UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES

PETER ANDREAS ENTSCHEV

SQUID E A VIRTUALIZAÇÃO: REFINANDO AS CONFIGURAÇÕES DE *SOFTWARE* PARA OBTER O MELHOR DESEMPENHO EM SERVIDORES *PROXY* COM ALTA DEMANDA

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2010

PETER ANDREAS ENTSCHEV

SQUID E A VIRTUALIZAÇÃO: REFINANDO AS CONFIGURAÇÕES DE *SOFTWARE* PARA OBTER O MELHOR DESEMPENHO EM SERVIDORES *PROXY* COM ALTA DEMANDA

Trabalho de Conclusão de Curso apresentado ao Departamento Acadêmico de Eletrônica como requisito parcial para obtenção do grau de Tecnólogo no Curso Superior de Tecnologia em Sistemas de Telecomunicações da Universidade Tecnológica Federal do Paraná.

Orientador: Hugo Vieira Neto

CURITIBA

2010

PETER ANDREAS ENTSCHEV

SQUID E A VIRTUALIZAÇÃO: REFINANDO AS CONFIGURAÇÕES DE *SOFTWARE* PARA OBTER O MELHOR DESEMPENHO EM SERVIDORES *PROXY* COM ALTA DEMANDA

Este trabalho de conclusão de curso foi julgado e aprovado como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Telecomunicações pela Universidade Tecnológica Federal do Paraná.

Curitiba, 31 de agosto de 2010.

José Ricardo Alcântara
Coordenador de Curso
Departamento Acadêmico de Eletrônica

Décio Estevão do Nascimento
Responsável pelo Trabalho de Conclusão de Curso
Departamento Acadêmico de Eletrônica

BANCA EXAMINADORA

Prof. Dr. Hugo Vieira Neto
Orientador

Prof. Dr. Luís Alberto Lucas

Prof M. Sc. Vagner Gonçalves Leitão

AGRADECIMENTOS

Agradeço a meus pais, Cecilia e Erich Entschev, pelo apoio e incentivo dados em relação aos estudos e à educação. Agradeço também ao meu orientador, Hugo Vieira Neto, pela paciência e esforços dedicados com a formulação de críticas construtivas e de todo o apoio fornecido durante os estudos realizados e a escrita do trabalho.

RESUMO

ENTSCHEV, Peter Andreas. Squid e a Virtualização: Refinando as Configurações de *Software* para Obter o Melhor Desempenho em Servidores *Proxy* com Alta Demanda. 97 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas de Telecomunicações, Universidade Tecnológica Federal do Paraná. Curitiba, 2010.

Nesse trabalho realiza-se um estudo de caso de melhoria de desempenho em serviços de *proxy* de rede com o *software* livre Squid, utilizando-se sistemas operacionais também livres, como Linux, FreeBSD e NetBSD, além da virtualização total ou da paravirtualização de máquinas, utilizando *software* de virtualização Xen ou KVM. A melhoria de desempenho estudada é relativa a diversas características dos sistemas operacionais, dos sistemas de arquivos, do próprio *software* Squid ou da rede. O desempenho do *cache* em disco é testado através de comparativos realizados no Linux com os sistemas de arquivos XFS e ReiserFS. A correta escolha de um serviço de DNS ou de resolução de nomes, relativa à sua posição geográfica ou à sua localização dentro da rede, é mostrada como forte alternativa para a redução do tempo de resposta não só de um serviço de *proxy*, mas também de qualquer outro serviço de rede que necessite da resolução de nomes. A utilização do *software* SquidGuard visa também a redução do tempo de resposta, mas através da redução do tempo de processamento de listas de controle de acesso. O uso de diversos sistemas operacionais reflete a diferença que um mesmo serviço ou aplicativo pode sofrer devido às características construtivas destes. Todo o estudo realizado é feito utilizando-se das técnicas de virtualização total ou paravirtualização para as máquinas nas quais o serviço de *proxy* é executado.

Palavras-chave: Desempenho. *Software* Livre. Squid. *Proxy*. Virtualização.

ABSTRACT

ENTSCHEV, Peter Andreas. Squid and the Virtualization: Tuning Software Configurations to Obtain the Best Performance on High Demand Proxy Servers. 97 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas de Telecomunicações, Universidade Tecnológica Federal do Paraná. Curitiba, 2010.

This work presents a case study of performance improvement in network proxy services using the free software Squid and free operating systems, such as Linux, FreeBSD and NetBSD, using full machine virtualization or machine paravirtualization with Xen or KVM. The studied performance improvement relates to various operating systems properties, file systems, the Squid itself or the network. The disk cache performance is tested throughout comparative tests made under Linux with XFS and ReiserFS file systems. The correct choice of a DNS or name resolution service, considering its geographical or network location, is shown as a strong alternative for the response time reduction not only for a proxy service but also for any network server that needs domain name resolution. SquidGuard utilization also aims to reduce the response time, but through the reduction of machine processing time of access control lists. The use of various operating systems reflects the difference that the same service or application may suffer due to constructive properties of those operating systems. The whole study is done using total virtualization or paravirtualization techniques on the machines where the proxy service is executed.

Keywords: Performance. Free Software. Squid. Proxy. Virtualization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

MB	Megabytes
ms	Milisegundos
CPU	*Central Processing Unit* ou Unidade de Processamento Central
KB/s	Quilobytes por segundo
MB/s	Megabytes por segundo
KB	Quilobytes
EB	Exabytes

SUMÁRIO

1 INTRODUÇÃO

Os *proxies* são soluções amplamente utilizadas no mundo da informática, especialmente em grandes redes corporativas. Esses servidores têm dois objetivos principais:

- Acelerar o acesso a um recurso, por meio da técnica de *caching*. É comumente usado para fazer *cache* de páginas de um servidor *web* da Internet, ou seja, armazena as páginas mais frequentemente acessadas, para que não seja necessário consumir banda de Internet toda vez que uma requisição para a página seja realizada (THOMAS; SICAM, 2008).
- Manter os computadores da rede anônimos, atrás deste tipo de servidor, para manter a segurança da rede. O servidor *proxy* é, acima de tudo, um dispositivo de segurança (GRENNAN, 2000).

Este trabalho tem como objetivo realizar o melhor aproveitamento de *hardware* e *software* em uma solução de *proxies* com balanceamento de carga, utilizando o aplicativo Squid para prover tal solução. O Squid é comumente utilizado sob o sistema operacional Linux, mas na vivência prática nem sempre o Linux proverá o melhor desempenho final. No decorrer do trabalho, serão testados outros sistemas operacionais como FreeBSD e NetBSD, ambos da família BSD, para se chegar à solução de melhor desempenho.

Basicamente, o estudo propõe a realização de alterações nas configurações do *software*, tanto do *proxy* (Squid) em si, quanto do sistema operacional utilizado, analisando quantitativamente seu impacto na solução final ou através de gráficos de desempenho da rede (taxas máximas de transferência e número máximo de requisições simultâneas) e da máquina[1] (carga no processador, memória RAM disponível, etc.), além da documentação do processo, chegando ao final do projeto com a melhor configuração encontrada para a situação existente.

No desenvolvimento do projeto será sempre utilizado *software* livre[2] para todas as etapas envolvidas.

[1] No decorrer deste trabalho, a palavra máquina refere-se ao *hardware* utilizado na análise.

[2] *Software* livre nesta dissertação refere-se a *software* licenciado sob licenças como a GPL, idealizada por Richard Stallman no final da década de 1980.

O *hardware* existente para utilização nos estudos é de alto desempenho e específico para servidores de grande porte. Esse *hardware* será utilizado como um modelo de alta performance (objetivo dos estudos, torná-lo o mais eficiente possível) e alta disponibilidade (através do balanceamento da carga entre os dois servidores), duas características que são tendências atuais na área de Tecnologia da Informação.

A virtualização será utilizada como ferramenta de limitação dos recursos físicos disponíveis ao servidor *proxy*, ou seja, evitar o desperdício de recursos de *hardware* com serviços que não necessitem de tal disponibilidade, impedindo que haja demasiada ociosidade dos recursos disponíveis. Além disso, a virtualização provê um mecanismo adicional de segurança, pois, virtualmente, separa serviços em máquinas distintas, impossibilitando que um possível ataque à máquina interfira com outros serviços de rede.

O projeto que se beneficiará com esta pesquisa primeiramente será o Paraná Digital, projeto do Governo do Estado do Paraná, que hoje proporciona inclusão digital a mais de 2000 escolas estaduais do Paraná, e quando estiver totalmente implantado possuirá as mais de 2100 escolas estaduais paranaenses com laboratórios próprios do projeto. Como o Paraná Digital é proveniente de verbas públicas, toda e qualquer redução de custo que possa existir é bem-vinda, visto que essas verbas são sempre muito limitadas e devem ser utilizadas para elevar a qualidade de vida de toda a população (SECRETARIA DE ESTADO DA EDUCAÇÃO DO PARANÁ, 2010).

1.1 PROBLEMA

O problema que se pretende resolver com essa pesquisa é a economia com *hardware* desnecessário, já que o custo com *software* é nulo neste caso (pela utilização de *software* livre gratuito). Essa economia só se tornará possível a partir do momento em que forem realizadas as corretas configurações do *software* envolvido, tanto para o *hardware* existente, quanto para o cenário existente. O cenário levará em consideração o número de clientes simultâneos em horários de pico, a banda disponível para cada cliente e as bandas de entrada e saída disponíveis para atender aos clientes. Entenda-se como banda de entrada a velocidade do *link* que conecta os clientes ao servidor e como banda de saída o *link* que conecta o servidor à Internet.

1.2 JUSTIFICATIVA

Atualmente existem duas máquinas virtuais em operação como servidores de *proxy* no Paraná Digital, cada uma executando dez instâncias do *software* Squid. Os servidores não rodam

apenas uma instância, pois, há algum tempo, existiam quatro máquinas físicas como *proxies* que apresentaram problemas de desempenho, onde o tempo de resposta era elevado, tempo esse que chegava a até 2 minutos, e pelas análises feitas a única instância do Squid que existia em cada um dos servidores não consumia todo o poder de processamento disponível do *hardware*. No momento em que as outras duas instâncias foram executadas, através de uma simples análise da carga da máquina e de um teste que realizava uma requisição nos *proxies*, constatou-se um enorme ganho de desempenho, com tempos de resposta muito inferiores, em geral, inferior a 5 segundos.

O cenário dos servidores de *proxy* do Paraná Digital se alterou desde que a proposta deste trabalho foi redigida. Na época, existiam as quatro máquinas físicas citadas no parágrafo anterior executando a função de *proxy*. Hoje existem apenas duas máquinas virtuais executando tal função, porém, com maior poder de processamento total em relação às antigas quatro máquinas físicas.

