ACELERAÇÃO NO ACESSO À INTERNET: estudo sobre o servidor *proxy*/cache *Squid* [1]

You Chwen Yeu[2]

Gabriel de Souza Fedel[3]

RESUMO

Com a praticidade dos recursos e serviços oferecidos pela Internet, de forma simples e rápida, houve um aumento considerável no número de usuários que se conectam à rede, impulsionado pela sua popularidade e aceitação pelo público em geral. Tal fato tem gerado impactos perceptíveis, com a diminuição do desempenho nos acessos à Internet. Em vista deste cenário, como objetivo do trabalho, propõe-se o estudo sobre servidores *proxy* cache, mais especificamente o *software* livre *Squid*, que realiza recursos de Cache *Web* a partir dos objetos solicitados e acessados pelos usuários, localizados em âmbito rede local (LAN). Com foco nos benefícios para melhora de desempenho e redução do tempo necessário para acesso a páginas e conteúdos *web*, ao final deste trabalho, é realizado um projeto de estudo prático utilizando o *Squid*, sendo proposto a análise de dois cenários diferentes, a primeira com o recurso de cache do *Squid* desligado (cache *off*) e a segunda com este ligado (cache *on*). Através dos testes práticos pôde-se compreender o funcionamento de um servidor *proxy* cache, em específico o *Squid* além de perceber, analisar e entender os diferentes resultados obtidos habilitando-se ou não o recurso de *Web caching* do *proxy*.

Palavras-Chave**:** Squid; *Proxy*; Cache Web.

ABSTRACT

With the convenience of the services and resources offered by the Internet, in a simple and quickly way, there was a considerable increase in the number of users who connect to the network, driven by its popularity and acceptance by the general public. This fact has generated noticeable impacts, with the decrease in performance when accessing the Internet. Considering this scenario, the objective of this paper is to propose a study of proxy/cache servers, more specifically the Squid free software, which performs Web Cache features from the requested objects by users, located in the local network (LAN). Focusing on the benefits to improve performance and reduce the time needed to access web pages and content, by the end of this research, we conducted a case study project using the Squid, being proposed analysis of two different scenarios, the first with the Squid cache feature turned off (cache off) and the second with the tool enabled (cache on). Practical tests allowed us to understand the functioning of a proxy/cache server, Squid in particular, in addition to observing, analyzing and understanding the different results obtained by enabling or not the proxy Web Caching feature.

*Keywords**: Squid; Proxy; Web Cache.***

1. INTRODUÇÃO

A Internet trouxe consigo uma enorme gama de recursos, benefícios e facilidades para o usuário, assim sua popularidade e aceitação pelo público têm crescido e aumentado significativamente, impactando com isso um salto considerável em relação à geração de tráfego de dados na rede mundial de computadores. Tais conteúdos são serviços como o *streaming* de vídeos (populares Netflix e YouTube), além de imagens, sons, softwares e aplicações de mais variadas plataformas – e a partir destes, formam o denominado WWW (*World Wide Web*) ou simplesmente Web, um sistema de documentos dispostos na Internet que permitem o acesso a informações apresentadas na forma de hipertexto (textos exibidos em formato digital, informações em formato de imagem, áudio entre outros, além do conteúdo principal da web: o texto). Neste contexto, o

[1] Artigo baseado em Trabalho de Conclusão de Curso (TCC) desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Segurança em Sistemas de Informação da Faculdade de Tecnologia de Americana, depositado no 1º semestre de 2013.

[2] Tecnólogo em Segurança em Sistema de Informação da Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza ; Contato: ycy.andre@hotmail.com

[3] Prof. Me. Fatec Americana – Graduação em Ciências de Computação, Mestrado em Ciência da Computação; Contato: gabrielfedel@gmail.com

número de usuários que se conectam à rede cresce de forma muita rápida e acentuada, sendo que atualmente mais de 2/3 do tráfego da Internet é gerado pela Web – segundo Watanabe (2000). Além disso, estudos recentes revelam que até 2015, o tráfego na Internet deve aumentar sete vezes em relação ao volume registrado em 2010, passando dos 183 mil petabytes anuais para 1,3 milhão de petabytes anuais em 2015, segundo estimativas feitas por Cottle (2011), e, desta vez, o grande propulsor será o vídeo.

Segundo Watanabe (2000), a respeito do cenário atual, afirma que o desempenho nos acessos à Internet tem caído consideravelmente, aumentando as latências e os tempos de resposta. Diante desta questão, é perceptível a necessidade de se minimizar tais consequências, sendo que alguns métodos adotados geralmente são a atualização dos recursos: servidores mais rápidos, a troca de *switches*, o aumento da banda, entre outros. No entanto, esta não seria a solução ideal, uma vez que não é economicamente viável, devido aos altos custos dos equipamentos – conforme o autor. Alternativas são o espelhamento de arquivos e a utilização de servidores de cache e *proxy* que realizam *caching* de conteúdos e objetos Web, que ajudam a agilizar e reduzir os tempos de espera pelo usuário. Tais métodos são os mais utilizados para minimizar os gargalos existentes na rede (WATANABE, 2000).

Este trabalho tem como objetivo fazer um estudo sobre servidores *proxy* com foco na melhora do desempenho do acesso à web, através do método denominado Cache de conteúdos Web – ou simplesmente Cache Web. Aspecto como aceleração no ambiente da Internet são o ponto relevante a ser tratado com atenção. Para tal objetivo o *software* escolhido é o *Squid*, software livre para implementação de serviços de cache e *proxy*, sendo considerado um dos *proxy* mais utilizados no universo da área de Tecnologia da Informação (TI) (*SQUID* SOFTWARE FOUNDATION, 2011).

O trabalho será dividido da seguinte maneira, o segundo capítulo apresenta uma revisão bibliográfica sobre os conceitos e definições fundamentais, também dos trabalhos relacionados. O terceiro capítulo destina-se às questões de desempenho *Web Caching*, enfatizando os requisitos de hardware como servidor *proxy* cache. No quarto capítulo, o foco é específico sobre o *Squid*, apresentando os recursos e funcionalidades que dispõe. Finalmente, apresenta-se um projeto de estudo prático utilizando o *Squid* e a conclusão do trabalho.

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo, são descritos os conceitos e as definições das ferramentas apresentadas neste trabalho, fazendo parte de forma direta ou indireta ao decorrer dos capítulos, além de uma pesquisa de trabalhos com temas correlatos ao do trabalho.

2.1. Redes locais

Redes locais, também chamadas de LAN (*Local Area Network*), são redes privadas existentes em um campus universitário ou edifício, em raios de até alguns quilômetros de extensão. As redes locais possuem três características que as diferenciam de outros tipos de redes: o tamanho, a tecnologia de transmissão e a topologia.

No aspecto do tamanho, são consideradas restritas por serem limitadas a ambientes internos, uma empresa de médio porte por exemplo. Sua tecnologia de transmissão consiste em um cabo, no qual todas as máquinas estão conectadas, podendo operar a velocidades que vão desde 10 Mbps a 10 Gbps – esta última para as LANs mais modernas. Amplamente utilizadas para conectar computadores e estações de trabalho, permitem realizar trocas de informações (por exemplo, dados, imagens, softwares) e o compartilhamento de recursos, como impressoras. Tratando da topologia, admitem diversos modelos, isto é, tipo de sistema de difusão, tais como uma rede de barramento (cabo linear) ou uma rede em anel (TANENBAUM, 2003; p. 18-19).

2.1.1. Latência e tempo de resposta

De acordo com Cheshire (2006) a latência pode ser explicada como o atraso de tempo entre o momento que um evento iniciou e o momento que os efeitos iniciam. A palavra deriva do fato que durante o período de latência, os efeitos do evento estão latentes, ou seja, potenciais ou não ainda observados. O significado de latência pode variar dependendo do domínio do problema.

Em se tratando de latência em redes de computadores, o modelo de medição mais usado é o de tempo de ida e volta, porque pode ser medida a partir de um único ponto, contabilizado a latência de ida mais o tempo decorrido desde o envio da resposta pelo receptor até o recebimento pelo emissor. Vale ressaltar que tal latência exclui o tempo que o receptor gasta processando o pacote.

Nesse contexto, existe o conhecido comando *ping*, disponível em vários sistemas operacionais, que usa o protocolo *Internet Control Message Protocol* (ICMP), e pode ser utilizado para medir a latência da rede (ida e volta). O *ping* é um meio bastante preciso para medir a latência pelo fato de ele não processar os

pacotes, ele apenas envia uma resposta quando recebe um pacote, tornando-o eficiente no quesito para testes (CHESHIRE; 2006).

2.1.2. Largura de banda

Segundo o CERT.br (2012), largura de banda é a quantidade de dados que pode ser transmitida em um canal de comunicação, em um determinado intervalo de tempo. É um aspecto crucial quando se trata de velocidade e desempenho de rede, pois está relacionada com o tráfego e fluidez do canal para transferência de conteúdos, além de impactar diretamente na experiência do usuário no uso da rede.

2.2. Cache de conteúdos Web

Cache Web (ou também *Web Caching*) consiste em armazenar os conteúdos das páginas Web tais como vídeos, imagens, arquivos estáticos, softwares, dados de aplicações web entre vários outros na memória ram e/ou nos discos rígidos tanto em máquinas clientes como em servidores de cache e *proxy* – conforme ilustrado (Figura 1), ajudando assim a diminuir significativamente o tempo médio de acesso às páginas e transferência de arquivos. Já que muitos deles são requisitados mais de uma vez, exceto no primeiro acesso, no qual o conteúdo é requisitado a um servidor externo que contêm a página ou arquivo solicitado pelo usuário (WATANABE, 2000).

2.2.1 Política de reposição cache

Ao realizar processos de web cache dos conteúdos (resultantes de requisições feitas pelos usuários), outro subconceito necessário é a Política de Reposição *(Replacement Policy)*, que segundo Arlitt, Friedrich e Jin (1998), é um algoritmo que determina qual objeto(s) deve ser despejado/removido do cache, a fim de criar espaço suficiente para adicionar um novo objeto, permitindo que outro conteúdo seja reposto no lugar. A partir deste método, aproveita-se melhor o cache dos servidores Web *proxy*, garantindo uma rotatividade dos objetos em disco, deixando mais ágil o processo para requisição de conteúdos. Em outras palavras, são algoritmos de expiração que o cache utiliza, a fim de eliminar documentos obsoletos, que se pode basear pelo tempo em que está armazenado, tamanho, histórico de acesso.

Algumas políticas de reposição utilizadas são: *Least recently used, Least frequently used, Size, GreedyDual-size e First in first out,* que serão apresentadas detalhadamente abaixo, conforme Pontes, Hirata e Honório (2008).

