Janeiro de 2014. 1, 16, 17

[4] A. S. TANENBAUM. Redes de computadores, tradução da 4a edição, rio de janeiro: Campus, 2003. tanenbaum. 1, 2, 10, 13, 14, 15, 16, 36

[5] K. Hosanagar, R. Krishnan, M. Smith, and J. Chuang. Optimal pricing of content delivery network (cdn) services. *Proceedings of the International ...*, pages 10 pp.–, Jan 2004. 1, 25, 27, 28

[6] CAPES/MEC. Portal de periódicos da capes. Disponível em: <http://www.periodicos.capes.gov.br>, Acessado em: 05 de Janeiro de 2014. 7

[7] Mehregan Mahdavi and John Shepherd. Enabling dynamic content caching in web portals. In *Proceedings of the IEEE International Workshop on Research Issues in Data Engineering*, volume 14, pages 129–136, 2004. 7

[8] Wen Syan Li, Oliver Po, Wang Pin Hsiung, K. Seluk Candan, and Divyakant Agrawal. Freshness-driven adaptive caching for dynamic content Web sites. *Data and Knowledge Engineering*, 47:269–296, 2003. 7

[9] Anindya Datta, K Dutta, and H Thomas. Proxy-based acceleration of dynamically generated content on the world wide web: An approach and implementation. *ACM Transactions on ...*, 29(2):403–443, 2004. 7

[10] W Shi, R Wright, and E Collins. Workload characterization of a personalized web site and its implications for dynamic content caching. *Web Caching and Content*, 29:14–19, 2002. 7

[11] Gokul Soundararajan and Cristiana Amza. Using semantic information to improve transparent query caching for dynamic content web sites. In *Proceedings - International Workshop on Data Engineering Issues in E-Commerce, DEEC 2005*, volume 2005, pages 132–138, 2005. 7

[12] K. Psounis. Class-based delta-encoding: a scalable scheme for caching dynamic web content. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 799–805, 2002. 7

[13] Wenzhong Chen, P. Martin, and H. Hassanein. Differentiated caching of dynamic content using effective page classification. In *Performance, Computing, and Communications, 2004 IEEE International Conference on*, pages 293–298, 2004. 7

[14] Wenzhong Chen, P. Martin, and H.S. Hassanein. Caching dynamic content on the web. In *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, volume 2, pages 947–950 vol.2, May 2003. 7

[15] Internet Engineering Task Force (IETF). Rfc-3229 - delta encoding in http. Disponível em: <http://www.rfc-base.org/txt/rfc-3229.txt>, Acessado em: 24 de Abril de 2014. 7

[16] O.O. Abiona, T. Anjali, C.E. Onime, and L.O. Kehinde. Proxy server experiment and the changing nature of the web. In *Electro/Information Technology, 2008. EIT 2008. IEEE International Conference on*, pages 242–245, May 2008. 8

[17] L. Braun, a. Klein, G. Carle, H. Reiser, and J. Eisl. Analyzing caching benefits for YouTube traffic in edge networks — A measurement-based evaluation. *2012 IEEE Network Operations and Management Symposium*, pages 311–318, April 2012. 8, 24, 25, 28

[18] Ping-Jer Yeh, Jie-Tsung Li, and Shyan-Ming Yuan. Tracking the changes of dynamic web pages in the existence of url rewriting. In *Proceedings of the Fifth Australasian Conference on Data Mining and Analystics - Volume 61*, AusDM '06, pages 169–176, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc. 8, 48, 49

[19] Internet Engineering Task Force (IETF). Rfc 791 - internet protocol. Disponível em: <http://www.rfc-base.org/txt/rfc-791.txt>, Acessado em: 10 de Janeiro de 2014. 11

[20] Internet Engineering Task Force (IETF). Rfc 793 - transmission control protocol. Disponível em: <http://www.rfc-base.org/txt/rfc-793.txt>, Acessado em: 10 de Janeiro de 2014. 11

[21] Internet Engineering Task Force (IETF). Rfc 768 - user datagram protocol. Disponível em: <http://www.rfc-base.org/txt/rfc-768.txt>, Acessado em: 10 de Janeiro de 2014. 11