Como esse problema já aconteceu anteriormente, neste mesmo cenário, este estudo pretende dar uma solução mais aprofundada ao problema. Como é possível reparar, existiam já quatro servidores atuando como *proxies* e nenhum deles usava todo o poder de processamento disponível, levando essa situação a uma conclusão: algumas dessas quatro máquinas, se bem configuradas, poderiam nem mesmo ser necessárias, partindo-se do observado que as máquinas atenderam à demanda com folga quando mais instâncias do Squid foram iniciadas. Ou seja, existe a hipótese de haver recursos financeiros investidos desnecessariamente em *hardware*.

O tipo de estudo proposto é hoje pouco explorado e há poucas referências na literatura. De uma forma geral, a grande maioria dos servidores são de pequeno porte, ou seja, possuem um número de requisições muito baixo para um determinado período de tempo, o que torna ineficiente o custo com mão-de-obra especializada para deixá-los ainda mais velozes. Com o crescimento da Internet, esse tipo de estudo vem tornando-se cada vez mais importante e inevitável para prover serviços que respondam quase que instantaneamente, em geral na casa dos milisegundos, afinal, hoje, a característica mais desejada da Internet é agilidade. Um caso real que se pode ilustrar é o da ferramenta de buscas do Google, que usa mais de 15 mil PCs, com um tipo de tecnologia chamada de *cluster* (ou agrupamento) (BARROSO; DEAN; HÖLZLE, 2003) para realizar o processamento das buscas feitas pelos usuários em seus gigantescos bancos de dados.

No caso citado acima, muitas pesquisas devem ter sido feitas para se chegar a um nível de desempenho em que toda a capacidade de *hardware* fosse utilizada adequadamente. Devido aos altos custos com *hardware*, as empresas têm optado por realizar melhorias no desempenho

do sistema computacional através de *software*, já que não existe custo envolvido, a não ser com mão-de-obra especializada. Apesar da necessidade da contratação de pessoal especializado para executar tal tarefa, este acaba sendo muito menos significativo que o custo com *hardware*, visto que não se pode esquecer que com o aumento no parque de máquinas há também o aumento de consumo de energia (que acaba gerando um custo que não se pode desprezar).

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Pesquisar e documentar a configuração ideal de *software* para a infraestrutura já existente no Paraná Digital, objetivando adquirir o máximo desempenho de *hardware* e rede, evitando a aquisição desnecessária de novos componentes de *hardware*, tanto para o serviço de *proxy*, quanto para os equipamentos transparentes aos usuários, como *switches* e *firewalls*.

1.3.2 Objetivos Específicos

- Estudar as principais formas de melhorar o desempenho dos sistemas operacionais propostos, para uso com serviços de *proxy*, visando obter o máximo dos recursos da máquina e evitando consumo de processamento desnecessário;
- Reduzir a alocação desnecessária de memória RAM, além de evitar a utilização de espaço em disco ou a constante consulta dos dados deste, a fim de possibilitar o envio das informações solicitadas pelo usuário num período de tempo desprezível;
- Monitorar o desempenho dos servidores quando possível, realizando estatísticas numéricas ou gráficas;
- Estabelecer registros para serem utilizados comparativamente entre si, permitindo obter análises de impacto individuais para cada mudança efetuada no sistema;
- Descrever os processos de maior relevância estudados durante o período de elaboração deste trabalho, para que a comunidade de *software* livre possa entender, utilizar e contribuir para o método pesquisado.

1.4 METODOLOGIA DE PESQUISA

Primeiramente, foram coletadas informações em livros específicos que tratam do tema em questão, visando entender de forma mais aprofundada como trabalha o gerenciamento de re-

cursos do sistema operacional, realizando um levantamento teórico de como as tecnologias envolvidas funcionam em mais baixo nível, ou seja, quais os métodos construtivos usados e como tirar o melhor proveito destes.

Em seguida, uma pesquisa foi realizada, especialmente na Internet, buscando por relatos de outras pessoas com problemas similares de desempenho (para servidores *proxy* ou não), e qual foi o caminho seguido para se chegar a uma conclusão que permitisse realizar melhorias ao sistema em questão. Essa pesquisa teve objetivo não só de levantar quais os principais aspectos do sistema que tiveram de ser modificados, mas também de encontrar ferramentas que podem auxiliar no processo, ferramentas que possam, de forma direta ou indireta, mostrar as deficiências dos serviços e como melhorá-las.

Quando já existia uma significativa gama de ferramentas, relatos e documentação, capazes de auxiliar no desenvolvimento do projeto, foram iniciados os testes práticos nos servidores, tomando como base as principais alterações sugeridas no material encontrado, bem como os dados estatísticos que foram gerados durante o processo.

Por fim, foi iniciada a implementação dos métodos, que se mostraram capazes de gerar os resultados esperados. Neste ponto foi necessário que uma configuração específica fosse desenvolvida, procurando melhor atender ao Projeto Paraná Digital. Este é um dos pontos críticos do estudo, pois uma correta configuração é o que define o quão elevado se tornará o desempenho, levando em consideração capacidades de processamento de cada máquina, áreas de armazenamento (no caso, apenas armazenamento temporário, ou seja, memória RAM) e também taxas de transferência da rede, já que esse será o processo final do projeto, considerando que outras modificações que possam ser realizadas, não terão influência positiva, ou então, não tenham influência significativa no serviço final oferecido.

2 RECURSOS UTILIZADOS

As principais fontes de pesquisa utilizadas foram, além dos manuais do *software*, livros, *websites* especializados, artigos e relatos de problemas e soluções encontrados na mesma área de pesquisa (no caso, o funcionamento mais aprofundado dos sistemas operacionais e do Squid) e também documentações relacionadas à melhoria da performance de todo o sistema, bem como de seu monitoramento.

2.1 AS LICENÇAS DE *SOFTWARE* LIVRE

Antes de iniciar a análise técnica, é importante citar alguns conceitos importantes da GPL, a General Public License (ou Licença Pública Geral). A GPL foi desenvolvida por uma fundação chamada Free Software Foundation (Fundação para o Software Livre), abreviada por FSF, iniciada em 1983 por um cientista da computação, chamado Richard M. Stallman (FREE SOFTWARE FOUNDATION, 2008).

O principal objetivo da GPL e da FSF é a de legalizar e popularizar o conceito de *software* livre. A definição de *software* livre refere-se a quatro tipos de liberdade, sendo elas (FREE SOFTWARE FOUNDATION, 2007):

- A liberdade de executar um programa, para qualquer propósito (liberdade 0);
- A liberdade de estudar como o programa funciona, e adaptá-lo às suas necessidades (liberdade 1). Acesso ao código fonte é um pré-requisito para isso;
- A liberdade de redistribuir cópias para que você possa ajudar ao próximo (liberdade 2);
- A liberdade de melhorar o programa e lançar suas melhorias (e versões modificadas em geral) para o público, para que toda a comunidade se beneficie (liberdade 3). Acesso ao código fonte é um pré-requisito para isso.

Como serão também estudados os sistemas operacionais da família BSD (Berkeley Software Distribution), é necessário também citar que os mesmos não são licenciados pela GPL,

mas sim pela licença BSD, que, apesar de também classificar os aplicativos como *software* livre, tem como principal diferença da GPL a possibilidade da incorporação desses códigos em produtos comerciais proprietários, o que é muito comum, como é o caso do Mac OS X (APPLE INC., 2007), sistema operacional desenvolvido e comercializado pela Apple Inc.

Basicamente, pode-se então considerar um programa como *software* livre, quando se possui o acesso ao seu código fonte e a permissão para utilizá-lo, modificá-lo para qualquer propósito, além de redistribuí-lo para qualquer pessoa livremente. Nesta pesquisa será utilizado unicamente *software* que possua esta natureza.

2.2 LINUX

O exemplo mais comum e mais conhecido de *software* livre hoje em dia é o Linux, do qual todas as pessoas ligadas à área de tecnologia da informação já devem, ao menos, ter ouvido falar.

O Linux é um clone do sistema operacional Unix, projetado por Linus Torvalds com a assistência de um time livre e unido de *hackers* através da Internet. O sistema visa conformidade com os padrões POSIX e Single UNIX Specification (LINUX KERNEL ORGANIZATION, 2008).

O sistema operacional Linux, na realidade, é composto apenas de seu *kernel* (ou núcleo). O *kernel* é o principal componente de um sistema operacional, que funciona como uma interface entre os aplicativos e o *hardware*. Esses aplicativos é que tornarão o sistema útil para o usuário final, funcionando então como uma interface entre o usuário e o *kernel* do Linux.

Existem atualmente dezenas de distribuições do Linux. Uma distribuição Linux, muitas vezes abreviada por “distro”, nada mais é do que um conjunto de aplicativos que funcionam sobre o *kernel* do Linux, podendo ser para uso geral, isto é, com diversos aplicativos para todos os fins, ou então, aplicativos para uso específico. Como exemplos de distribuições de uso geral estão o Debian, o Slackware e o Red Hat, e como exemplos de distribuições de uso específico, existem o SystemRescueCd, o Debian Med e o Linux in Schools. Essas distribuições podem ainda possuir aplicativos próprios (que podem não funcionar se instalados em outra “distro”), criados pelos seus desenvolvedores, normalmente, para instalar, remover e gerenciar os aplicativos nela instalados.

Apenas para exemplificar as diferenças entre distribuições de uso específico, abaixo estão as descrições das três distribuições deste tipo, citadas anteriormente.

O SystemRescueCd é um sistema Linux inicializável através de um CD-ROM que tem a finalidade de realizar reparos do sistema e recuperar dados da máquina após a ocorrência de algum problema. Ele pretende prover uma maneira fácil para executar tarefas de administração no computador, como criar e editar partições do disco rígido. Além disso, possui inúmeras ferramentas e utilitários básicos para profissionais da área (SYSTEMRESCUECD, 2009).

O Debian Med é um sistema Linux baseado na distribuição Debian, adequado às necessidades da prática e pesquisa médica. Além disso, o Debian Med é um sistema completo para todas as atividades médicas, constituído completamente de *software* livre (SOFTWARE IN THE PUBLIC INTEREST, INC., 2009).

O projeto K12Linux é um sistema que utiliza o LTSP (Linux Terminal Server Project), baseado na distribuição Linux chamada Fedora 10, que pode ser distribuído e instalado através de um *pendrive* USB ou um DVD. Desde 1999, o LTSP tem sido utilizado em muitas escolas e empresas com servidores baseados em Linux e *thin clients* (terminais sem disco rígido que utilizam a rede local como meio de comunicação com um servidor concentrador do sistema operacional e das informações pessoais, antigamente conhecidos como terminais gráficos), permitindo que máquinas de baixo custo ou muito antigas tornem-se estações de trabalho de última geração. Nesse sistema as estações de trabalho conectam-se a um servidor central, através do qual podem executar diversos aplicativos necessários para o dia-a-dia dos usuários, como interfaces gráficas, navegadores de Internet e editores de texto e planilhas eletrônicas (K12LINUX, 2009).