- *Least Recently Used (LRU)*: menos utilizado atualmente, pois mantém no cache os objetos referenciados recentemente, e remove o objeto referenciado há mais tempo (mais antigo). O problema deste algoritmo é que ele não leva em conta o tamanho do objeto, uma vez que vários objetos pequenos podem ser substituídos por um objeto grande;
- *Least Frequently Used (LFU)*: utilizado com menor frequência. Remove o objeto menos popular, porém não considera tempo do último acesso. Em vez disso, ele considera pelo número de acessos;
- *Size*: este é baseado no tamanho do objeto, substitui primeiro objetos com tamanhos maiores;
- *GreedyDual-Size (GDS)*: atribui valores baseados no custo de um hit rate[4] para os objetos armazenados no cache. O custo pode ser latência ou pacotes transmitidos pela rede. São mantidos em cache objeto menores, referenciados mais vezes; e,
- *First in First* out (FIFO): primeiro a entrar é o primeiro a sair, isto é, objetos são removidos na mesma ordem que entram. Não é muito usado em servidores cache.

Segundo trabalho de Arlitt, Friedrich e Jin (1998), foi realizado um estudo que aborda a questão de como são feitas as escolhas e decisões do que será despejado e do que será reposto/substituído no cache, através das políticas de resposição citadas acima. Após a simulação do ambiente de testes e realizar as avaliações de desempenho, os autores obtiveram como resultado o seguinte: eles concluem que algoritmos que operam tendo como base o tamanho de objetos, beneficiam o *request hit rate*, enquanto que algoritmos que funcionam como base a frequência com que o conteúdo é acessado, beneficia o *byte hit rate*.

2.3. Servidor *proxy cache*

A principal função de um servidor *proxy*/cache (Figura 1) é atuar como intermediário entre um cliente qualquer e o servidor de destino onde contém o conteúdo, objeto ou serviço requisitado pelo usuário.

Segundo definição de Pinheiro (2006):

> "Um servidor *proxy* funciona como um intermediário no contato dos computadores da rede local com outras máquinas fora dela, como por exemplo na Internet. Ele recebe as requisições de acesso externo dos *hosts*

[4] ***Hit rate*:** taxa de acerto do cache, porcentagem de sucesso em que um determinado conteúdo é obtido a partir do cache ao invés de acessar o conteúdo original, num servidor externo. Disponível em: <http://www.marcosmonteiro.com.br/mm/Cursos/Organizacao_de_Computadores/CACHE.pdf>. Acesso em 5 set. 2013.

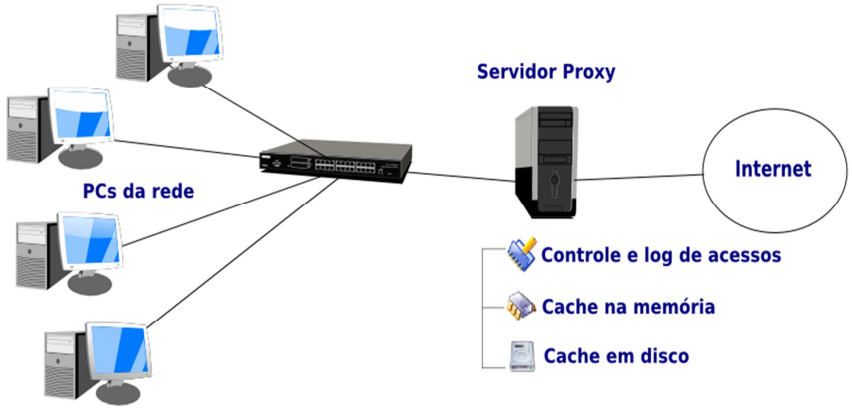
> locais e as repassa a outros computadores fora da rede local, retornando as respostas aos computadores que as solicitaram."

Com isso permite agilizar o processo de acesso às páginas da Internet, reduzindo assim o tempo médio de espera para acessar um determinado conteúdo, através do método conhecido como cache de conteúdos Web, citado anteriormente.

E sobre a visão de Zanoni (2007):

> "O objetivo principal de um servidor *proxy* é possibilitar que máquinas de uma rede privada possam acessar uma rede pública, como a Internet, sem que para isto tenham uma ligação direta com esta. O servidor *proxy* costuma ser instalado em uma máquina que tenha acesso direto à Internet, sendo que as demais efetuam as solicitações através desta. Justamente por isto este tipo é chamado de *proxy,* pois é um procurador, ou seja, sistema que faz solicitações em nome de outros."

Figura 1: Funcionamento de um servidor *proxy* cache



Fonte: http://www.hardware.com.br/

De acordo com Pontes, Hirata e Honório (2008), o *proxy/cache* colabora com três dos maiores desafios dos clientes, sendo eles:

- Aceleração de aplicações e conteúdo Web: o *proxy/cache* reduz a latência, utilização de banda e sobrecarga nos servidores, aumentando a distribuição do conteúdo Web e aplicações baseadas em Web como sistemas de CRM (*Customer Relationship Management*) e ERP (*Enterprise Resource Planning*);
- Segurança na Internet: pode-se implementar processos de segurança, incluindo *proxy*, caching, controle de acesso, filtragem de conteúdo, anti-virus web, scanning SSL (*Secure Sockets Layer*), bloqueio de Instant message (msn messenger, AOL Instant Messenger) e P2P (*peer to peer conection*), *antispam* e relatórios; e,
- Distribuição de vídeo: O *proxy*/cache melhora a qualidade dos treinamentos online, vídeos executivos e serviços de vídeo sob demanda.

2.4. Modelos de *proxy*

O *proxy* possui diversas formas de implementação e operação, cada um possui características e objetivos específicos que melhor atendem aos requisitos dos clientes para um determinado do tipo de rede, empresas ou instituições de ensino, sendo eles:

Web *proxy*:

Também denominado de Web *proxy/cache*, é o tipo mais comum de *proxy* existente, realiza o *caching* de páginas web e arquivos que estão localizadas nos servidores Web externos, possibilitando que os usuários da rede local (LAN) tenham um acesso mais rápido e confiável aos conteúdos da Internet – por eles requisitados, através do armazenamento em cache dos recursos Web, tais como conteúdos estáticos como figuras e gráficos (PONTES; HIRATA; HONÓRIO, 2008).

***Proxy* reverso:**

Como o próprio nome diz, o *proxy* reverso funciona ao contrário do *proxy*. No modelo convencional de *proxy* se faz a interceptação das requisições dos clientes que estão localizados na rede local (LAN) com destino à Internet. Enquanto que no *proxy* reverso a operação é inversa, interceptando requisições vindas da Internet com destino à rede local. Em suma, ambos são responsáveis em garantir que o cliente não tenha uma conexão direta com o servidor que armazena o conteúdo requisitado, a fim de garantir o anonimato das estações clientes (SABOCINSKI NETO, 2009).

Conforme Silva (2011), os benefícios de utilizar *proxy* reverso os seguintes aspectos:

- Segurança: como o *proxy* é a única interface externa da rede, ele "esconde" os demais servidores;
- Criptografia: a criptografia SSL pode ser delegada ao *proxy* ao invés dos servidores internos;
- Balanceamento de carga: o servidor pode distribuir a carga para vários servidores da rede;
- Cache: assim como o web *proxy*, o p*roxy* reverso pode manter em cache o conteúdo estático das requisições realizadas, ajudando assim a diminuir a carga dos servidores web; e,
- Compressão: o p*roxy* reverso pode tornar o acesso mais rápido através da compressão do conteúdo acessado.

***Proxy* transparente:**

Segundo Pontes, Hirata e Honório (2008), conhecido também como trans*proxy*, é o tipo de *proxy* onde não é necessário fazer configurações adicionais nas estações clientes (nos navegadores dos computadores, por exemplo), visto que ele combina NAT (*Net Address Translation*) com o servidor *proxy*, facilitando assim o uso dele em instituições de ensino e empresas, reforçando as políticas de uso da rede. Em casos onde não se utiliza este método, numa configuração manual, por exemplo, há então a necessidade de especificar manualmente nos navegadores o endereço IP (*Internet Protocol*) do p*roxy* bem como a porta de operação.

Como benefícios de sua utilização, destaca-se o protocolo de roteamento de conteúdo *Web Cache Communication Protocol* (WCCP), desenvolvido pela Cisco Systems em 1997; criado com mecanismos de balanceamento de carga, redundância, tolerância a falhas e garantia de serviço *(failsafe)*. Presente nos roteadores Cisco, ele permite redirecionar o fluxo de tráfego de rede em tempo real.

Quando se utiliza roteadores com o protoloco WCCP implementado junto com o servidor *proxy* transparente, eles permitem atuar como intermediário entre a rede local e a internet, provendo uma distribuição inteligente de caches de rede para maximizar o desempenho de download e disponibilidade de conteúdo, reduzindo e otimizando assim a utilização do link de Internet (CISCO DOCWIKI, 2012).

2.5. Servidor *proxy Squid*

Licenciado nos termos da *General Public License (GPL)*, o *Squid* é um software livre que possui uma enorme gama de recursos quando se trata de gerenciamento *proxy*, graças a uma comunidade de usuários que contribuem com o desenvolvimento e aperfeiçoamento do mesmo – segundo *Squid* Software Foundation (2011).

Segundo Pinheiro (2006):

> "O *Squid* é um dos servidores *proxy* mais utilizados no mundo, dado a sua robustez, segurança e recursos que oferece. Apesar dos poucos protocolos que ele consegue trabalhar, no caso apenas o HTTP, HTTPS, FTP e gopher, é ainda uma alternativa muito interessante, já que estes são os principais protocolos da internet, e além do mais, muitos dos aplicativos que usam outros protocolos tem capacidade de usar o *Squid* através de um dos protocolos suportados por ele. O *proxy Squid* funciona ouvindo requisições numa determinada porta padrão, ou numa outra porta que pode ser configurada pelo administrador da rede.

Atualmente, o *Squid* vem sendo amplamente utilizado por empresas dos mais variados ramos e portes, indo desde simples casas domésticas até as complexas grandes corporações. E cada vez mais é implementado em arquiteturas de distribuição de conteúdo a fim de proporcionar a entrega de conteúdos dinâmicos, estáticos e *streaming* de vídeo/áudio para os milhões de usuários da rede mundial de computadores (a Internet) com melhor eficiência e racionalização do uso do canal de comunicação (*SQUID* SOFTWARE FOUNDATION, 2011).

Além de atuar simplesmente como intermediador para o acesso à Internet, o *Squid* possui muitos outros recursos plausíveis e excelentes que proporcionam melhor uso da rede e redução da utilização de banda de Internet. De acordo com Pinheiro (2006), dentre esses recursos, destacam-se:

- Cache - através desse recurso o *Squid* armazena em cache o conteúdo acessado, de forma que se algum host fizer novamente uma requisição ao mesmo conteúdo, que já se encontra armazenado, ele recebe diretamente do cache, sem a necessidade de efetuar uma nova busca dos dados na Internet. O uso desse recurso pode trazer rapidez maior ao acesso à Internet, pois provavelmente o link do host com o *proxy* é bem mais rápido do que deste com a Internet;

- Autenticação - podemos restringir o acesso ao servidor *proxy* com o uso da autenticação de usuários, de forma que seja melhorada a segurança, já que somente usuários autorizados poderão acessar a Internet. Este recurso é bastante flexível e pode ser implementado de várias maneiras, como uso do protocolo LDAP, SMB, módulos PAM;
- Registro de acessos - os acessos são registrados em arquivos de log, podendo esses serem utilizados para as mais diversas finalidades, que vão desde a análise de performance do servidor, até a geração de relatórios detalhados dos acessos à Internet. Existem vários softwares analisadores de logs do *Squid* capazes de gerar relatórios tão bons, que por si já justificariam o uso do *Squid*, em razão do controle proporcionado;
- Controle centralizado - com o uso do *proxy* temos a facilidade de um único ponto centralizador do acesso à Internet, o que torna a gerência da rede mais fácil e eficiente. Uma única máquina é capaz de prover acesso a várias outras; e.
- Segurança - como apenas o *proxy* está diretamente ligado à Internet, temos apenas uma (ou mesmo poucas, caso tenhamos mais de um servidor *proxy*) máquina potencialmente vulnerável. Desta forma fica mais fácil concentrar esforços na melhoria da segurança de apenas um ponto na rede.