[22] Internet Engineering Task Force (IETF). Rfc 2821 - simple mail transfer protocol. Disponível em: <http://www.rfc-base.org/txt/rfc-rfc2821.txt>, Acessado em: 10 de Janeiro de 2014. 12

[23] Internet Engineering Task Force (IETF). Rfc 783 -file transfer protocol. Disponível em: <http://www.rfc-base.org/txt/rfc-783.txt>, Acessado em: 10 de Janeiro de 2014. 12

[24] Internet Engineering Task Force (IETF). Rfc 1035 - domain name system. Disponível em: <http://www.rfc-base.org/txt/rfc-rfc1035.txt>, Acessado em: 10 de Janeiro de 2014. 12

[25] Internet Engineering Task Force (IETF). Rfc 3040 - internet web replication and caching taxonomy. Disponível em: <http://www.rfc-base.org/txt/rfc-3040.txt>, Acessado em: 10 de Janeiro de 2014. 13

[26] Cisco Systems Inc. About cisco. Disponível em: <http://www.cisco.com/web/about/index.html>, Acessado em: 20 de Janeiro de 2014. 13

[27] Netcraft Inc. June 2013 web server survey. Disponível em: <http://news.netcraft.com/archives/2013/06/06/june-2013-web-server-survey-3.html>, Acessado em: 20 de Janeiro de 2014. 14

[28] Q-Success Inc. Web servers market position report. Disponível em: <http://w3techs.com/technologies/market/web_server/10>, Acessado em: 5 de Janeiro de 2014. 14

[29] Internet Engineering Task Force (IETF). Rfc 1945 - hypertext transfer protocol http. Disponível em: <http://www.rfc-base.org/txt/rfc-rfc1945.txt>, Acessado em: 25 de Fevereiro de 2014. 16

[30] Internet Engineering Task Force (IETF). Rfc 3986 - uniform resource identifier. Disponível em: <http://www.rfc-base.org/txt/rfc-3986.txt>, Acessado em: 10 de Janeiro de 2014. 16

[31] Shudong Jin and A. Bestavros. Popularity-aware greedy dual-size web proxy caching algorithms. In *Distributed Computing Systems, 2000. Proceedings. 20th International Conference on*, pages 254–261, 2000. 22

[32] L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. *Networking, IEEE/ACM Transactions on*, 8(2):158–170, Apr 2000. 22

[33] K.-Y. Wong. Web cache replacement policies: a pragmatic approach. *Network, IEEE*, 20(1):28–34, Jan 2006. 22

[34] A Finamore, M Mellia, and MM Munafò. Youtube everywhere: Impact of device and infrastructure synergies on user experience. *Proceedings of the 2011 . . .*, 2011. 23, 24

[35] Alexa Internet Inc. About alexa. Disponível em: <http://www.alexa.com/about>, Acessado em: 20 Janeiro de 2014. 24

[36] Google Inc. Estatística do youtube. Disponível em: <https://www.youtube.com/yt/press/pt-BR/statistics.html>, Acessado em: 12 de Março de 2014. 24

[37] Louis Plissonneau and Ernst Biersack. A longitudinal view of http video streaming performance. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 203–214, New York, NY, USA, 2012. ACM. 24

[38] Ashwin Rao, Arnaud Legout, Yeon-sup Lim, Don Towsley, Chadi Barakat, and Walid Dabbous. Network characteristics of video streaming traffic. *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies on - CoNEXT '11*, pages 1–12, 2011. 24

[39] Pablo Ameigeiras, Juan J. Ramos-Munoz, Jorge Navarro-Ortiz, and J.M. Lopez-Soler. Analysis and modelling of youtube traffic. *Transactions on Emerging Telecommunications Technologies*, 23(4):360–377, 2012. 25, 29

[40] Google Inc. Formatos de arquivos compatíveis com o youtube. Disponível em: <https://support.google.com/youtube/troubleshooter/2888402?hl=pt-BR&ref_topic=2888648>, Acessado em: 8 de Março de 2014. 25