2.3 SQUID

A função do Squid, quando do seu desenvolvimento, era de fazer *cache* de páginas remotas na rede local (no caso, no próprio servidor *proxy*). O *cache* nada mais é do que uma cópia das páginas mais frequentemente acessadas pelos clientes (usuários) da rede local. O objetivo dessa cópia é evitar que toda vez que um cliente queira acessar uma página externa (normalmente da Internet) exista consumo da banda da ligação entre a rede local e a rede externa. Portanto, uma vez que um usuário acessa uma página remota, esta é armazenada no servidor *proxy* para evitar que haja utilização da ligação com a rede remota quando outro usuário acessá-la novamente.

O Squid possui também outras finalidades, como previnir usuários de visitar páginas não apropriadas (especialmente no trabalho ou escola), garantir que apenas usuários autorizados possam acessar a Internet, aumentar a privacidade dos usuários filtrando determinadas informações nas requisições, e até mesmo reduzir a carga em servidores *web* da rede (WESSELS,

2004).

Todas as funcionalidades que o Squid fornece, como ocorre em qualquer *software*, consomem recursos da máquina. Por isso, todas as funcionalidades utilizadas devem ser levadas em consideração na realização da análise de desempenho e em seu aperfeiçoamento.

2.3.1 Filtros para o Squid

A solução de controle de acesso do Squid é muito versátil, permitindo até que agentes externos atuem na filtragem. A função principal do Squid sempre foi a criação e manutenção de *cache* na rede local. Entretanto, ele provê funcionalidades que podem nem sempre ter um desempenho muito favorável ou, muitas vezes, podem não atender às necessidades de um determinado usuário.

Devido a esse tipo de problema, muitos desenvolvedores decidiram criar novas ferramentas para trabalhar em conjunto com o Squid. Entre essas ferramentas criadas por terceiros está o SquidGuard, que promete fornecer filtros de conteúdos com desempenho significativamente superior ao Squid.

O SquidGuard é um redirecionador de endereço URL que se utiliza de listas negras para efetuar o bloqueio de *websites* impróprios para visualização em determinado ambiente. Ele é utilizado em conjunto com o aplicativo de *proxy* Squid. Segundo seus próprios desenvolvedores, ele possui duas grandes vantagens: é rápido e é *software* livre (SHALLA SECURE SERVICES, 2009).

Aplicativos como o SquidGuard serão também utilizados, executando funções que o Squid pode não executar da maneira mais adequada, tornando o sistema ainda mais eficaz.

2.4 MONITORAMENTO

É muito comum em grandes empresas que os principais serviços e equipamentos possuam algum tipo de monitoramento. Na área de tecnologia da informação isso não é diferente. Os tipos de monitoramento mais comuns são: o monitoramento de desempenho e o de disponibilidade. O desempenho costuma ser monitorado para prever possíveis falhas, permitindo que os técnicos possam corrigí-las antes de acontecerem, podendo ser considerado como manutenção preventiva. Já o monitoramento de disponibilidade é utilizado para verificar erros, ou seja, identificar e alertar quando um serviço para de funcionar. Esses dois tipos podem servir tanto para identificar falhas de *hardware* (algum componente danificado), como para identificar falhas de

software (um *bug* em algum dos aplicativos, falta de espaço em disco, etc.).

Para esse trabalho, ferramentas de monitoramento que forneçam gráficos da utilização dos recursos das máquinas serão necessárias. Os gráficos serão as informações mais importantes na realização de comparativos de desempenho durante as pesquisas.

O monitoramento realizado deverá abranger os principais recursos utilizados da máquina, como o consumo da capacidade do processador, da memória RAM, do espaço em disco e da banda de rede, além do tempo de resposta médio para o carregamento das páginas, número de requisições por determinado período de tempo, quantidade de objetos em *cache* (páginas, imagens, etc.), dentre outros parâmetros que poderão ser julgados convenientes para tais análises.

Uma ferramenta já existente na rede atual, e que será também utilizada no desenvolvimento do projeto, é o Cacti. Trata-se de uma ferramenta que coleta as informações dos recursos citados anteriormente, dos servidores *proxy* da rede e os reporta através de um protocolo UDP chamado SNMP.

2.4.1 Cacti

O Cacti é uma solução completa de criação de gráficos de rede que utiliza o aplicativo RRDTool para armazenar os dados obtidos e criar gráficos baseados nesses dados. Possui também um rápido coletor de dados, modelagem avançada de gráficos, vários métodos de aquisição de dados e ferramentas de gerenciamento de usuário, com capacidade para permitir que esses tenham acesso apenas às informações que lhes dizem respeito. Todas essas funcionalidades estão integradas em uma interface simples de usar, que permite fácil administração tanto para redes pequenas quanto para redes com centenas de dispositivos (THE CACTI GROUP, 2009).

O Cacti, assim que instalado, já vem com uma enorme quantia de modelos prontos para facilmente iniciar o processo de coleta de dados dos principais recursos das máquinas que seguem o modelo Unix (como o Linux e os sistemas da família BSD).

Além dos modelos já existentes, o usuário pode criar outros novos que atendam as necessidades específicas de um sistema, desde que eles forneçam tais informações através do protocolo SNMP. O próprio Squid possui, nativamente, dezenas de informações que podem ser coletadas através de tal protocolo, as quais serão também utilizadas como parâmetros para a análise de desempenho do sistema.

2.4.2 Munin

O Munin é uma ferramenta de monitoramento de recursos de rede que pode ajudar a analisar tendências de recursos e problemas do tipo "o que ocasionou a redução do desempenho?". Foi projetado para ser *plug and play*, em português, ligar e usar. Uma instalação padrão fornece muitos tipos de gráficos sem muito trabalho (COMUNIDADE MUNIN, 2010).

Assim como o Cacti, o Munin já possui vários modelos pronto para coletar informações dos principais recursos das máquinas que seguem o modelo Unix.

Essa ferramenta será utilizada também para coletar informações às quais o Cacti não fornece suporte nativo.

2.4.3 Protocolo SNMP

O SNMP (Simple Network Management Protocol) é um protocolo da camada de aplicação (segundo o modelo OSI) que tem o objetivo de facilitar a troca de informações de gerenciamento entre dispositivos de rede. Ele é parte do conjunto de protocolos TCP/IP (Transmission Control Protocol/Internet Protocol). O SNMP permite que administradores de rede gerenciem o desempenho de rede, descubram e resolvam problemas na rede e planejem sua expansão (CISCO SYSTEMS INC., 2003).

Este protocolo, possibilita que qualquer equipamento de rede, através de um identificador de objeto (OID), colete informações relacionadas aos vários parâmetros do *software* instalado, podendo o *software* fornecer dados de como ele está se comportando ou de como o *hardware* está se comportando em relação a ele.

O único inconveniente do SNMP é que o *software* que coleta as informações deve ter previamente a capacidade de interpretá-las, ou seja, o *software* coletor pode coletar inúmeros objetos, porém, só poderá tirar proveito daqueles que saiba interpretar, inutilizando os demais.

2.5 REFINAMENTO DE DESEMPENHO

A primeira pergunta que surge neste tópico é “Por que refinar um sistema?”. Quando um aplicativo é distribuído para seus clientes, ele deve ser compatível com a maioria dos computadores disponíveis, além de necessitar compatibilidade com outros aplicativos que podem estar presentes em paralelo com este. Essas são duas condições altamente heterogêneas, pois existem dezenas de fabricantes de *hardware* e *software* que produzem produtos com as mais diferentes características. Então os desenvolvedores e distribuidores destes aplicativos optam

por deixá-los sistemas genéricos, para que possam ser executados corretamente em quase todos os cenários possíveis, o que não é possível com uma distribuição pré-refinada (APESTEGUIA, 2006).

Refinar o sistema não é uma tarefa simples, já que normalmente os sistemas não são fornecidos "pré-refinados" pelos desenvolvedores, devido às várias formas de se operar um determinado *software*.

A grande maioria dos sistemas operacionais atuais já fornecem uma série de ferramentas capazes de informar ao administrador o estado atual da máquina e dos aplicativos que estão sendo executados. Portanto, essas ferramentas serão muito úteis para realizar o refinamento do sistema, já que foram desenvolvidas justamente para mostrar ao administrador o que está causando a redução do desempenho do sistema. Ferramentas de terceiros também podem ser utilizadas para avaliar essas características, como por exemplo, o Cacti.

2.6 CONFIGURAÇÕES DE *HARDWARE*

Na execução dos procedimentos que ocorrerão nos próximos capítulos serão utilizadas duas *blades* do fabricante DELL, modelo PowerEdge M600, cada uma com a seguinte configuração de *hardware*:

- 2 processadores Intel E5410 de 4 núcleos e *clock* de 2,33 GHz;
- 16 GB de memória RAM;
- 2 discos SAS Seagate, modelo ST973402SS, de 73,4 GB de capacidade, 10000 RPM e taxa de transferência de 3 Gbps, utilizando RAID 1 (espelhamento);
- 2 interfaces de rede Broadcom NetXtreme II BCM5708, com taxa de transferência de 1 Gbps.

Cada *blade* é utilizada como máquina hospedeira para máquinas virtuais com o Xen Hypervisor ou o KVM e para cada um dos dois servidores de *proxy* é criada uma máquina virtual com as seguintes características:

- 4 núcleos;
- 8 GB de memória RAM;
- 5 GB de espaço livre para a instalação do sistema operacional;

- 35 GB de espaço livre para armazenamento do cache, *logs* do proxy e de toda a máquina virtual.

O *hardware* dos servidores utilizados antes da migração dos serviços de *proxy* para as duas *blades* era o seguinte:

- 1 processador AMD Opteron 242 com *clock* de 1,6 GHz;
- 2 GB de memória RAM;
- 2 discos SCSI Seagate, modelo ST336607LW, de 36,7 GB de capacidade, 10000 RPM e taxa de transferência de 2560 Mbps, utilizando RAID 1 (espelhamento);
- 2 interfaces de rede Broadcom NetXtreme BCM5704, com taxa de transferência de 1 Gbps.

3 *CACHE*

Os *proxies* têm como uma de suas funções a criação e alimentação de um *cache*, ou seja, um repositório com os arquivos de páginas já acessadas por algum usuário da rede. Esses arquivos podem ser as próprias páginas estáticas, imagens e até mesmo os arquivos que foram baixados, bem como documentos de texto, planilhas eletrônicas, músicas e qualquer outro tipo de arquivo que necessita de um aplicativo específico para ser aberto no computador do cliente. Devido a essa função dos *proxies*, à medida em que o *cache* vai crescendo, o número de arquivos e o consumo de espaço em disco aumentam.