2.6 Trabalhos relacionados

Foram feitas pesquisas a respeito de trabalhos relacionados que abordam servidores *proxy* cache, sob aspectos diferentes, em outra óptica/abordagem no que diz respeito à metodologia e ao objetivo do trabalho. Caso de temas de autores como: Fernandes e Serrão Junior (2010); Pontes, Hirata e Honório (2008) e também de Silva (2009).

A praticidade, comodidade e facilidade que o uso da Web dispõe, gera uma grande popularidade no acesso dos mais disversos tipos como comércio eletrônico *(e-commerce)*, portais de vídeo e áudio, sites de relacionamento como redes sociais e bate papo, além de *websites* sobre notícias e novidades das mais diversas áreas de conhecimento e gosto. Em vista deste cenário, depara-se com a necessidade de controlar o acesso aos conteúdos ofertados pela Internet, por exemplo facebook, youtube, páginas pornográficas, twitter entre outros, que dividem a atenção do funcionário das instituições corporativas (ou até mesmo alunos das redes de ensino) durante as jornadas de trabalho, uma vez que gera um impacto direto no rendimento e na capacidade de produção de trabalhos. Segundo Fernandes e Serrão Junior (2010), este fato pode ser solucionado com a implementação de servidores de *proxy* e cache dentro dos perímetros da empresa para administrar, controlar e filtrar todo e qualquer acesso à Internet, o que garante maior segurança, privacidade e integridade dos dados e informações da mesma.

Benefícios adicionais resultantes da implementação do *proxy* cache, segundo Pontes, Hirata e Honório (2008) que abordam em seu trabalho, é a possível diminuição da latência no acesso à Internet (e de suas aplicações), através do método conhecido como Cache Web, onde armazena os dados e conteúdos Web previamente consultados e os torna disponível para uso posterior por um ou mais usuários, otimizando o uso de banda e tornando mais rápido o acesso à rede.

Sob outra óptica a respeito de servidores *proxy* cache, no trabalho de Silva (2009), um comparativo de performance entre as soluções *Squid* e Varnish é colocado em estudo, enfatizando diferenças e semelhanças, dadas a partir de um conjunto de testes, submetendo os dois aceleradores HTTP em prática/simulação. Apesar das diferenças, isto é, o *design* diferente que cada um implementa, concluiu-se a performance de ambos são tecnicamente equivalentes.

Dos trabalhos que abordam servidores *proxy* cache, o que está mais relacionado com o deste projeto é o de Silva (2010), que aborda os métodos para implantação de um *proxy*/cache *Squid*, com o principal objetivo de acelerar o acesso e diminuir a utilização de banda da Internet, além do controle de conteúdo. O aspecto que difere o trabalho deste com o do presente trabalho se dá devido ao estudo de caso. Pois neste o foco é estudar os cenários com ou sem a utilização do *Squid proxy* cache, enfatizando diferenças de performance entre esses dois ambientes e os resultados dos testes nele obtidos, a partir de metodologia do *Squid* com o cache ligado ou desligado, ou seja, realizando cache de conteúdos Web ou não, respectivamente.

3. QUESTÕES DE DESEMPENHO DE WEB CACHING

Atualmente, os servidores *proxy* ganharam uma nova visão, uma nova perspectiva, ocupando uma posição muito importante dentro das empresas, sendo considerados uma necessidade, visto que um servidor *proxy* permite ao administrador adicionar um serviço de Caching para Web, controlar o acesso à rede além de permitir o armazenamento de consultas DNS (STANGER; LANE; DANIELYAN, 2002).

Assim, a qualidade no acesso a páginas, conteúdos e recursos Web é questão que está ligada diretamente ao desempenho dos servidores de *proxy*/cache, mais especificamente o serviço de Web Caching realizado pelo *proxy*. Como vertentes do assunto, destaca-se os métodos e práticas correlacionadas a este

processo de Caching, no qual temos as políticas de reposição cache, que mostram os diferentes tipos de algoritmos que podem ser utilizados para determinar a limpeza do cache, a fim de garantir um fluxo harmonioso dos objetos em cache.

Essas políticas, por sua vez, norteiam o funcionamento do *proxy*/cache, pois este opera de acordo com o tipo de política que é programado e configurado nele, gerando assim um impacto direto na rede, dependendo do algoritmo escolhido.

A seguir serão abordadas questões relacionadas ao desempenho, sobre o que influência e o que impacta, na visão e perspectiva dos requisitos de hardware para performance de um servidor *proxy*/cache.

3.1 Requisitos de hardware

Sistemas de Cache Web *(caching)* certamente exigem muito do desempenho dos subsistemas de hardware, estressando-os ao ponto máximo. A chave para o bom desempenho cache está ligada diretamente à performance geral do sistema *(overall system performance)*, tendo maior impacto os seguintes aspectos, listado por ordem decrescente de importância – do mais crítico ao menos crítico:

- Tempo de busca aleatória (disco rígido);
- Quantidade de memória RAM do sistema;
- Taxa de transferência do disco *(throughput); e,*
- Poder de processamento da CPU, de acordo com Pearson (2012).

Ressalta-se que ao reduzir ou comprometer drasticamente a performance de qualquer um dos subsistemas de hardware acima mencionados, o desempenho será prejudicado.

É certo que para determinar o hardware necessário e suficiente (potência do cache) é uma tarefa bastante difícil, então a coleta de estatísticas sobre o uso da Internet do local em questão é algo para ser feito, segundo Pearson (2012).

Segundo o autor, para decidir a potência do cache, muitos fatores precisam ser levados em conta. A princípio, deve-se ter uma ideia, uma base de quanto será a carga que o servidor *proxy*/cache irá sustentar, a partir do número máximo de requisições por minuto. Esta por sua vez indica o número de objetos baixados em um minuto pelos clientes, o que pode ser utilizado para se ter uma ideia estimada do quanto será a carga do cache.

Em casos onde não se possuem muitos dados estatísticos sobre o uso da Internet, recorre-se então a instalação de máquinas testes (como montar um servidor de cache teste) a fim de estimar índices para as tomadas de decisão quanto ao hardware mais adepto e que atenda às necessidades da rede.

Nas seções seguintes, serão explorados itens relacionados aos requisitos de hardware como disco rígido, memória RAM e CPU (processador).

3.1.1. Disco rígido

Segundo Pearson (2012), o disco rígido *(Hard Disk – HD)* representa um componente que impacta em muito no desempenho quando se trata da execução dos processos de *caching*, uma vez que os objetos são armazenados nele. Conforme o autor, dois pontos importantes a serem levados em conta quando se pretende mensurá-los ou para adquir novos HDs são: a velocidade do disco e espaço em disco necessário.

- **Velocidade do disco:** para armazenamento eficaz de objetos no cache, não são necessários discos rápidos que retêm pequena quantia de dados, mas sim de discos que permitem armazenar uma grande quantidade de dados baixados e que sejam rápidos o suficiente para não retardar o cache em operações de busca/rastreamento.

Um dos pontos mais importantes a serem levados em conta quando se pretende utilizar o Cache em disco é o tempo de busca aleatória. Que é o tempo médio que as cabeças do disco demoram para passar de uma faixa aleatória para outra, medidas em milissegundos (ms), sendo melhor quando este valor é menor.

- **Espaço em disco necessário:** determinar o espaço em disco necessário para o cache (do *Squid* por exemplo) depende de vários fatores, o mais importante é prever em quanto tempo o espaço disponível alocado em disco irá encher – será preenchido pelos objetos Web, normalmente medidos em dias.

Para ter uma noção base, o método mais eficaz é se baseando na taxa de transferência máxima teórica da largura de banda da Internet, o link de Internet, por exemplo:

Para um link de 1mb/s (megabytes por segundo), transfere-se cerca de 125.000 bytes por segundo. Caso todos os clientes sejam configurados para acessarem o cache, o uso do disco seria de aproximadamente 125k por segundo, e que significa uma média de 450 megabytes por hora. Para um dia inteiro, então, o valor será cerca de 3,6 gigabytes (GB) transferidos por dia.

A partir disso, infere-se que para manter todos os dados para um dia, o HD deverá ter 3,6 GB de espaço disponível para o cache, a fim de conseguir armazenar todos os conteúdos cacheados.

Sobre a viabilidade do esquema de caching, se este é viável ou não, recorre-se a um valor mínimo, que exige a taxa de pelos menos dois ou mais usuários visitem a mesma página, dentro do período de tempo quando o objeto ainda estiver disponível no cache. Para os sites maiores, sendo mais populares e mais

acessados, esta meta pode ser facilmente alcançada, como os motores de busca e sites de *streaming* de vídeos, google e youtube respectivamente. No entanto, para os sites e páginas menores que são menos solicitados, o valor dificilmente será atingido.

Quanto ao tempo que os objetos (páginas) ficam no cache, deve-se ter como basear no número de usuários que as solicitam, com maior prioridade àqueles requisitados com maior frequência.

Diante de tais aspectos, a decisão de quanto espaço em disco alocar para o cache dependerá, prioritamente, da largura de banda disponível.

3.1.2 Memória RAM

Segundo Pearson (2012), os *proxy*/cache mantém na memória RAM uma tabela de objetos, devido ao fato deste realizar buscas e verificações sobre objetos armazenados no cache da RAM, com isso, um acesso rápido à tabela é muito importante e crucial.

Havendo partes da tabela situadas na SWAP (memória virtual, usada para cache de armazenamento secundário como paginação e *swapping*/troca), o desempenho sofre drásticas diminuições. Uma vez que o tamanho do processo *Squid* é grande, utilizar *swapping* pode ser algo ruim, pois afeta muito na fluidez do mesmo.

Para ter noção, cada objeto armazenado no disco ocupa cerca de 75 bytes de RAM para guardar o índice *(index)*.

O tamanho médio de um objeto da Internet é de cerca de 13kb (kilobytes), segundo (PEARSON, 2012). Então, se tiver disponível um gigabyte de espaço em disco, é provável que consiga armazenar 80.000 objetos aproximadamente.