[41] E. Hoffert, M. Krueger, L. Mighdoll, M. Mills, J. Cohen, D. Camplejohn, B. Leak, J. Batson, D. Van Brink, D. Blackketter, M. Arent, R. Williams, C. Thorman, M. Yawitz, K. Doyle, and S. Callahan. Quicktime: an extensible standard for digital multimedia. In *Compcon Spring '92. Thirty-Seventh IEEE Computer Society International Conference, Digest of Papers.*, pages 15–20, Feb 1992. 25

[42] Chu-Hsing Lin, Jung-Chun Liu, and Chun-Wei Liao. Energy analysis of multimedia video decoding on mobile handheld devices. In *Multimedia and Ubiquitous Engineering, 2007. MUE '07. International Conference on*, pages 120–125, April 2007. 25

[43] D. Omerasevic, N. Behlilovic, S. Mrdovic, and A. Sarajlic. Comparing randomness on various video and audio media file types. In *Telecommunications Forum (TELFOR), 2013 21st*, pages 381–384, Nov 2013. 25

[44] I. Bouazizi, K. Jarvinen, and M.M. Hannuksela. 3gpp mobile multimedia services [standards in a nutshell]. *Signal Processing Magazine, IEEE*, 27(5):125–130, Sept 2010. 25

[45] The WebM Project. Webm: an open web media project. Disponível em: <http://www.webmproject.org/>, Acessado em: 22 de Abril de 2014. 25

[46] Google Inc. Youtube player api reference for iframe embeds. Disponível em: <https://developers.google.com/youtube/iframe_api_reference>, Acessado em: 10 de Abril de 2014. 26, 27, 57

[47] Google Inc. Parâmetros do player incorporado do youtube. Disponível em: <https://developers.google.com/youtube/player_parameters>, Acessado em: 25 de Novembro de 2013. 26

[48] Google Inc. Qualidade do vídeo - youtube. Disponível em: <https://support.google.com/youtube/answer/91449?hl=pt-BR>, Acessado em: 20 de Setembro de 2013. 27

[49] R. Torres, A. Finamore, Jin Ryong Kim, M. Mellia, M.M. Munafo, and Sanjay Rao. Dissecting video server selection strategies in the youtube cdn. *Proceedings of the International . . .*, pages 248–257, June 2011. 28, 56

[50] W. Van Lancker, D. Van Deursen, E. Mannens, and R. Van De Walle. Http adaptive streaming with media fragment uris. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, pages 1–6, July 2011. 28, 29

[51] Central Intelligence Agency United Estates of America. Internet host of world - the world factbook. Disponível em: <https://www.cia.gov/library/publications/the-world-factbook/rankorder/2184rank.html>, Acessado em: 10 de Julho de 2013. 32

[52] REDECOMEP-DF. O que é gigacandanga ? Disponível em: <http://gigacandanga.net.br/index.php/gigacandanga/>, Acessado em: 20 de Julho de 2013. 33

[53] REDECOMEP-DF. Instituições participantes - gigacandanga. Disponível em: <http://gigacandanga.net.br/index.php/acessos-externos>, Acessado em: 05 de Maio de 2014. 33

[54] Nagios overview. Disponível em: <http://www.nagios.org/about/overview>, Acessado em: 30 de Agosto de 2013. 35

[55] Ntop – network top an overview. Disponível em: <http://www.ntop.org/wp-content/uploads/2011/09/ntop-overview.pdf>, Acessado em: 30 de Agosto 2013. 35, 36

[56] Internet Engineering Task Force (IETF). Rfc-1157 - a simple network management protocol (snmp). Disponível em: <http://www.ietf.org/rfc/rfc1157.txt>, Acessado em: 24 de Abril de 2014. 35

[57] C. Issariyapat, P. Pongpaibool, S. Mongkolluksame, and K. Meesublak. Using nagios as a groundwork for developing a better network monitoring system. In *Technology Management for Emerging Technologies (PICMET), 2012 Proceedings of PICMET '12:*, pages 2771–2777, July 2012. 35