3.1 CONFIGURAÇÃO DE *SOFTWARE* PARA OS TESTES REALIZADOS

- Xen Hypervisor versão 3.2;
- Debian GNU/Linux versão 5.0 (codinome Lenny), utilizando o *kernel* 2.6.26 compilado com suporte ao Xen;
- Squid versão 2.7 STABLE3.

3.2 SISTEMAS DE ARQUIVOS

Quanto mais o *cache* cresce, a tendência é de que se leve mais tempo para procurar os arquivos quando requisitados. Por motivos como esse, hoje existe uma infinidade de sistemas de arquivos, cada um feito para resolver um tipo de problema específico. Há sistemas de arquivos que foram desenvolvidos para se obter maior velocidade com arquivos pequenos, outros para maior velocidade de gravação ou leitura, alguns que foram criados para aproveitar ao máximo o espaço em disco, com tamanhos de bloco variáveis que não desperdiçam espaço quando se possui muitos arquivos pequenos e assim por diante.

O sistema de arquivos que será buscado nessa seção será o que possua maior velocidade de leitura com arquivos relativamente pequenos, para até 50 KB, tamanho suficiente para a maioria

das páginas *web* e logotipos da atualidade. Essas características são apropriadas para o caso em questão, pois a grande maioria dos arquivos que ficam armazenados em um cache são muito pequenos como páginas estáticas como uma notícia, que não deve passar de alguns poucos quilobytes e imagens, que na sua maioria não passam de algumas dezenas de quilobytes, como *banners* ou logomarcas. A maior velocidade de leitura é importante porque o objetivo do cache é evitar o desperdício de banda de saída para a Internet, armazenando na rede local as páginas mais acessadas, como portais de notícias, e isso necessita apenas uma única gravação no disco, que poderá ser acessada centenas ou até milhares de vezes num único dia pelos computadores da rede local.

3.2.1 XFS vs. ReiserFS

Atualmente, dois dos sistemas de arquivos mais comuns utilizados com o Linux são o XFS e o ReiserFS. Portanto, os dois serão os sistemas de arquivos escolhidos para a realização dos testes de desempenho com o Squid, levando-se em consideração a utilização prévia desses mesmos dois sistemas de arquivos nos serviços da rede do Paraná Digital.

Pretende-se nesta seção executar uma comparação entre os dois sistemas, coletando os dados de como a carga de Entrada/Saída (I/O) do sistema se comporta nos vários horários do dia em que os *proxies* são utilizados, especialmente aqueles horários em que a carga atinge seu pico.

Como principais características do XFS pode-se citar sua rápida recuperação em caso de interrupções inesperadas, devido à tecnologia de *journaling*, buscas e alocações de espaço rápidas, implementação de 64 bits, permitindo sistemas de arquivos de tamanhos na ordem de exabytes, eficiência em gerenciamento de espaço com o uso de mecanismos de alocação de blocos de tamanhos variáveis e capaz de fornecer desempenho próximo do limite de entrada e saída do dispositivo (SGI, 2010).

O ReiserFS tem como principais características o *journaling* rápido, excluindo a necessidade de verificações frequentes do sistema de arquivos, árvores balanceadas, que permitem que mais de cem mil arquivos existam em um único diretório sem grande perda de desempenho e a eficiência de utilização de espaço em disco, por meio da alocação de espaço variável de *inodes* (NAMESYS, 2007).

3.2.2 Testes e Comparativos de Desempenho

Para realizar os comparativos e verificar o comportamento real do sistema, uma das máquinas foi mantida com o sistema de arquivos XFS (padrão atual) e a outra foi formatada com o ReiserFS. É importante ainda deixar claro que neste capítulo e em todo o restante do trabalho serão sempre usadas dez instâncias do Squid em ambos os servidores.

Além da modificação do sistema de arquivos, após alguns dias de observação sobre os dados e gráficos obtidos foi possível perceber que essa alteração não proporcionou nenhum ganho ou perda efetivos do ponto de vista da velocidade de leitura, que é o objeto de estudo nesse capítulo (a gravação não é objeto de estudo neste caso). Como se pode observar nos gráficos das Figuras 1 e 2.

A partir desse momento e após cuidadosas observações dos sistemas, tornou-se possível identificar a causa da modificação não ter realizado nenhuma mudança efetiva no desempenho, caso que será descrito adiante. A ferramenta mais importante nessa observação foi o Iotop, *software* que avalia cada um dos processos do sistema, mostrando a taxa de leitura e gravação em disco no formato de uma tabela.

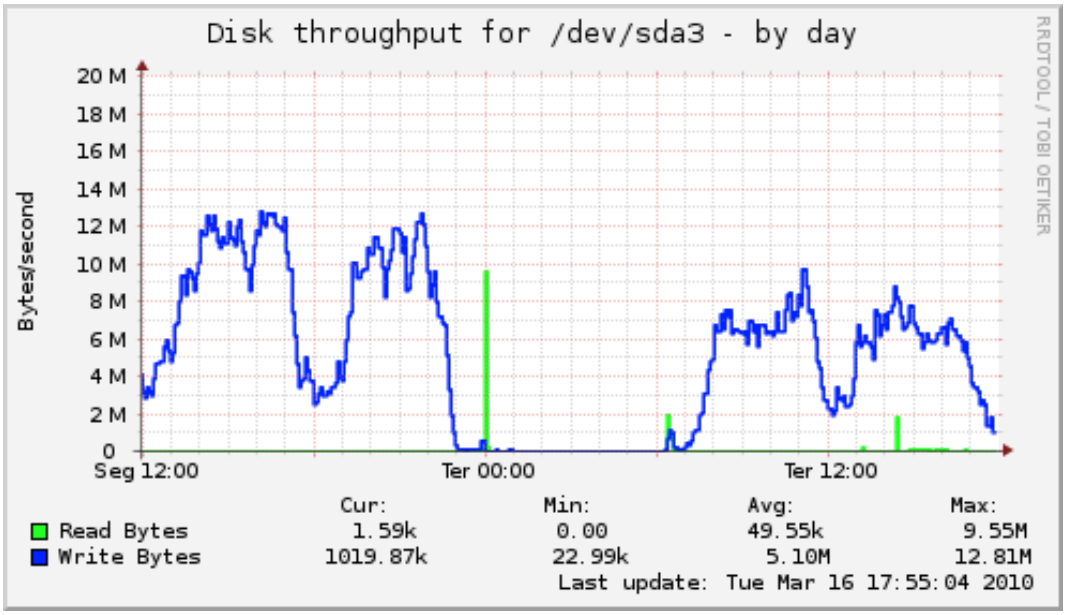
Como pode-se observar na Figura 3, a taxa de gravação em disco é alta em relação à taxa de leitura, devido principalmente ao armazenamento de *logs*[1]. Apesar da alta taxa de gravação, a taxa de leitura em disco se mantém quase nula, com média em torno de 40 KB/s, como mostram as Figuras 1 e 2, justamente a taxa que se esperava que tivesse um alto consumo nos discos.

A situação observada com o Iotop leva à provável conclusão de que o *cache* em disco não está sendo utilizado, ou, ao menos, seu acesso está muito abaixo do esperado. A maneira mais eficiente de se descobrir se o *cache* está efetivamente em uso é através das análises dos *logs* do Squid.

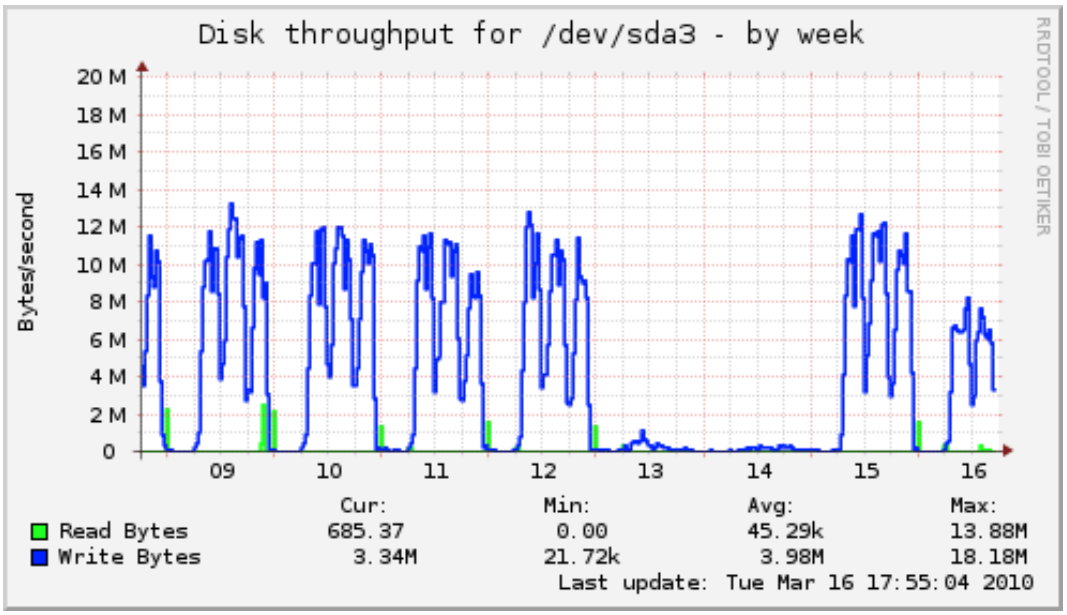
Fazendo uma busca nos *logs* do Squid das duas máquinas, foram analisados os códigos de identificação de estado do Squid chamados TCP_HIT e TCP_MEM_HIT, ocorrências de arquivos encontrados no cache localizado no disco e na memória respectivamente. Sempre que há um arquivo sendo acessado por algum usuário dos *proxies* que contém esse código de identificação, isso significa que o arquivo acessado foi encontrado no *cache* local, ou seja, não houve necessidade de buscá-lo na rede externa.

No primeiro dia, a situação contida na Tabela 1 foi encontrada.

[1] Os *logs* são informações armazenadas em disco sobre as ações executadas pelos processos, como o horário em que o processo foi iniciado ou parado, problemas que ocorreram durante sua execução e até as informações das páginas acessadas pelos usuários, no caso do Squid, por exemplo.