Seguindo a lógica, para os 75 bytes de RAM necessários por objeto, 80.000 objetos requerem em torno de seis megabytes de RAM. E supondo que tenha 8 GB de espaço em disco, 48 MB de RAM serão necessárias apenas para guardar os índices de objeto (index), sem levar em conta (excluindo) a memória utilizada para o sistema operacional, o binário do *Squid*, além de programas, serviços e processos em execução (PEARSON, 2012).

3.1.3 CPU

Geralmente o *proxy*/cache não exerce uma carga intensa à CPU, somente durante o processo de inicialização do mesmo *(startup)*, quando utiliza boa parte do poder de seu processamento. O uso de uma CPU lenta retardaria o acesso ao cache durante os primeiros minutos pós inicialização.

Máquinas multiprocessadoras geralmente não proporcionam grande aumento de velocidade, apenas segmentam partes do código do programa, e não ajudam a reduzir os tempos de espera. O que realmente beneficia e ajuda são: mais memória (para cache de dados) e mais discos rígidos (PEARSON, 2012).

4. O SERVIDOR DE *PROXY* E CACHE *SQUID*

O *Squid* possui muitos recursos, configurações e otimizações de desempenho possíveis de serem feitas o que permite adaptá-lo melhor a este ou aquele ambiente, permitindo suprir as diferenças entre uma instituição de ensino e uma empresa, por exemplo. Cada órgão possui suas próprias políticas de uso da rede e dispõem de recursos computacionais diferentes à disposição, como o plano de Internet (velocidade e largura de banda) disponível para uso.

Para se ter ideia, com o *Squid* é possível impor restrições de acesso com base no horário, endereço IP da máquina e também por *login* (nome do usuário). Essa função é muito útil quando se pretende conceder acesso livre à Internet, em horários de descanso como no almoço. Em contrapartida, pode atuar para filtrar/bloquear acesso a conteúdos como vídeos, redes sociais (facebook, twitter, orkut), músicas, sites de chat online e pornografia, reduzindo distrações dos usuários em horários de trabalho/acadêmico, beneficiando a redução do uso da banda de Internet, obtendo melhor velocidade no acesso (SILVA, 2010).

Outras funções que completam o *Squid* são a possibilidade de registrar logs de acesso dos usuários em arquivos, graças ao trabalho conjunto com a ferramenta *Squid Analysis Report Generator* (SARG), permitindo conhecer o que acessam, quem acessou e em quais horários, o quê beneficia no incremento das listas de bloqueio (AGUADO, 2013).

Segundo Silva (2010), o *Squid* além de ser simplesmente um acelerador de acesso às páginas, tem o potencial de servir como um verdadeiro *cache* de arquivos, armazenando atualizações de softwares (*como o Windows Update*), *downloads* e até pacotes de instalação via gerenciador de pacotes APT do GNU/Linux (*apt-get*). Assim se um administrador baixar o navegador Firefox e uma imagem do sistema operacional Ubuntu para instalar em 20 máquinas do laboratório, com a execução do *download* pela primeira vez, o *Squid* já armazena uma cópia em seu cache, facilitando para as outras 19 máquinas restantes, pois não precisará requisitar novamente o conteúdo ao servidor original, e sim ao cache do *Squid*, sendo muito mais rápido o processo e economiza em muito a banda de Internet.

4.1 Protocolo ICP

Segundo Wessels e Claffy (1997), o protocolo *Internet Cache Protocol* (ICP*)* é um formato de mensagem usada para comunicação e troca de informações entre servidores *proxy*/cache sobre conteúdos armazenados em seu cache. É um recurso muito útil e eficiente em questões de flexibilidade no desempenho para navegação Web. Embora os *proxy*s (*Squid*) usem HTTP para transferência de dados de objeto, beneficiam melhor ao utilizar um protocolo de comunicação mais simples e versátil, caso do ICP.

Segundo os autores, o ICP é implementado principalmente em malhas de cache, isto é, em ambientes com um conjunto de vários servidores cache vizinhos a fim de localizar – a uma distância menor e mais rápida – objetos e conteúdos Web específicos cacheados. Um cache envia uma consulta ICP para seus vizinhos, e estes são responsáveis em devolver uma resposta ICP, indicando se possui ou não o objeto (*um "HIT" ou um "MISS"*).

4.2 Modo de aceleração

Recurso disponível no *Squid* que consiste em ajudar a acelerar um servidor Web lento, atuando à frente deste, realizando cache de objetos e conteúdos do servidor Web (páginas, sites) e o disponibiliza para acesso de ambientes externos (a usuários da Internet), através do método conhecido como Accelerator Mode – ou Modo de Aceleração.

Normalmente servidores de cache podem atuar como servidores Web e vice-versa. Esses servidores, por sua vez aceitam pedidos tanto no formato padrão de requisição Web (web-request), quando apenas o caminho e o nome é dado e no formato específico de *proxy,* onde a URL inteira é fornecida. Por padrão, o *Squid* não veêm configurado para operar dessa modo, devido à decisão tomada pelos desenvolvedores e designers, pois evita diversos problemas e reduz a complexidade do código, o que o deixa mais seguro, sólido e confiável, segundo Pearson (2012).

A respeito de quando usar o modo acelerador, o ideal é que não deve ser ativado a menos que precise. Um conjunto limitado de motivos e circunstâncias são impostos para que seja necessário seu uso, uma situação exemplo para tal uso seria a aceleração de um servidor lento.

Colocar o *Squid* à frente do servidor lento, permite o armazenamento em cache dos resultados do servidor e repassá-los aos clientes. Uma solução muito útil quando o servidor de origem esta com tráfego lento.

4.3 ACLs

Access Control Lists (ACLs) ou listas de controle de acesso, são as regras de acesso criadas, elaboradas, definidas e configuradas pelo administrador, com a finalidade de controlar e administrar a utilização da rede e o acesso à determinadas páginas e conteúdos considerados inapropriados. As ACLs constituem-se na grande flexibilidade e eficiência do *Squid*, pois são através delas que se pode criar regras para controlar o acesso à Internet – dos mais diferentes métodos possíveis. Praticamente todo o processo de controle do *Squid* é feito com o seu uso (PINHEIRO, 2006).

Ainda segundo o autor, o uso das listas de controle de acesso são a parte mais importante e crucial da configuração de um servidor *proxy Squid*. Uma vez bem configuradas as ACLs proporcionam um aumento do nível de segurança da rede como um todo, e caso estejam mal configuradas, o resultado obtido será praticamente o oposto, gerando falsa sensação de segurança, além de não extrair ao máximo dos recursos e funcionalidades que o mesmo oferece.

Antes de realizar e construir as regras de acesso, há dois pontos que se deve ter em mente, conforme Pinheiro (2006) o primeiro é que o *Squid* interpreta as ACLs de cima para baixo, portanto é de suma importância observar este aspecto ao elaborá-las e, segundo, as ACLs são *case-sensitive*, isto é diferenciam entre letras maiúsculas e minúsculas, exigindo um cuidado especial com isso. Para habilitar seu uso a opção **(a *flag*) –i** é necessária para que o recurso seja ativado, havendo assim a diferenciação entre caracteres maiúsculos e minúsculos.

Segundo Aguado (2013), em relação às ACLs, por padrão, as listas disponibilizadas pelo *Squid* permitem ser agrupadas em três tipos, sendo ACLs de origem, ACLs de destino e ACLs de horário:

1º) ACLs de origem são aquelas que controlam todo e qualquer acesso que tenha como origem algum determinado local. A origem normalmente consiste em endereço IP de máquina ou endereço da rede. Exemplo de uso para criar uma ACL chamada rede interna, tendo como origem (src) a rede 192.168.1.1/24:
acl rede_interna src 192.168.1.1/24;

2º) ACLs de destino, são as mais comuns e mais utilizadas, são aquelas onde se pretende criar padrões combinando com determinados endereços de rede, de domínio e expressões regulares. Exemplo de uso para se criar uma ACL chamada liberado_dominio, tendo como destino os domínios .globo.com.br e .terra.com.br: ACL liberado_dominio dstdomain .globo.com.br .terra.com.br ; e,

3º) ACLs de horário são muitos úteis pois permitem configurar e programar para liberar o acesso a determinados conteúdos, por exemplo, redes sociais e sites de entretenimento nos horários fora do expediente, horários de descanso ou almoço. Exemplo de uso para criar uma ACL chamada hora_almoco, tendo como intervalo de tempo o horário 12:30 às 13:30: ACL hora_almoco time 12:30 – 13:30.

Abaixo temos uma lista dos mais variados tipos de ACL que podem ser utilizados, sendo comumente os descritos abaixo, conforme a explicação de Pinheiro (2006):

src: faz parte das ACLs de origem, é um tipo utilizado para indicar endereços IP de origem. Por exemplo, para especificar uma determinada faixa ou endereço de rede, tal como: acl rede_adm src 192.168.16.0/24, pode ser também endereço de um computador específico, como acl pc1 src 192.168.16.10/24 ou uma faixa de endereços, como: acl redetotal 192.168.16.10-192.168.16.20/24;

dst: tipo ACL de destino. É semelhante ao tipo anterior, mas está relacionado ao endereço de destino, pode-se utilizá-lo para especificar um determinado endereço IP de destino;

***proxy*_auth**: uma opção muito útil quando se pretende implementar autenticação de usuários no *Squid proxy*. A autenticação é feita com uso de softwares externos. Podem ser passados os nomes dos usuários ou usada a opção REQUIRED para que seja autenticado qualquer usuário válido. Exemplo para criar uma ACL chamada user_passwords e especificar o arquivo que contêm os dados necessários para o processo de autenticação, localizado em /tmp/user_passwds: acl user_passwords *proxy*_auth /tmp/user_passwds;

srcdomain: tipo ACL de origem. Indicado para verificar o domínio da máquina cliente. Os domínios serão obtidos por resolução reversa de IP, o que pode causar atrasos para a resposta da requisição. A definição do domínio deve ser feita da seguinte forma: ".meudominio.com.br", não podendo ser esquecido o "." (ponto) no início. Por exemplo para verificar um determinado domínio de origem:
acl origemdom srcdomain .linuxdebian.br;

dstdomain: tipo ACL de destino. É usado da mesma forma que *srcdomain*, entretanto com relação ao destino. Exemplo para especificar um determinado domínio de destino:
acl destinodom srcdomain .linuxdebian.br;

srcdom_regex: tipo ACL de origem. Avalia o domínio usando expressões regulares. Seu uso é semelhante as duas anteriores, acrescentando a flexibilidade do uso da expressão regular;

dstdom_regex. tipo ACL de destino. Usado da mesma forma que *srcdom_regex*, entretanto com relação ao destino;

time: tipo ACL de horário. Permite ser utilizado para especificar dias da semana e horários. Os dias da semana são definidos através de letras que os representam, e os horários através de intervalos na forma hora:minuto_inicio-hora:minuto_final. Os dias da semana são especificados assim: S - Sunday (domingo), M - Monday (segunda-feira), T - Tuesday (terça-feira), W - Wednesday (quarta-feira), H - Thursday (quinta-feira), F - Friday (sexta-feira) e A - Saturday (sábado);