[58] L. Deri and S. Suin. Effective traffic measurement using ntop. *Communications Magazine, IEEE*, 38(5):138–143, May 2000. 36

[59] Cord Beermann. Calamaris home page. Disponível em: <http://cord.de/calamaris-home-page>, Acessado em: 20 de Setembro de 2013. 41, 42

[60] J. Dilley and M. Arlitt. Improving proxy cache performance: analysis of three replacement policies. *Internet Computing, IEEE*, 3(6):44–50, Nov 1999. 41

[61] Varangpa Suranuntakul and Chutimet Srinilta. PP Caching: Proxy Caching Mechanism for YouTube Videos in Campus Network. *Proceedings of the International . . .*, I(0):4–8, 2011. 48

[62] Josilene A Moreira, Márcio Neves, Victor Souza, and Djamel Sadok. Estratégias de Cache para a Redução do Consumo de Banda e dos Atrasos na Distribuição de Vídeo Sob-Demanda na Internet. 2011. 48

[63] André. Yt-cache: youtube cache project. Disponível em: <https://code.google.com/p/yt-cache/>, Acessado em: 15 de Novembro de 2013. 49

[64] Pinheiro Luciano. incomum. Disponível em: <http://sourceforge.net/projects/incomum/>, Acessado em: 10 de Novembro de 2013. 49

[65] Universidade Lausitz. Projekt youtube. Disponível em: <http://www2.fh-lausitz.de/launic/comp/misc/squid/projekt_youtube/>, Acessado em: 10 de Novembro de 2013. 49, 50

[66] André L. Dos Santos. Youtube-cache - cache youtube videos. Disponível em: <https://code.google.com/p/youtube-cache/>, Acessado em: 20 de Abril de 2014. 49

[67] yt-cache YouTube Cache Project. Project committed changes. Disponível em: <https://code.google.com/p/yt-cache/source/list>, Acessado em: 20 de Abril de 2014. 50

[68] L. Pinheiro. Incomum - mailing lists. Disponível em: <http://sourceforge.net/p/incomum/mailman/incomum-users/?viewmonth=201311>, Acessado em: 20 de Abril de 2014. 50, 56

[69] S. Jahanzaib. Howto cache youtube with squid / lusca. Disponível em: <http://aacable.wordpress.com/2014/04/21/howto-cache-youtube-with-squid-lusca-and-bypass-cached-videos-from-\mikrotik-queue/>, Acessado em: 22 de Abril de 2014. 50

[70] The Squid Software Foundation. Squid caching dynamic content - mailing lists. Disponível em: <http://squid-web-proxy-cache.1019090.n4.nabble.com/squid-caching-dynamic-content-td4665779.htmla4665787>, Acessado em: 8 de Abril de 2014. 56

[71] Google Inc. Os vídeos mais acessados do youtube desde 2007 a partir do brasil. Disponível em: <http://gdata.youtube.com/feeds/api/standardfeeds/BR/top_rated?time=all_time&v=2>, Acessado em: 10 de Abril de 2014. 58

[72] Google Inc. Os vídeos mais acessados do youtube no mês atual a partir do brasil. Disponível em: <http://gdata.youtube.com/feeds/api/standardfeeds/BR/top_rated?time=this_month&v=2>, Acessado em: 10 de Abril de 2014. 58

[73] Netmarketshare Market Share Statistics for Internet Technologies. Market share reports. Disponível em: <http://www.netmarketshare.com/>, Acessado em: 25 de Abril de 2014. 58

[74] Web Analytics Clicky. Clicky web analytics - marketshare web browser. Disponível em: <http://clicky.com/marketshare/global/web-browsers/>, Acessado em: 10 de Abril de 2014. 58

[75] Awio Inc. W3counter - marketshare web browser. Disponível em: <http://www.w3counter.com/globalstats.php>, Acessado em: 10 de Abril de 2014. 58

[76] StatCounter Inc. Stat counter global stats - marketshare web browser. Disponível em: <http://gs.statcounter.com/>, Acessado em: 10 de Abril de 2014. 58