(a) Análise da janela de 30 horas

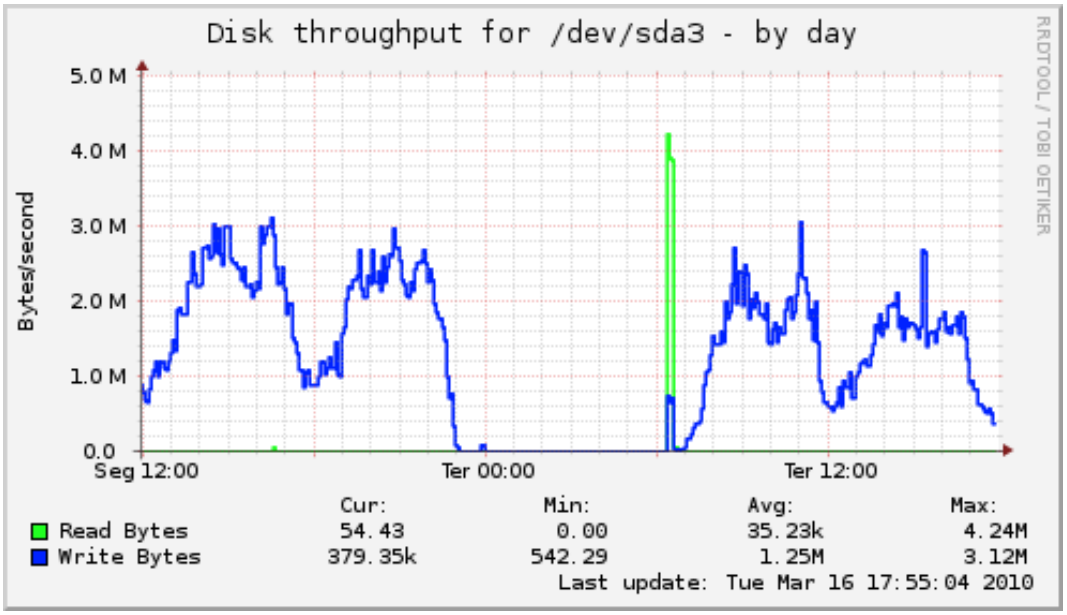


(b) Análise da janela de 7 dias

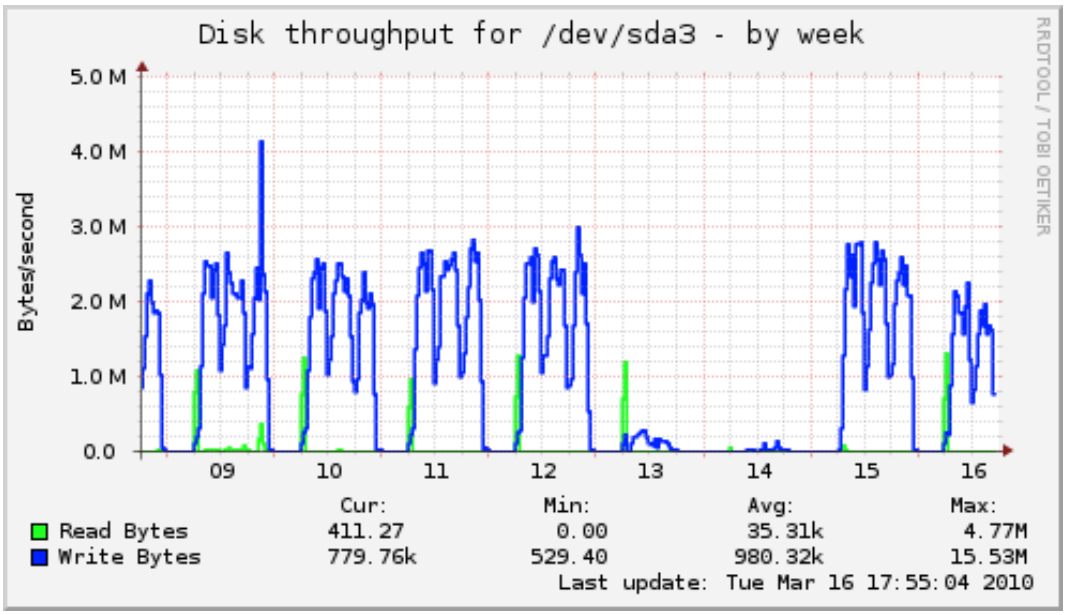
Figura 1: Taxa de transferência do disco do servidor *proxy* com XFS.

Em ambos os casos o número total de arquivos acessados no *cache* em disco chegou a aproximadamente 60% do total de arquivos acessados do *cache*.

No primeiro dia, a quantidade de memória reservada para *cache* em cada uma das instâncias era de 256 MB, ou seja, um total de 2560 MB por servidor e 1536 MB de espaço em disco para *cache* de cada uma dessas instâncias, totalizando 15360 MB. Sendo assim, o *cache* em disco possui 6 vezes (ou 600%) a capacidade de armazenamento daquele na memória, entretanto, a média da quantidade total de acessos ao *cache* em disco é superior em apenas 46%.



(a) Análise da janela de 30 horas



(b) Análise da janela de 7 dias

Figura 2: Taxa de transferência do disco servidor *proxy* com ReiserFS

O valor obtido para o *cache* em disco é baixo, considerando-se, que talvez esse valor pudesse ser facilmente reduzido ainda mais com pouco aumento do total de *cache* em memória para cada uma das dez instâncias do Squid existente nos dois servidores, possibilitando-se chegar à utilização do *cache* em disco próxima do nulo.

A média do tempo necessário para cada requisição foi calculada, utilizando-se o valor do "tempo decorrido" que é armazenado nos logs de acesso, como mostra o exemplo da Tabela 2, que será descrita com maiores detalhes ainda neste capítulo. Para o primeiro servidor *proxy*,

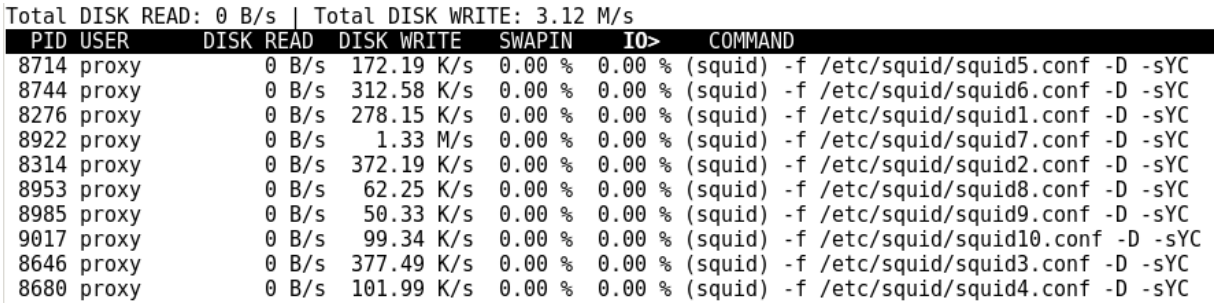


Figura 3: Taxa de Entrada/Saída com Iotop

Tabela 1: Quantidade de arquivos obtidos do *cache* no Squid no período de um dia, com 256 MB de *cache* em memória.

	Total de Acessos	TCP_MEM_HIT	TCP_HIT	Total de Arquivos em Cache Acessados	Percentual de Arquivos em Disco
Servidor 1	23.305.572	2.104.238	3.240.833	5.345.071	60,63%
Servidor 2	22.374.544	2.063.196	2.843.104	4.906.300	57,95%

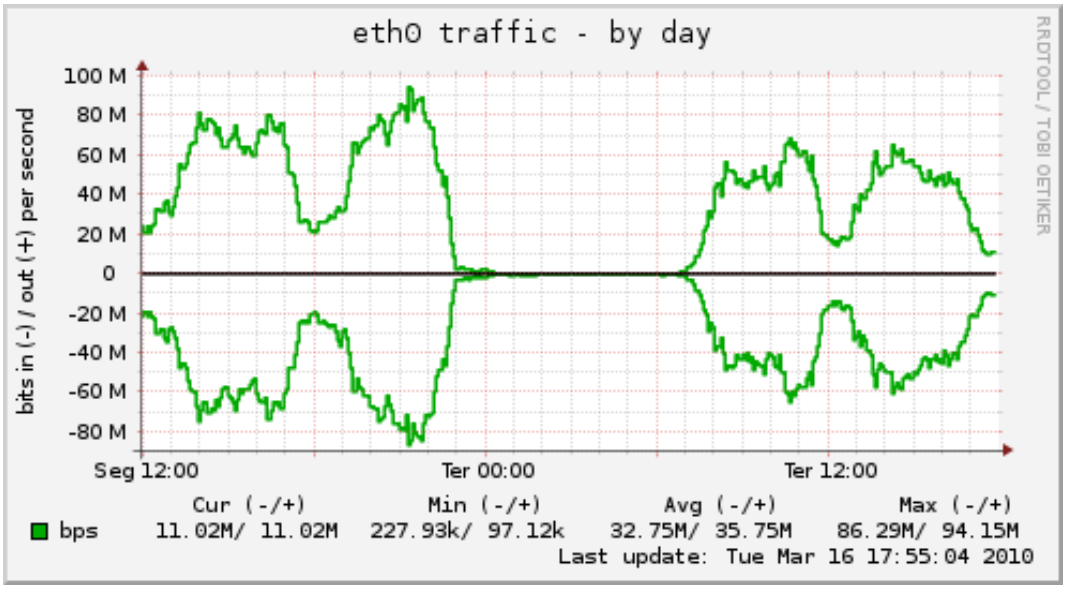
com o sistema de arquivos XFS, o tempo decorrido médio para os acessos ao *cache* em disco foi de 270 ms e para o outro servidor, com o sistema de arquivos ReiserFS, o tempo foi de 256 ms. O tempo decorrido médio foi muito similar em ambos os casos, não mostrando nenhuma vantagem ou desvantagem para a utilização de *cache* do Squid em disco com o XFS ou o ReiserFS, isto é, não há nenhuma grande diferença na escolha entre um deles.

No caso do *cache* em memória, o tempo decorrido médio foi de 13 ms no primeiro servidor, com o sistema de arquivos XFS, e de 20 ms no servidor com o sistema de arquivos ReiserFS e apesar de haver uma diferença relativamente grande (mais de 50%), o tempo é baixo em relação aos arquivos obtidos em disco, com uma diferença de quase 20 vezes no tempo decorrido médio.