url_regex: este tipo percorre a URL a procura da expressão regular especificada. Deve ser observado que a expressão é *case-sensitive*, para que seja *case-insensitive* deve ser usada a opção -i. É o tipo mais comum de ACL dada a flexibilidade proporcionada pelo uso de expressões regulares. Abaixo um exemplo para se criar uma ACL chamada PornSites com a finalidade para controle de conteúdos impróprios, tendo como base o arquivo pornlist: acl PornSites url_regex "/usr/local/*squid*/etc/pornlist";

urpath_regex: tipo semelhante a *url_regex*, mas procura a expressão regular na URL sem levar em conta o nome do servidor e o protocolo, isto quer dizer que a procura vai ser feita apenas na parte da URL após o nome do servidor, como por exemplo, na URL http://www.servidor.com.br/pasta/sexo.html a procura será realizada apenas na parte /pasta/sexo.html. Ela é também *case-sensitive*, para que seja *case-insensitive* deve ser usada a opção -i;

arp: tipo usado para construir lista de acesso baseada no MAC *Address* da interface de rede do cliente, ou seja, em vez de utilizar endereço IP da placa, usa-se o seu endereço MAC;

port: permite realizar o controle pela porta de destino do servidor, neste tipo deve ser especificado o número da porta. Exemplo para especificar portas seguras: acl Safe_ports port 80 21 443 563 70 210 1025-65535;

proto: este serve para especificar o protocolo de transferência (HTTP, FTP entre outros, por exemplo);

maxconn - especifica um limite de conexões vindas de um determinado cliente, interessante para uso com outras ACLs de forma a limitar quantidades de conexões para determinados endereços específicos. Exemplo para determinar um limite máximo de cinco conexões por cliente: acl 5conexoes maxconn 5; e,

method: especifica o tipo de método usado para requisições HTTP, como por exemplo GET, CONNECT ou POST. Exemplo para determinar a utilização do método CONNECT: acl conectar method CONNECT.

Para maiores detalhes e informações sobre o funcionamento de cada delas é possível consultar o site oficial do *Squid* a respeito das ACLs, disponibilizado pelo *Squid* FAQ[5].

4.4 Arquivo de configuração *squid*.conf

O arquivo de configuração do *Squid*, *squid*.conf, nos permite elaborar uma estratégia de controle e operacionalização do servidor *proxy Squid* que melhor se adapta ao ambiente que este está inserido. Por padrão, este arquivo é composto por mais de 3.000 linhas, sendo a maior parte deles constituídas de comentários sobre as funções de cada *TAG* (opção) e como devem ser utilizados.

Segundo Pinheiro (2006), a princípio para ter um servidor configurado e funcional é preciso de apenas 30 linhas aproximadamente no arquivo, pois se deve considerar que boa parte das opções já definidas no *squid*.conf não necessitam ser modificadas, uma vez que representam a configuração mais adequada. Alguns dos pontos mais importantes e necessários para ter um *proxy* cache *Squid* em funcionamento básico são as opções descritas abaixo, conforme o autor:

- **cache_dir ufs /var/cache/*squid* 300 64 64: e**sta opção determina onde e como será feito o cache – no disco rígido, neste caso o diretório de cache é /var/cache/*squid*, com o tamanho de 300 MB, tendo 64 diretórios e, dentro de cada um deles, com mais 64 sub-diretórios, necessários para a operacionalização do cache *Squid*.

Segundo Costa (2013) o fato de deixar estes valores muito pequenos ou muito grandes podem trazer resultado e impactos diferentes, podendo aumentar ou diminuir consideravelmente a velocidade de navegação para o usuário final, além de fatores como desperdiço de espaço em disco caso configurado valores mal calculados;

- **cache_mem 64 MB: o**pção que define o tamanho, a quantidade de memória RAM que o *Squid* irá usar para os objetos em trânsito, isto é, a memória que será utilizada apenas para manipulação dos seus objetos e não ao total de memória consumida por ele, que pode ser duas ou três vezes maior. Este valor geralmente é calculado baseando-se no tamanho que é ocupado o cache (cache_dir), podendo variar dependendo do tamanho alocado para o cache e da arquitetura do sistema operacional (SO), sendo diferente em sistemas 32 bits e 64 bits.

Segundo Costa (2013) um detalhe muito importante que se deve lembrar e ser levado em conta para efeitos de cálculo, é que em sistemas 32 bits, para cada GB de HD destinado ao cache, o *Squid* consome 10 MB de memória RAM e para sistemas 64 bits, para cada GB de HD destinado ao cache, o *Squid* consome 16 MB de memória RAM.

- **hierarchy_stoplist cgi-bin ?:** opção que vem habilitada por padrão, seu uso é recomendado pois é responsável por comunicar ao *Squid* que este deve buscar os dados diretamente na origem, sem passar pelos vizinhos na hierarquia (de servidores *proxy*/cache). Esta configuração se refere a conteúdo dinâmico, portanto se a URL contém o padrão aqui especificado, será tomada a decisão de buscar diretamente o conteúdo no servidor de origem;

```
acl QUERY urlpath_regex cgi-bin ?
no_cache deny QUERY
```

Esta ACL diz ao *Squid* para não armazenar em cache o conteúdo dos CGIs, pois obviamente não é interessante por tratar-se de conteúdo dinâmico;

```
refresh_pattern ^ftp: 1440 20% 10080
refresh_pattern ^gopher: 1440 0% 1440
refresh_pattern . 0 20% 4320
```

As opções acima são definidas por padrão pelo *Squid* pois configuram como são tratados os tempos de vida de cada um dos objetos armazenados no cache, ou seja, o *Squid* utiliza estes valores para verificar se os objetos em cache são os mais recentes ou se estão obsoletos, necessitando assim atualizá-los;

[5] JEFFRIES, A. ***Squid FAQ:*** *access controls in Squid*. Disponível em: <http://wiki.squid-cache.org/SquidFaq/SquidAcl>. Acesso em: 05 out. 2013.

```
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl to_localhost dst 127.0.0.0/8
acl SSL_ports port 443 563
acl Safe_ports port 80 #http
acl Safe_ports port 21 #ftp
acl Safe_ports port 443 563 #https, snews
acl Safe_ports port 70 #gopher
acl Safe_ports port 210 #wais
acl Safe_ports port 1025-65535 #unregistered ports
acl Safe_ports port 280 #http-mgmt
acl Safe_ports port 488 #gss-http
acl Safe_ports port 591 #filemaker
acl Safe_ports port 777 #multiling http
acl CONNECT method CONNECT
```

As ACLs acima fazem parte da configuração padrão do *Squid*, sendo o mínimo recomendável para utilizá-los. Cada um faz referência a determinadas porta de conexão ou portas seguras (através da acl SSL_ports e acl Safe_ports), como porta 443 que diz respeito ao HTTPS (conexão segura/criptografada) e porta 21 ao FTP (protocolo de transferência de arquivos), por exemplo.

Outras funcionam para especificar determinadas faixas de rede – além da máscara dessa rede, através de ACLs como: acl localhost e acl to_localhost;

```
acl MINHA_REDE src 192.168.16.0/24
acl USUARIOS proxy_auth REQUIRED
acl PORNO url_regex -i "/etc/squid/porn"
acl PORNO1 url_regex -i "/etc/squid/porn1"
acl NAO_PORNO url_regex -i "/etc/squid/noporn"
```

Fazem parte da seção em que são configuradas e definidas as ACLs criadas pelo usuário (administrador de rede), a fim de especificar listas que podem conter endereços IP, sites ou determinadas palavras-chave que são permitidos o acesso ou negados, como acontece nas ACLs: acl NAO_PORNO e acl PORNO. Também é permitido especificar qual rede que se pretende controlar, administrar ou filtar, graças à acl seguinte: acl MINHA_REDE src 192.168.16.0/24;

```
http_access allow manager localhost
http_access deny manager
http_access allow localhost
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
```

Esta parte refere-se à definição do que será permitido (*allow*) ou negado (*deny*) a passagem/o acesso, isto é, o tráfego de dados e informações. São as regras de acesso referentes às ACLs da parte da configuração padrão do *Squid* citadas acima. Por exemplo, é permitido a passagem de tudo que é referente à acl localhost, através da linha: http_access allow localhost ;

```
http_access allow USUARIOS NAO_PORNO MINHA_REDE
http_access allow USUARIOS !PORNO !PORNO1 MINHA_REDE
```

Semelhante à anterior, mas esta faz referência para controlar as ACLs criadas pelo usuário (administrador de rede), e também especifica do que será permitido (*allow*) ou negado (*deny*) a passagem/o acesso, isto é, o tráfego de dados e informações. Por exemplo, é permitida a passagem de tudo que é referente às ACLs USUARIOS, NAO_PORNO e MINHA_REDE concomitantemente, ou seja, têm de satisfazer as três ao mesmo tempo, através da linha: http_access allow USUARIOS NAO_PORNO MINHA_REDE;

Seguem mais alguns exemplos de regras:

- http_access deny all: esta regra de acesso é recomendada para uso como última regra da lista define o acesso ao *proxy*. Ela diz ao *Squid* que se nenhuma das regras anteriores for aplicada o acesso será então negado, portanto seu uso vai impedir acessos indesejados ao *proxy;*
- icp_access allow all: permite o acesso a porta icp de acordo com a configuração feita na ACL all, que no nosso caso representa qualquer origem. Está porta é usada para troca de informações entre servidores *proxy*;
- error_directory /usr/share/*squid*/errors/Portuguese: ao usar esta opção pode-se determinar em qual linguagem serão apresentadas as mensagens de erro, neste caso na língua portuguesa;
- visible_hostname www.meu_seite.com.br: esta opção permite mostrar o nome do servidor configurado nas mensagens de erro, caso contrário o *Squid* irá tentar descobrir o nome; e,
- cache_mgr email_administrador: opção para configuração do e-mail do administrador do *proxy* que vai aparecer nas mensagens de erro. Isto é importante para que os usuários tenham informações de como interagir com o responsável pelo servidor em caso de problemas, como por exemplo um acesso bloqueado de forma errada.

Por fim, em ocasiões onde se utiliza o *proxy* transparente, há opções que fazem com que o *Squid* se comporte como um servidor Web, de forma que o cliente não perceba que está interagindo com um *proxy*/cache, apresentadas abaixo:

httpd_accel_port 80
httpd_accel_host virtual
httpd_accel_with_*proxy* on
httpd_accel_uses_host_header on

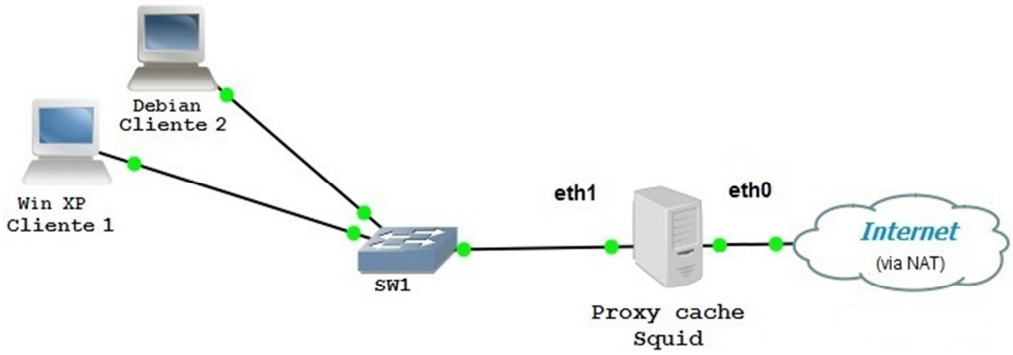
5. ESTUDO PRÁTICO UTILIZANDO *SQUID*

Após compreensão dos conceitos básicos e gerais abordados sobre *Squid* e servidores de *proxy* e cache, alavancados através do desenvolvimento deste trabalho, propõe-se um estudo prático utilizando *Squid* como um serviço para acesso à rede, à Internet, sendo o intermediário entre o cliente e a rede mundial de computadores, a Web, conforme mostrado na Figura 8.

O objetivo do estudo prático é comparar dois cenários utilizando *Squid*, um com o recurso de *Web caching* desligado (*cache OFF*) e outro com este recurso ligado (*cache ON*), a fim de verificar e comprovar a

eficiência do *Squid* como servidor de cache e *proxy* para otimizar o uso da banda de Internet, além de permitir o controle de acesso a determinados conteúdos previamente configurados no *squid*.conf e verificar características como o uso de recursos pelo *Squid* no servidor *proxy*.

Figura 2: Ilustração do ambiente de testes utilizando *Squid*



Fonte: Próprio autor

Em relação ao ambiente de teste, são compreendidos em dois cenários, sendo eles:

- Cenário 1: ambiente sem utilização do servidor *proxy* cache *Squid*, isto é, tendo o serviço que realiza Cache Web desativado – o qual denominaremos de *cache OFF* – para acesso à Internet, resultando numa conexão direta com a Internet sem a intervenção do cache do *Squid*, no qual os clientes fazem suas requisições diretamente ao servidor externo, de origem (que detêm o conteúdo por ele solicitados). Em outras palavras, é um cenário que não utilizará o sistema de caching para armazenar conteúdos Web acessados pelos usuários;

- Cenário 2: denominado de *cache ON*, é o ambiente com servidor *Squid proxy* cache para acesso à Internet, com o *Squid* atendendo as requisições feitas pelas máquinas clientes locais (rede local). O serviço que realiza Cache Web é ativado neste cenário, e nos permite realizar *caching* de conteúdos Web, além de controlar o acesso à Internet, criando regras de acesso do que é permitido e do que é negado, proibido.

A seção a seguir aborda uma visão geral dos sistemas e componentes que foram utilizados, em questões de hardware e software, além de ferramentas usadas no auxílio para que a conclusão dos testes fossem possíveis, caso como para realizar medições dos tempos de *download* e registro dos níveis de uso de recursos do servidor *Squid*, em questões de desempenho.

5.1 Sistemas e componentes utilizados

Os testes foram conduzidos usando-se máquinas virtuais ou *virtual machines (VMs)*, através do software Oracle VM VirtualBox v4.3.0, ferramenta bastante conhecida quando se trata de virtualização de sistemas operacionais (SOs). Para isto, foram criados três VMs, cada um utilizando um SO específico (Figura 2), sendo os seguintes:

1º) Um servidor com sistema operacional Linux Debian 7.2.0 (*codename wheezy*) para servir de *proxy* cache *Squid*, sem utilização de interface gráfica, fato que aumenta a performance na atuação como servidores; e,

2º) Duas estações clientes: sendo um deles rodando Windows XP Professional SP2 e como segundo foi instalado o Debian 7.2.0 com interface gráfica Xfce.

Para o seguinte estudo, foi utilizado como plano de Internet banda larga um *link* com conexão de 1 Mbps (megabits por segundo).

Como máquina física foi utilizado um notebook (sistema hospedeiro) para instalar e executar o software VirtualBox e consequentemente as três *VMs*. Possui as seguintes especificações conforme Tabela 1 abaixo:

Tabela 1: Especificações de hardware e software do notebook utilizado

Componentes notebook	Especificações
Processador	Intel® Celeron(R) Dual-Core CPU T3300 64 bits @ 2 GHz, 1MB cache L2
Sistema Operacional	Windows 8 Professional with Media Center x86
Memória RAM	A-DATA 2GB DDR2 800 MHz SDRAM
Disco rígido	Seagate 500GB SATA, 5400RPM
Chipset	NVIDIA MCP75L
Placa de rede	Qualcomm Atheros AR9285 Wireless Network Adapter e Realtek PCIe GBE Family Controller
Placa gráfica	NVIDIA GeForce G205M CUDA

Fonte: Próprio autor

Softwares e ferramentas utilizadas nas VMs:

Abaixo, detalha-se os softwares utilizados nas máquinas virtuais para os testes, tanto do servidor *proxy Squid* como das máquinas clientes.

1) Servidor *proxy* cache *Squid*:

Foram instalados (via gerenciador de pacotes APT) e configurados os seguintes serviços e ferramentas:

- ***Squid*** (*squid3*) – versão 3.1.20. Atuando como servidor de *proxy* e cache, disponibilizando o serviço para acesso à rede/Internet pelos clientes;
- ***DHCP Server*** (isc-dhcp-server) – versão 4.2.2. Serviço de DHCP necessário para a atribuição dinâmica de endereços IP às máquinas clientes da rede local; e,
- ***atop*** – versão 1.26. O atop é um monitor de desempenho capaz de relatar a atividade de todos os processos do sistema, ou seja, uma ferramenta para monitoramento tempo real que permite registrar em *logs* sobre os níveis de uso dos recursos do sistema, como CPU, memória e rede (SOUZA, 2012).

2) Máquinas clientes:

Foi utilizada apenas a ferramenta ***GNU Wget*** – versão 1.13.4 para Debian e versão 1.11.4 para Windows. É um software livre que realiza *downloads* de arquivos via linha de comando, ou seja, via terminal, e suporta protocolos como HTTP, HTTPS, FTP e *proxy* HTTP, sendo facilmente instalado em qualquer sistema *Unix-like*, derivados Unix assim conhecidos (FREE SOFTWARE FOUNDATION, 2012). E segundo Cardoso (2010) o Wget também é compatível para ambientes Windows.

O GNU Wget será a ferramenta responsável por exercer a atividade como gerenciador de downloads nos testes, em ambas as estações clientes (XP e Debian), trazendo consigo o benefício de mostrar o tempo total gasto para baixar um determinado arquivo, e sua velocidade média atingida.

5.2 Metodologia

A metodologia utilizada para o estudo prático será compreendida em três partes, da seguinte maneira: primeiramente é descrito o método que fora utilizado para o a execução dos testes, em seguida é feito o detalhamento das máquinas virtuais utilizadas, sendo divido em dois tópicos, o primeiro diz respeito ao servidor *Squid* e a segundo diz respeito às máquinas clientes (tendo duas estações). Por fim, ao final desta seção, são mostradas as etapas do processo, do passo a passo cronológico das tarefas para a condução do projeto prático.

5.2.1 Detalhamento das máquinas virtuais:

Como foco de teste para o estudo prático, propõe-se o *download* do software Mozilla Firefox versão 24.0 pt-br (Português Brasil), um navegador Web gratuito com tamanho de arquivo 21.6 MB, através da página oficial da Mozilla Foundation[6], a desenvolvedora e mantenedora do *browser*. Com a finalidade de mostrar a diferença do tempo necessário para descarregar o arquivo além da otimização do uso do link de Internet, nos dois cenários acima propostos.

Para cada máquina cliente (duas) será realizado o download de cinco vezes repetidas e contínuas a fim de obter uma média do tempo total gasto para baixar o arquivo em questão, em ambos os cenários, com cache OFF e também com cache ON. Totalizando 20 downloads, sendo 10 para os dois clientes no cenário de cache OFF e 10 para os mesmos clientes, mas no ambiente cache ON, conforme ilustra na Tabela 2.

Tabela 2: Número de repetições download nos dois cenários propostos

	Número de repetições do download	
Máquina cliente:	*Cenário cache OFF*	*Cenário cache ON*
Cliente Debian	5	5
Cliente XP	5	5
Total parcial:	10	10
Total:	**20**	

Fonte: Próprio autor

Concomitantemente, na execução de cada um dos *downloads*, dentro do intervalo, será feito o registro (*log*) no servidor *Squid* via monitor atop com os níveis de desempenho (uso de recursos) da CPU, memória e rede (eth0 e eth1).

A seguir, é feito o detalhamento das *virtual machines* utilizadas:

1º) Servidor *proxy* cache *Squid*:

No VirtualBox, foi alocado ao servidor a utilização de 256 MB de memória RAM e 8 GB de HD, além de 42 MB de memória gráfica.

Conforme ilustração da Figura 2, o servidor possui duas interfaces/placas de rede, sendo que a primeira (eth0) opera em modo NAT com o notebook hospedeiro para comunicação e acesso à Internet, e a segunda (eth1) em modo rede interna, para comunicação direta com as máquinas clientes.

Para efeito do estudo prático, foi escolhido registrar no servidor *Squid* usando a ferramenta atop apenas os níveis de uso de recursos como CPU, memória e rede (esta possui duas interfaces de rede: eth0 e eth1), uma vez que são os pontos mais relevantes, quando se trata de desempenho em servidores *proxy* e seus impactos, conforme detalhado abaixo na Tabela 3:

Tabela 3: Recursos do sistema a ser registrado pelo *atop* no servidor *Squid*

Recurso do sistema a ser registrado	Método de medição
CPU	Porcentagem de uso, na escala de medição que varia de 0% a 100% de uso e ocupação da CPU
Memória	Quantidade total de memória utilizada pelo sistema, medidos em MB (megabyte)
Rede	Quantidade de consumo do tráfego de dados que passa, medidos em Kbps (kilobyte por segundo) para cada uma

[6] Mozilla Foundation**:** Firefox projeto. Disponível em: <http://www.mozilla.org/firefox/>. Acesso em: 17 out. 2013.

(Interfaces: eth0 e eth1)	das interfaces de rede: *eth0* e *eth1*

Fonte: Próprio autor

2º) Máquinas clientes:

No VirtualBox, foi alocado ao cliente 1 (Windows XP) a utilização dos mesmos 256 MB de memória RAM e 20 GB de tamanho em disco, além de 120 MB de memória para vídeo. Já no cliente 2 (Debian) foi definido a mesma memória RAM de 256 MB, 8 GB de HD e 120 MB de memória gráfica.

Para ambas, as interfaces/placas de rede estão configuradas como rede interna, permitindo o acesso à Internet somente através da comunicação com o servidor *proxy* cache *Squid*, conforme ilustra a Figura 2.