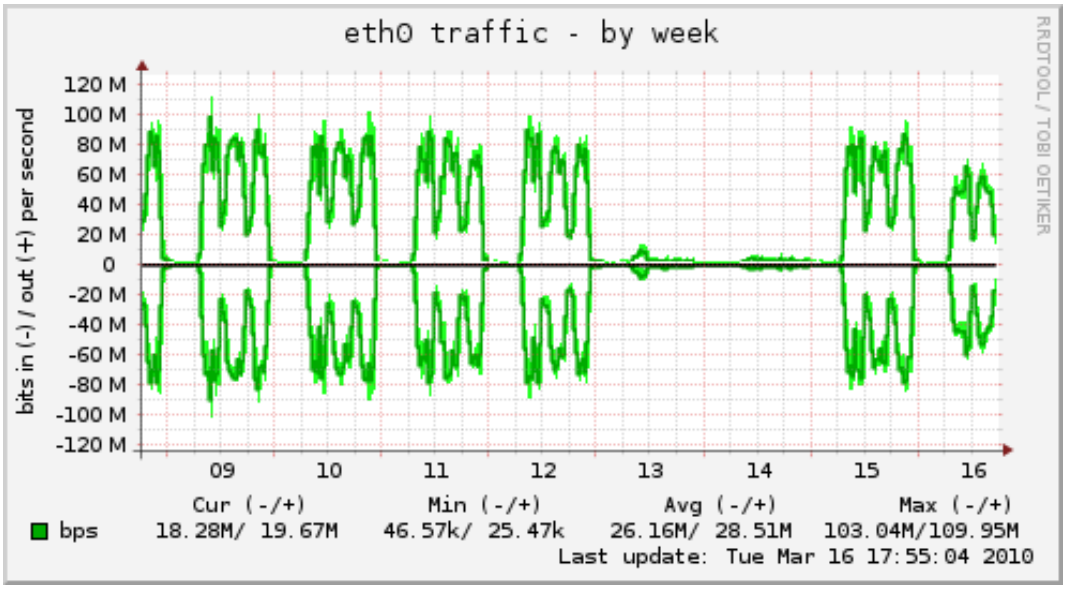
Durante dois dias foram coletados dados das principais informações do sistema, em especial, dos discos. Os dados coletados sobre os discos são a taxa de transferência (entrada e saída, quantidade de bytes escritos e lidos dos discos). Algumas informações do sistema como a taxa de transferência da rede e a utilização de CPU também serão usadas quando se considerar a veracidade dessas informações, ou seja, há de se considerar a carga real aplicada sobre os servidores.

Os gráficos da utilização da rede estão nas Figuras 4 e 5.

Como se observa nas Figuras 4 e 5, existe uma variação do servidor com sistema de arquivos XFS para aquele com ReiserFS, porém, como se trata de uma variação muito pequena, de aproximadamente 10%, tanto para entrada quanto saída do gráficos com janela de 30 horas e também dos gráficos com janela de 7 dias, esta será tratada como normal e irrelevante para os



(a) Análise da janela de 30 horas



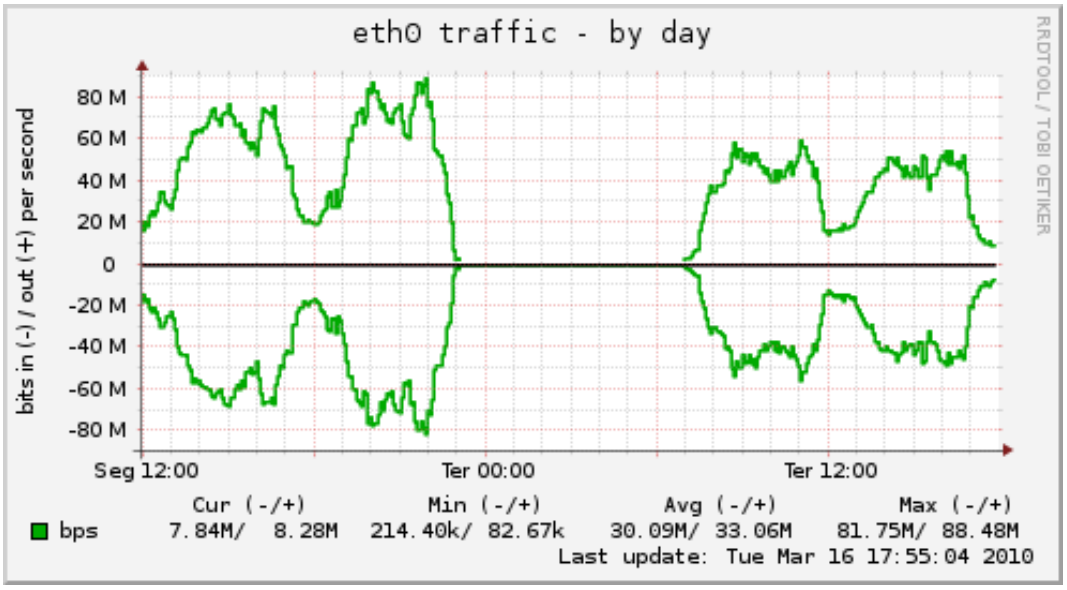
(b) Análise da janela de 7 dias

Figura 4: Consumo de banda de rede do servidor *proxy* com XFS.

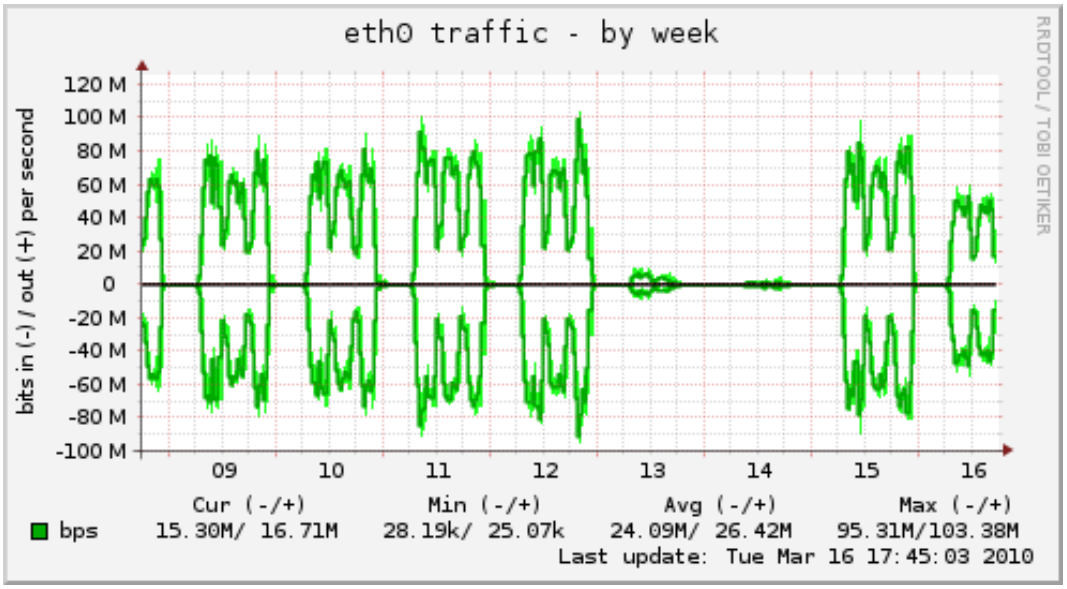
testes, por se tratar da variação natural, visto que o balanceamento entre os servidores não é por consumo, e sim pela origem dos acessos (IPs de origem), ou seja, trata-se de variação da ordem natural da necessidade humana instantânea.

A análise do consumo da banda de rede é apenas usada como forma de comprovar que a quantidade de dados trafegada é muito semelhante entre os dois servidores em questão.

Seguindo em frente com a análise, a mudança do sistema de arquivos utilizado não trouxe nenhum benefício direto. Entretanto, também não prejudicou outros recursos da máquina, como a utilização do processador, que pode ser observada nas Figuras 6 e 7.



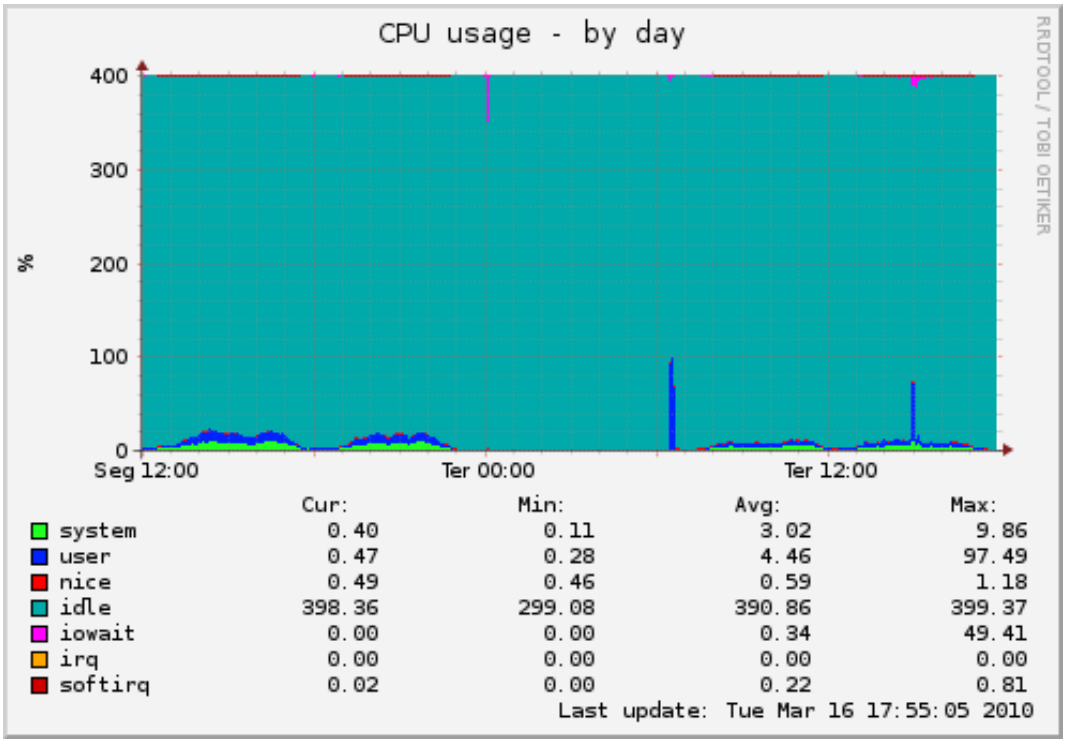
(a) Análise da janela de 30 horas



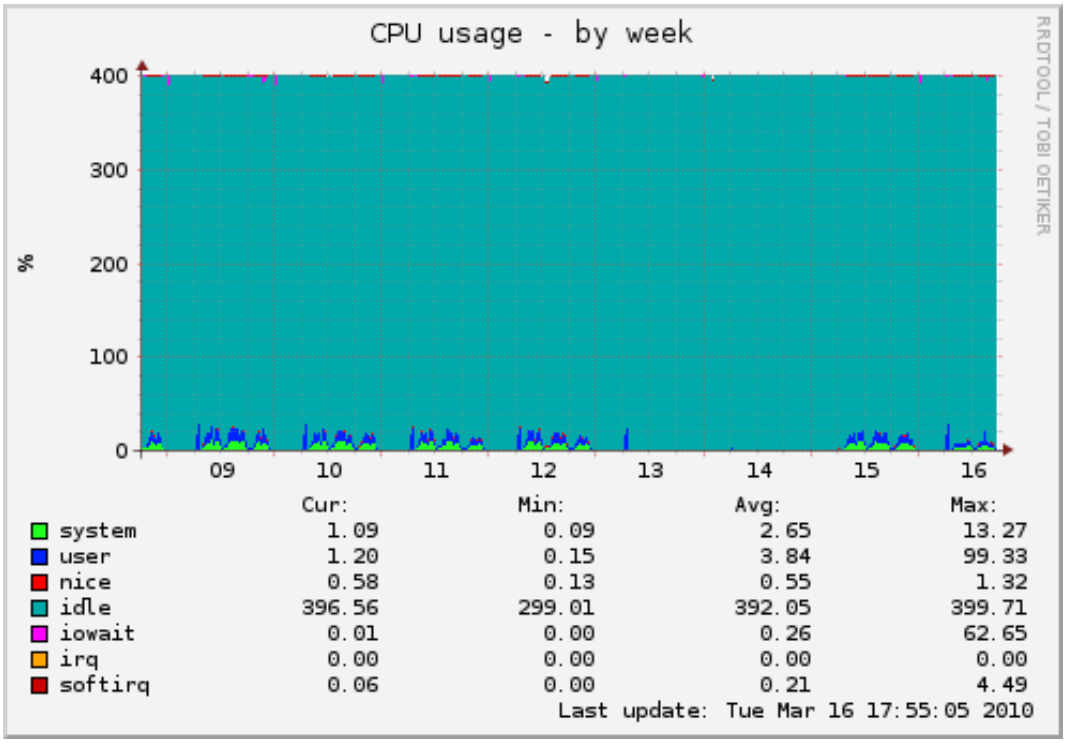
(b) Análise da janela de 7 dias

Figura 5: Consumo de banda de rede do servidor *proxy* com ReiserFS

Primeiramente, observa-se nas Figuras 6 e 7 que a escala do gráfico, especificamente no eixo das ordenadas vai de 0% até 400%, o que pode parecer incoerente a princípio. Entretanto, a explicação para essa escala é que 4 núcleos estão sendo utilizados na máquina virtual e a grande maioria do *software* hoje existente não trabalha nativamente com multiprocessamento (e esse é o caso do Squid 2.7), ou seja, cada instância utilizar-se-á de somente um núcleo para executar suas operações. Dessa forma, um único processo do Squid pode se utilizar de, no máximo, 100% desse total de processamento. Por este motivo, a escala vai chegar a 400% ao invés de apenas 100%, que poderia parecer mais lógico a princípio, ou seja, cada núcleo fornece 100% de processamento. Sendo assim, o percentual de processamento de uma máquina sempre



(a) Análise da janela de 30 horas

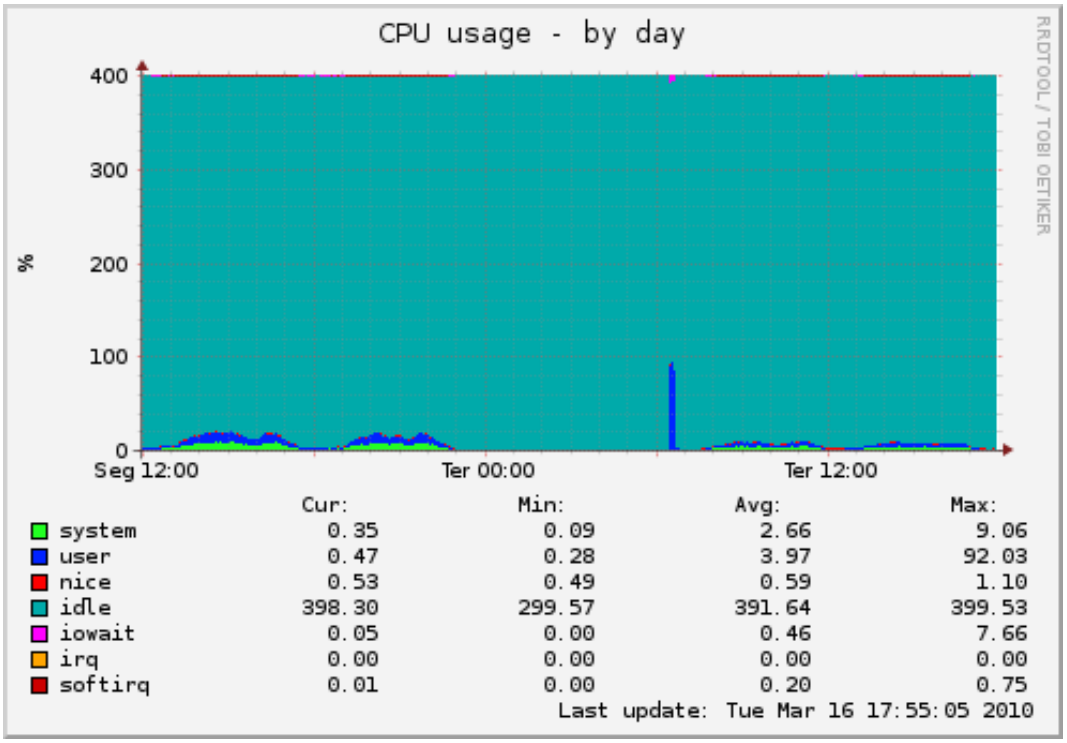


(b) Análise da janela de 7 dias

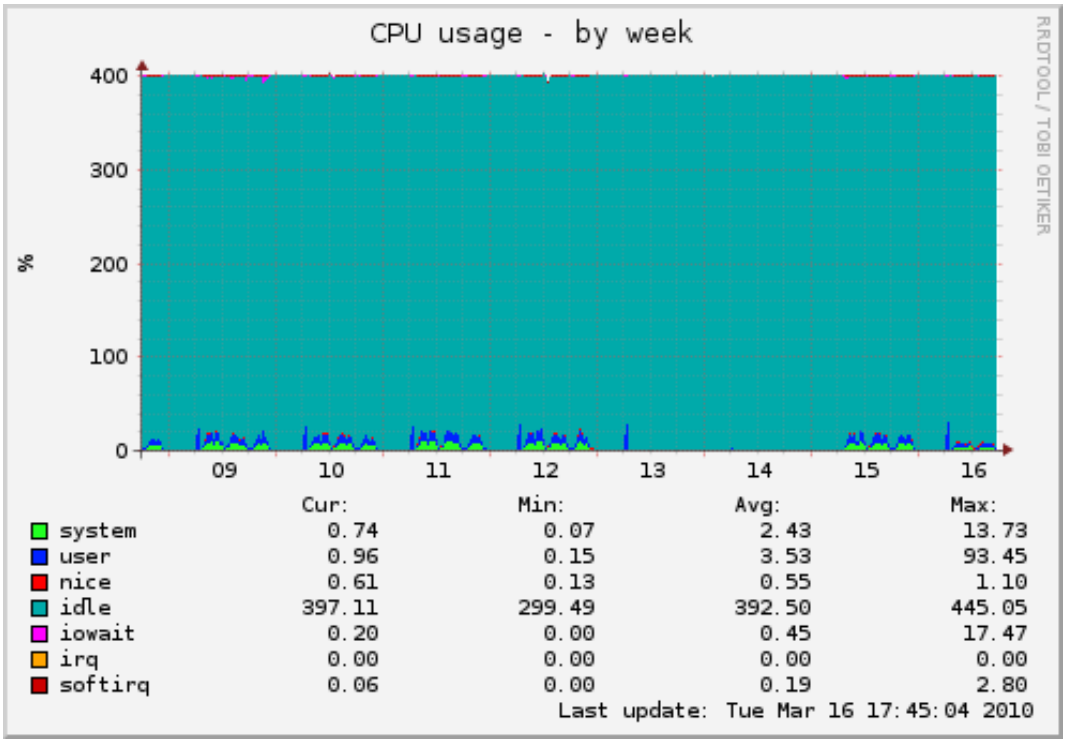
Figura 6: Percentual de consumo de processamento do servidor *proxy* com XFS.

será um múltiplo de 100%, sendo neste caso, 400%.

Voltando à análise, percebe-se que existem diversos estados, como *system*, *user*, *idle*, e as-



(a) Análise da janela de 30 horas



(b) Análise da janela de 7 dias

Figura 7: Percentual de consumo de processamento do servidor *proxy* com ReiserFS.

sim por diante. Está fora do escopo deste trabalho abordar todos eles e o que cada um deles significa, mas para as necessidades do trabalho, utilizar-se-á o estado *idle* (ou ocioso, em português) para os comparativos. O estado *idle* identifica a quantidade de processamento ainda

disponível na máquina, ou seja, sem uso naquele dado instante, apenas aguardando que uma tarefa lhe seja atribuída.

Observa-se nas Figuras 6 e 7 que o estado *idle* para ambos gira em torno de 391 para a janela de 30 horas e 392 para a janela de 7 dias, o que mostra que não há variação significativa também neste recurso do sistema, pois, ambas as máquinas virtuais são idênticas no quesito *hardware*.

Infelizmente, os gráficos de algumas figuras do trabalho podem não possuir visibilidade suficiente para tirar conclusões precisas, como é o caso das Figuras 6 e 7. Nestes casos os valores existentes no rodapé dessas serão suficientes para prosseguir com suas análises.

Apenas para se chegar a uma conclusão de qual sistema de arquivos seria mais eficaz para os *proxies*, o espaço reservado em memória para *cache* a partir desse ponto será reduzido para apenas 10 MB (total de 100 MB, quantidade próxima do desprezível, levando-se em consideração a quantidade de informações existentes na Internet nos dias de hoje) e o espaço em disco reservado permanecerá em 1536 MB por instância.

Como pode-se visualizar nos gráficos de taxa de transferência das Figuras 1 e 2 dos servidores, há uma grande taxa de gravação nos disco, representada pela linha azul, entretanto, a taxa de leitura em disco permanece praticamente nula.

Com a observação dos gráficos conclui-se que a leitura em disco é muito baixa, tão baixa que chega próxima de ser nula. Apesar disso, pode-se justificar facilmente essa baixa taxa de leitura do disco com o *log* da Tabela 2. (Os IPs dos clientes e as URIs de destino não serão mostrados por completo para garantir a privacidade da rede e dos usuários.)

Descrição das colunas:

- *Timestamp* (também conhecido como “Unix Time” ou “POSIX Time”, ou, horário do Unix/POSIX): valor em segundos desde as 0 horas, 0 minutos e 0 segundos do dia 1º de janeiro de 1970. No exemplo, esse valor indica as 20 horas, 15 minutos e 0 segundos do dia 13 de março de 2010. Os três dígitos após o ponto indicam os milissegundos.
- Tempo decorrido: tempo total em que o Squid levou desde receber a requisição do cliente, processá-la, e enviar o arquivo ao cliente, medido em milissegundos.
- IP do cliente: endereço de origem, o IP de onde partiu a solicitação de acesso.
- Identificação/código do estado: estado da requisição (existente em cache, inexistente em cache, acesso negado, etc.) e o respectivo código numérico utilizado pelo Squid.