A seguir, detalha-se o cronograma das etapas do processo, o passo a passo de como o teste foi preparado, conduzido e realizado:

1ª) etapa: preparação das máquinas virtuais (três), instalando-os com os devidos sistemas operaionais (um cliente e um servidor com Debian, e um cliente com Windows XP);

2ª) etapa: instalação dos softwares necessários:

- Servidor: squid3, isc-dhcp-server e atop (através do gerenciador de pacotes apt-get);
- Estações clientes: wget (através do gerenciador de pacotes apt-get para o Debian e para o XP foi necessário baixar o executável binário para utilização);

3ª) etapa: configuração do servidor com o serviço de atribuição dinâmica de IP via DHCP (interfaces, dhcpd.conf, isc-dhcp-server) e do serviço *Squid* (squid.conf);

4ª) etapa: execução dos testes de download nas máquinas clientes em ambos os cenário citados, fazendo registros (no servidor) dos níveis de desempenho via a ferramenta monitor atop para cada download executado; e,

5ª) etapa: parte final, coleta e análise dos logs de desempenho, passando os dados em planilha (Microsoft Excel) para cálculo dos tempos gastos, nível mínimo e máximo do uso de CPU, memória e rede a fim de obter a média do valores e fazer a comparação dos diferentes resultados obtidos em cada um dos cenários – cache OFF e cache ON, conforme a seção a seguir.

5.3 Arquivo de configuração *squid*.conf utilizado

Nesta subseção será apresentado o arquivo *squid*.conf utilizado na execução dos testes. Foram utilizados nos testes duas situação, cache OFF e cache ON, que são descritas a seguir:

- Cache OFF:

O cenário com o recurso de cache web do *Squid* desligado é feito da seguinte maneira, comentando-se (a partir do símbolo #) as linhas referentes a opções de cache_mem, referente ao tamanho máximo/mínimo do objeto na memória/disco, níveis máximo/mínimo do cache_swap e, por fim, a opção cache_dir referente ao diretório do cache, conforme as linhas abaixo:

```
#cache_mem 64 MB                          #definicao tamanho cache na memoria RAM
#maximum_object_size_in_memory 32 KB      #tamanho max do objeto na RAM
#maximum_object_size 1024 MB              #tamanho max do objeto no disco
#minimum_object_size 0 KB                 #tamanho min do objeto no disco
#cache_swap_low 90
#cache_swap_high 95
#cache_dir ufs /var/spool/squid3 100 16 256    #aponta o dir. do cache
```

- Cache ON:

O cenário com o recurso de Cache Web do *Squid* ligado é feito de maneira semelhante, apenas necessário realizar o inverso do cache OFF, removendo a indicação de comentário das linhas, conforme se segue:

```
cache_mem 64 MB                           #definicao tamanho cache na memoria RAM
maximum_object_size_in_memory 32 KB       #tamanho max do objeto na RAM
maximum_object_size 1024 MB               #tamanho max do objeto no disco
minimum_object_size 0 KB                  #tamanho min do objeto no disco
cache_swap_low 90
cache_swap_high 95
cache_dir ufs /var/spool/squid3 100 16 256    #aponta o dir. do cache
```

A seguir, é mostrado o arquivo de configuração utilizado para o *proxy Squid*, sendo comentadas as principais opções utilizadas e do que se tratam cada uma delas:

```
### Inicio squid.conf ###
http_port 3128 transparent        #definicao da porta Squid e modo transparente
visible_hostname YOU              #esta opcao permite definir o hostname do proxy

error_directory /usr/share/squid3/errors/Portuguese          #dir. de erro em PT

cache_mem 64 MB                                  #definicao tamanho cache na memoria RAM
maximum_object_size_in_memory 32 KB       #tamanho max do objeto na RAM
maximum_object_size 1024 MB                 #tamanho max do objeto no disco
minimum_object_size 0 KB                      #tamanho min do objeto no disco
cache_swap_low 90
cache_swap_high 95
cache_dir ufs /var/spool/squid3 100 16 256              #aponta o dir. do cache
access_log /var/log/squid3/access.log squid             #dir dos logs de acessos

acl manager proto cache_object
acl localhost src 127.0.0.1              #definicao da acl com origem o localhost


###    Minhas ACLs    ###
#definicao da acl rede interna para comunicao com as maquinas clientes:
acl rede_interna src 10.1.1.0/24

#a opcao abaixo define os sites que possum como dominio ".com.pt":
acl dominios_compt dstdomain .com.pt

#acl espeficicando palavras que contenham "bol":
acl blockpalavra1 url_regex bol

#a opcao abaixo permite bloquer via determinadas palavras:
acl palavrablock url_regex -i "/etc/squid3/auxiliares/palavrablock.conf"

#bloquear via IP:
acl ipblock src "/etc/squid3/auxiliares/ipblock.conf"

###    Portas Seguras    ###
acl SSL_ports port 443
acl Safe_ports port 80 #http
acl Safe_ports port 21 #ftp
acl Safe_ports port 443 #https
acl Safe_ports port 70 #gopher
acl Safe_ports port 210 #wais
acl Safe_ports port 1025-65535 #unregistered ports
acl Safe_ports port 280 #http-mgmt
acl Safe_ports port 488 #gss-http
acl Safe_ports port 591 #filemaker
acl Safe_ports port 777 #multiling http

#Bloqueio e Liberacoes
acl CONNECT method CONNECT #acl que define metodo das conexao/requisicao HTTP
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow localhost

### Meus Bloqueios ###
```

```
#esta opcao permite negar acesso a sites que possuam como dominio o ".com.pt":
http_access deny dominios_compt

#esta opcao permite negar o acesso a paginas que contenham a palavra "bol":
http_access deny blockpalavra1

#nega as duas acls ao mesmo tempo palavrablock E ipblock:
http_access deny palavrablock ipblock

#opcao que libera acesso as maquinas vindos da rede interna:
http_access allow rede_interna

#Opcao final que nega tudo o que nao tiver definido nas opcoes acima:
http_access deny all

#Configuracoes finais mantidas
coredump_dir /var/spool/squid3
refresh_pattern ^ftp:    1440 20% 10080
refresh_pattern ^gopher: 1440 0% 1440
refresh_pattern -i (/cgi-bin/|\?) 0 0% 0
refresh_pattern (Release|Packages(.gz)*)$ 0 20% 2880
refresh_pattern . 0 20% 4320

### Fim squid.conf ###
```

5.4 Resultados obtidos

Após a execução dos testes práticos com as máquinas virtuais, os resultados obtidos foram os seguintes:

Aspecto 1: Tempo médio de download:

Neste aspecto, apresenta-se o tempo médio resultado dos cinco testes de repetições de download do navegador Firefox, para cada um dos cenários de ambos os clientes, ou seja, com quatro situações:

- Cliente Debian fazendo downloads com o cache *Squid* desligado;
- Cliente XP fazendo downloads com o cache *Squid* desligado;
- Cliente Debian fazendo downloads com o cache *Squid* ligado; e,
- Cliente XP fazendo downloads com o cache *Squid* ligado;

Conforme Tabela 4 abaixo, percebe-se que a diferença de tempo necessário por cada máquina cliente é diferente, caso mais perceptível ainda quando faz a comparação entre os cenários com cache OFF e outro com cache ON. Com uma média de tempo (em segundos) de 54,5s (Debian) e 53,8s (XP), contra os 167s e 177s respectivamente no ambiente cache OFF. Diferença esta significantemente positiva, com um ganho em porcentagem no quesito redução do tempo necessário para download em 32,63% (Debian) e 30,40% (XP), graças ao recurso de Web caching do *Squid*, pois com o cache ON (realizando caches de conteúdo) reduz os tempos necessários para acesso à Web que, neste caso, beneficiou as máquinas clientes, diminuindo o tempo requerido para descarregar o arquivo *Firefox Setup 24.0.exe*. Além disso, ajuda a reduzir a utilização da banda de Internet, pois as estações fazem requisição primeiramente ao *proxy Squid*, e caso este possua uma cópia armazenada em seu cache, disponibilizará aos clientes, não requerendo a conectar-se a um servidor externo. Abaixo os tempos médios obtidos para cada uma dos ambientes conforme a Tabela 4.

Tabela 4: Tempo médio total gasto (cenário x máquina cliente)

	Tempo total gasto
Debian **(cache off)**	167,0s
XP **(cache off)**	177,0s
Debian **(cache on)**	54,5s
XP **(cache on)**	53,8s

Fonte: Próprio autor

Aspecto 2: Valores do consumo de recursos no servidor *Squid*;

Neste aspecto, apresenta-se o consumo de CPU, memória RAM e das interfaces de rede eth0 e eth1 do servidor *Squid*, ressaltando que a organização dos valores é feita através do relacionamento máquina cliente com o cenário nele atribuído, ou seja, para cada cliente – o XP e o Debian – quando executa as tarefas do teste de tempo médio de download, é gerado no servidor um log com os consumos daquele exato intervalo de tempo, sendo quatro situações:

- Cliente Debian fazendo downloads com o cache *Squid* desligado;
- Cliente Debian fazendo downloads com o cache *Squid* ligado;
- Cliente XP fazendo downloads com o cache *Squid* desligado; e,
- Cliente XP fazendo downloads com o cache *Squid* ligado;

Tabela 5: Uso de recursos do sistema do servidor *Squid* (máquina cliente x cenário)

	CPU (uso %)	RAM (cons. MB)	eth0 (tráf. Kbps)	eth1 (tráf. Kbps)
Debian **(cache off)**	4,8%	89,24 MB	828,20 Kbps	840,20 Kbps
Debian **(cache on)**	9,2%	128,62 MB	237,20 Kbps	6896,90 Kbps
XP **(cache off)**	7,0%	91,26 MB	863,60 Kbps	890,70 Kbps
XP **(cache on)**	5,9%	126,56 MB	176,50 Kbps	7338,30 Kbps

Fonte: Próprio autor

Na sequência, é analisado cada um dos recursos, nos quesitos de CPU, memória RAM e rede:

CPU: no aspecto da utilização de recursos da CPU, em porcentagem (%) de uso, conforme Tabela 5 acima, pode-se concluir o seguinte:

1. Percebe-se que o maior consumo alcançado foi de 9,2% de uso da CPU, quando o cliente 2 Debian, no cenário cache ON, realizava os processos de download (5 repetições), valor este que contradiz quando o mesmo processo era realizado pelo cliente 1 XP, com ocupação de apenas 5,9% da CPU, uma diferença de 3,3% em utilização CPU; e,
2. Nos cenários cache OFF, para ambos os clientes, os valores obtido foram 4,8% (Debian) e 7,0% (XP), uma diferença de 2,8% na utilização da CPU.

RAM: conforme Tabela 5 acima, no quesito de consumo de RAM, em *megabytes*, pode-se concluir o seguinte:

1. O consumo de memória RAM se mostrou maior quando o *Squid* estava configurado para realizar cache web (cache ON), consumindo valores em 128,62 MB (no Debian) e 126,56 MB (no XP). Valores estes pois os clientes solicitam o arquivo à um servidor externo, uma vez que o *Squid* utiliza parte da memória RAM para os processos de caching e manipulação de objetos em trânsito, definidos no arquivo de configuração *squid*.conf pela opção cache_mem; e,

2. Enquanto com o cache OFF, a quantidade de consumo se mostrou relativamente menor em ambos os clientes, com utilização de 89,24 MB no Debian e 91,26 MB no XP, uma economia de entorno de 35 – 39 MB de RAM em relação ao cenário cache ON de ambos clientes.

Consumo: conforme Tabela 5 acima, no quesito de consumo e utilização das interfaces de rede do servidor, com o tráfego gerado, conclui-se o seguinte:

1. Na interface eth0, que possui conexão com a Internet via NAT com a máquina física – notebook, mostrou que o uso da banda, do *link* é significativamente maior quando esta opera no cenário de cache OFF, para ambos os clientes, com valores de utilização do tráfego de rede que atingem, em Kbps (*kilobits* por segundo), 828,20 Kbps quando o cliente Debian fazia o download do navegador Firefox e 863,60 Kbps para o cliente XP, realizando a mesma tarefa;

Em modo cache ON, tal valor se mostrou relativamente muito menor, algo plausível devido ao recurso Web cache, mas em contrapartida, consome uma quantidade maior de RAM. Com valores de utilização de tráfego em 237,20 Kbps (cliente Debian) e 176,50 Kbps (cliente XP), uma diferença de aproximadamente 591–687 Kbps a menos de uso da rede eth0 de ambos clientes; e,

2. Na interface eth1, que opera no modo rede interna para comunicação somente com as máquinas clientes, percebe-se que o uso é relativamente muito maior quando o *Squid* está configurado no modo cache ON, tendo 6896,90 Kbps de tráfego alavancado pelo Debian e 7338,30 Kbps pelo XP.

Fato este explicado pelo cache *Squid*, uma vez que ambos os clientes solicitam o arquivo a partir do cache, consumindo assim mais recursos da rede local, reduzindo o uso do eth0 (internet via NAT).

Realizando-se uma análise em âmbito geral, comparando-se todos os resultados obtidos neste projeto prático utilizando *Squid*, pode-se afirmar que a implantação de um *proxy* cache *Squid* é realmente algo muito interessante e plausível. Os resultados obtidos nos testes mostraram ser muito satisfatórios, com ganhos (em benefício dos clientes) no quesito redução do tempo necessário para download entre o cenário cache OFF e cache ON em 32,63% (Debian) e 30,40% (XP). Os testes também mostraram consumir poucos recursos da máquina servidor *Squid*, com taxas de utilização em níveis razoavelmente aceitáveis, em termos de CPU, memória RAM e das interfaces de rede.

Outro aspecto a ser levado em consideração, justificando o uso do *Squid* é em relação ao seu recurso de Cache Web, um serviço muito útil e eficiente quando se tem mais de um usuário da rede local requisitando um mesmo objeto ou conteúdo. Neste contexto, a partir do segundo acesso a um determinado conteúdo, o *proxy* disponibilizará tal objeto solicitado diretamente a partir de seu cache, sem precisar de um novo acesso a um servidor externo, reduzindo assim a utilização da banda de Internet, do link disponível.

6. CONSIDERAÇÕES FINAIS

O aumento da demanda do uso da Internet, impulsionado pelos mais diversos usos, como usuários finais, empresas instituições de ensino e pesquisa, órgão públicos e governamentais entre outros, tem deixado o acesso à Internet insatisfatório e lento, devido as altas latências e tempos de resposta. Em vista deste cenário, a utilização de servidores de *proxy* e cache, em especial o *Squid*, é uma das soluções e alternativas melhor propostas para minimizar o problema.

A partir do desenvolvimento deste trabalho, com a apresentação e análise sobre servidores *proxy* cache, e consequentemente do *Squid*, observa-se que o recurso que realiza cache de conteúdos da Web, conhecido como Cache Web, disponibilizado pelo *proxy* cache *Squid* é realmente muito útil, obtendo resultados positivos no estudo prático realizado neste trabalho, pois reduz a utilização da banda de Internet, o tempo necessário para realizar *download* de arquivos, além de permitir, como recursos adicionais, o controle de acesso à determinadas páginas e conteúdos considerado inapropriados para certos horários expedientes e nos ambientes acadêmicos para os períodos de aula, e outras em ocasiões para negar o acesso a sites com teor impróprio.

Acelerar o acesso à Internet, dos conteúdos que ela dispõe, melhorar o desempenho e garantir um melhor fluxo de funcionamento da rede como um todo, a implantação do *Squid* certamente é uma solução muito viável, tanto em fatores de obtenção de benefícios operacionais às instituições dos mais diversos tipos como para questões de custo, não exigindo grandes investimentos financeiros da organização que utilizará, por ser um software livre e gratuito, licenciado nos termos da *General Public License* (GPL) tendo seu código-fonte com livre acesso, uso e modificação. Em contrapartida, é necessário lembrar que existe um custo complementar, muita vezes importante e requerido, a fim de saber como administrar e operar o *Squid* da forma mais adequada, eficiente e eficaz, uma vez que isso exige um certo tempo para aprendizado e configuração do mesmo.

Como proposta de trabalhos futuros, sugere-se o teste e implantação do *Squid* em um ambiente real, com a utilização de computadores físicos, pois os valores obtidos, o desempenho e a performance em âmbito geral seriam melhor mensurados e aproveitados.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUADO, A. G. **Tutorial *proxy Squid* e Sarg versão 3**. 2013. Disponível em: <http: //www.fatec.edu.br/moodle/pluginfile.php/2869/mod_resource/content/0/Encontros%20-%20*Squid*%20-%20V 1.pdf>. Acesso em: 18 abr. 2013.

ARLITT, M.; FRIEDRICH, R.; JIN, T. ***Performance evaluation of web proxy cache replacement policies***. California: Editora Springer Science, 1998.

CARDOSO, J. **Usando o comando wget no Windows**. 2010. Disponível em: <http://www.oficinadanet.com.br/artigo/windows/usando_o_comando_wget_no_windows>. Acesso em: 17 out. 2013.

CERT.br. **Cartilha de segurança para internet.** Disponível em: <http://cartilha.cert .br/>. Acesso em: 02 set. 2013.

CHESHIRE, S. ***Latency and the quest for interactivity***. Disponível em: <http:// www.stuartcheshire.org/papers/LatencyQuest.html>. Acesso em: 20 ago. 2013.

CISCO DOCWIKI. ***WCCP network caching***. 2012. Disponível em: <http://docwiki. cisco.com/wiki/Network_Caching_Technologies#WCCP_Network_Caching>. Acesso em: 27 ago. 2013.

COSTA, J. R. ***Squid*: Calculando cache_dir e cache_mem. Disponível em: <http:// www.vivaolinux.com.br/dica/*Squid*-Calculando-cache-dir-e-cache-mem>. Acesso em: 03 nov. 2013.

COTTLE, G. ***New report reveals shifting trends in global Internet traffic*.** Disponível em: <http://blogs.informatandm.com/2172/press-release-new-report-reveals-shifting-trends-in-global-internet-traffic />. Acesso em: 24 ago. 2013.

FERNANDES, A. R.; SERRÃO JUNIOR, C. C. **Implementação de um servidor *proxy*-cache *Squid* em ambiente de redes corporativa**. 2010. Disponível em: <http://www3.iesam-pa.edu.br/ojs/index.php/sistemas/article/view/555>. Acesso em: 30 jul. 2013.

FREE SOFTWARE FOUNDATION. ***GNU Wget***. 2012. Disponível em: <http://www. gnu.org/software/wget/>. Acesso em: 17 out. 2013.

PEARSON, O. ***Squid user guide***. Disponível em: <http://www.deckle.co.uk/squid-users-guide/>. Acesso em: 30 set. 2013.

PINHEIRO, A. C. S. **Uso e configuração do *Squid***. Disponível em: <http://www. *squid*-cache.org.br/index.php?option=com_content&task=view&id=81&Itemid=27>. Acesso em: 13 set. 2013.

PONTES, E.; HIRATA, S.; HONÓRIO, S. **Segurança e aceleração de internet: Utilização de *proxy* servers para manutenção de web caches.** Disponível em: <http://www.pontes.inf.br/docs/*proxy*.pdf>. Acesso em: 11 ago. 2013.

SABOCINSKI NETO, A. ***Proxy* reverso? O que danado é isso?**. Disponível em: <http://alfredosabo.blogspot.com.br/2009/05/*proxy*-reverso-o-que-danado-e-isso.html>. Acesso em: 28 ago. 2013.

SILVA, A. Q. **Implantação de servidor *proxy* utilizando o *Squid* em modo não autenticado, junto com o DansGuardiam para controle de conteúdo na instituição de ensino**. 2010. Disponível em: <http://www.ginux.ufla.br/node/317>. Acesso em: 30 jul. 2013.

SILVA, L. R. B. B. **Aceleração HTTP**: um comparativo de performance entre as soluções *Squid* e Varnish. 2009. Disponível em: <http://www.ginux.ufla.br/node/294>. Acesso em: 09 ago. 2013.

SILVA, N. V. ***Proxy* reverso com Apache**. 2011. Disponível em: <www.vivaolinux.com.br/artigo/*Proxy*-Reverso-com-Apache/?pagina=1>. Acesso em: 28 ago. 2013.

SOUZA, D. L. **Atop:** monitor de processo e carga do sistema. 2012. Disponível em: <http://www.vivaolinux.com.br/dica/Atop-Monitor-de-Processo-e-Carga-do-Sistema>. Acesso em: 16 out. 2013.

SQUID SOFTWARE FOUNDATION. ***Squid:*** *Optimising web delivery*. Disponível em: <http://www.squid-cache.org/>. Acesso em: 05 jul. 2013.

STANGER, J.; LANE, P.; DANIELYAN, E. **Rede segura Linux**. Rio de Janeiro: Alta Books, 2002.

TANENBAUM, A. S. **Redes de computadores.** 4 Ed. Rio de Janeiro: Editora Campus, 2003.

WATANABE, C. S. **Introdução ao cache de web**. 2000. Disponível em: <http://www.rnp.br/newsgen/0003/cache.html>. Acesso em: 09 ago. 2013.

WESSELS, D.; CLAFFY K. ***Internet cache protocol (ICP)***. Disponível em: <http://icp.ircache.net/>. Acesso em: 01 out. 2013.

ZANONI, G. S. **Servidor *proxy* (*Squid*).** Disponível em: <www.vivaolinux.com.br/artigo/Servidor-*proxy*-%28*Squid*%29>. Acesso em: 28 ago. 2013.

Prof. Me. Gabriel de Souza Fedel

Possui mestrado em Ciência da Computação pela Universidade Estadual de Campinas - Unicamp (2011), graduação em Ciências de Computação pela Universidade de São Paulo (2008) e Curso Técnico em Informática pelo Colégio Técnico da Unicamp (2002). Tem experiência na área de Banco de Dados, atuando principalmente nos seguintes temas: mineração de dados, detecção de agrupamentos de dados, integração de mineração de dados com SGBD e busca multimodal. É Professor Assistente na FATEC Americana (Integrada ao CEETEPS - Centro Estadual de Educação Tecnológica Paula Souza) desde 2013.

Contato: gabrielfedel@gmail.com

Fonte: CNPQ – Curriculo Lattes