Tabela 2: *Log* de acessos do Squid com período de 1 segundo.

Timestamp	Tempo Decorrido (ms)	IP do Cliente	Identificação/Código do Estado	Tamanho (Bytes)	Método HTTP	URI
1268435700.008	171	10.175...	TCP_MISS/200	391	GET	http://ad...
1268435700.009	20	10.174...	TCP_REFRESH_HIT/304	424	GET	http://www...
1268435700.041	32	10.174...	TCP_MISS/200	1619	GET	http://br...
1268435700.054	518	10.175...	TCP_MISS/200	8492	GET	http://www...
1268435700.057	33	10.175...	TCP_MISS/200	1714	GET	http://stf...
1268435700.072	431	10.175...	TCP_MISS/200	1629	POST	http://by126w...
1268435700.126	438	10.174...	TCP_MISS/200	1821	GET	http://s20...
1268435700.132	162	10.175...	TCP_MISS/200	516	GET	http://suggest...
1268435700.140	448	10.174...	TCP_MISS/200	1126	GET	http://dg...
1268435700.150	27	10.174...	TCP_MISS/200	24663	GET	http://l...
1268435700.168	24	10.175...	TCP_MISS/200	650	GET	http://clients1...
1268435700.168	96	10.175...	TCP_MISS/200	517	GET	http://suggest...
1268435700.178	147	10.174...	TCP_MISS/304	283	GET	http://www...
1268435700.179	1	10.174...	TCP_HIT/200	720	GET	http://afe...
1268435700.186	31	10.175...	TCP_MISS/200	8493	GET	http://stf...
1268435700.192	6	10.175...	TCP_DENIED/403	3737	GET	http://h...
1268435700.275	724	10.174...	TCP_MISS/200	15446	GET	http://msdn...
1268435700.328	3	10.174...	TCP_NEGATIVE_HIT/204	337	GET	http://clients1...
1268435700.331	2	10.175...	TCP_IMS_HIT/304	360	GET	http://img...
1268435700.417	359	10.174...	TCP_MISS/200	513	GET	http://row...
1268435700.450	52	10.174...	TCP_MISS/200	530	GET	http://clients1...
1268435700.456	18	10.174...	TCP_MISS/200	497	GET	http://clients1...
1268435700.462	1125	10.174...	TCP_MISS/200	16072	GET	http://br...
1268435700.477	177	10.174...	TCP_MISS/304	283	GET	http://www...
1268435700.514	24	10.174...	TCP_REFRESH_HIT/304	424	GET	http://www...
1268435700.587	49	10.174...	TCP_MISS/200	25665	GET	http://vagalume...
1268435700.629	22	10.175...	TCP_MISS/200	1313	GET	http://www...
1268435700.637	42	10.175...	TCP_MISS/200	494	GET	http://clients1...
1268435700.659	383	10.174...	TCP_MISS/200	513	GET	http://row...
1268435700.697	521	10.175...	TCP_MISS/200	15623	GET	http://www...
1268435700.750	506	10.174...	TCP_MISS/200	1378	GET	http://dg...
1268435700.768	32	10.174...	TCP_MISS/200	498	GET	http://clients1...
1268435700.909	17	10.174...	TCP_MISS/200	499	GET	http://clients1...
1268435700.932	22	10.174...	TCP_MISS/200	501	GET	http://clients1...

- Tamanho: tamanho total do arquivo retornado em bytes.
- Método HTTP: GET se o usuário está apenas visualizando a página/arquivo ou POST se o usuário estiver enviando alguma informação para a página.
- URI: endereço do destino (arquivo a ser acessado).

O *log* da Tabela 2, mostra todos os acessos dos *proxies* durante um período de 1 segundo, em um horário de pico de utilização dos *proxies*. O *log* mostra os acessos de uma das instâncias de um dos servidores, apenas como demonstrativo. O *log* completo foi omitido pois no total seriam necessárias 970 linhas para mostrá-lo, ou seja, nesse 1 segundo, 970 acessos foram realizados nos *proxies*.

No caso da Tabela 2, existem 34 requisições àquela instância do Squid durante um período de 1 segundo, sendo que apenas cinco foram encontradas em *cache* e uma dessas requisições foi encontrada em memória, ou seja, somente 4 dessas requisições estavam no disco, um total de 11,76%. Somando-se a quantidade de dados lidos do disco, obtém-se o valor de apenas 1491 bytes, um valor próximo do desprezível para os padrões atuais.

Para todas as 10 instâncias dos dois servidores, encontrou-se a situação da Tabela 3.

Tabela 3: Quantidade de arquivos obtidos do *cache* no Squid no período de um segundo, com 10 MB de *cache* em memória.

	Total de Acessos	TCP_MEM_HIT	TCP_HIT	Total de Arquivos em Cache Acessados	Percentual de Arquivos em Disco	Bytes Lidos do Disco
Servidor 1	412	6	72	78	18,93%	281557
Servidor 2	558	15	188	203	33,69%	1649795

Com os dados obtidos na Tabela 3, percebe-se uma diferença significativa dos acessos instantâneos entre os dois servidores, como mostra a coluna "Total de Acessos", que foge ao padrão de igualdade da quantidade de acessos entre os dois servidores, existindo 35% mais acessos no servidor 2. Devido a essa diferença, o teste anterior foi repetido para o período de um minuto, que deverá aproximar-se desse padrão de igualdade.

Como esse trabalho tem como principal objetivo a comparação de dados para fins demonstrativos, os dados obtidos durante o comparativo instantâneo de 1 segundo não serão omitidos para servir como base de estudo, comprovando que análises instantâneas não são eficazes e nem completamente confiáveis nesse tipo de estudo, em que os dados são suscetíveis ao comportamento, interesse e necessidade humanos.

Em um período de um minuto, em apenas uma instância do Squid (a mesma que foi analisada separadamente no passado), houve 3039 acessos, sendo 96 desses acessos obtidos da memória e 604 desses acessos obtidos do *cache* em disco. Nesse mesmo período, um total de 3927136 bytes foram lidos do disco, uma média de aproximadamente 64 KB/s, média que, devido à escala do gráfico (gerada dinamicamente em relação ao maior valor, no caso a da gravação em disco, que passa de vários MB/s), na maior parte do tempo nem mesmo aparecerá representada.

Tabela 4: Quantidade de arquivos obtidos do *cache* no Squid no período de um minuto, com 10 MB de *cache* em memória.

	Total de Acessos	TCP_MEM_HIT	TCP_HIT	Total de Arquivos em Cache Acessados	Percentual de Arquivos em Disco	Bytes Lidos do Disco
Servidor 1	27094	367	4310	4677	15,91%	32062510
Servidor 2	32172	861	6742	7603	20,96%	69489820

Em ambos os casos da Tabela 4, apesar de haver uma diferença considerável da quantidade de dados lidos entre os dois servidores, nenhum deles chega a uma taxa de acesso elevada o suficiente para se poder considerar a diferença dos sistemas de arquivos realmente relevante para o Squid, como mostra a relação da divisão do total de bytes lidos do disco por 1024 (para realizar a conversão do valor de bytes para KB) e a divisão desse resultado por 60 segundos que no caso do servidor 1, a taxa média de leitura resulta em 522 KB/s e no servidor 2 resulta em 1131 KB/s ou 1,1 MB/s. Mesmo a taxa de leitura do servidor 2 sendo superior em quase 117%, o total de bytes lidos do disco não é expressivo quando considerado o total do consumo de banda de rede, que passa de 60 Mbps durante quase todo o dia, como se observa nas Figuras 4 e 5.

3.2.3 *Cache* em Memória

O *cache* gravado em memória é milhares de vezes mais rápido que aquele gravado em disco. Isso se deve à tecnologia utilizada na construção da memória RAM, que, além de muitas outras características que a difere dos discos rígidos, é apenas lida/gravada eletricamente, não dependendo de partes mecânicas, como as cabeças dos discos rígidos, que despendem um tempo muito maior do que a gravação exclusivamente eletrônica.

Após os dados apresentados na subseção 3.2.2, aqui pode-se avaliar se há realmente vantagens e necessidades de se ter *cache* também em disco, ao invés de apenas em memória.

A Tabela 5 mostra um caso com 512 MB reservado na memória para *cache* e os mesmos 1536 MB já reservados em disco para a mesma finalidade.

Tabela 5: Quantidade de arquivos obtidos do *cache* no Squid no período de um dia, com 512 MB de *cache* em memória.

	Total de Acessos	TCP_MEM_HIT	TCP_HIT	Total de Arquivos em Cache Acessados	Percentual de Arquivos em Disco
Servidor 1	22.689.110	2.836.507	2.028.461	4.864.968	41,70%
Servidor 2	22.470.663	2.906.931	2.163.031	5.069.962	42,66%

Como mostra a Tabela 5, a quantidade de arquivos em *cache* recuperados em disco, foi de pouco mais de 40%, uma redução significativa em relação aos dados obtidos com 256 MB de *cache* em memória, conforme apresentado na Tabela 1.

Para o caso sendo estudado agora, a relação entre o espaço reservado em disco e memória para o *cache* é de apenas 3 vezes, ou seja, o espaço reservado em disco é de 300% do espaço reservado em memória. Apesar disso, a quantidade total de arquivos obtidos da memória, foi superior em apenas 37% àquele obtido em disco.

O tempo decorrido médio para acesso do *cache* em disco no primeiro servidor, com o sistema de arquivos XFS, foi de 209 ms e para o segundo servidor, com sistema de arquivos ReiserFS, foi de 211 ms. Já para o *cache* em memória, o tempo decorrido médio foi de 10 ms para o primeiro servidor e de 11 ms para o segundo servidor.

4 RESOLUÇÃO DE NOMES

O DNS (*Domain Name System*, Sistema de Nomes de Domínio, em português) é o servidor utilizado para se traduzir o nome de um domínio como, por exemplo, www.utfpr.edu.br para seu endereço de rede, o endereço IP.

O sistema de nomes de domínio foi originalmente criado para se facilitar a compreensão e memorização de endereços. Por exemplo, é muito mais simples lembrar-se do domínio www.utfpr.edu.br do que de seu IP, que seria algo como 200.19.73.66. Fazendo uma analogia com o endereço físico de uma residência, é muito mais fácil memorizar que a UTFPR está fisicamente situada à Av. Sete de Setembro, do que lembrar-se de que seu CEP é 80230-901. É claro que poderia ser mais fácil lembrar-se de alguns endereços IP do que do seu nome de domínio, quando poucos existem ou são necessários. Entretanto, à medida que mais IPs são necessários de se memorizar, esses mesmos endereços acabam se tornando confusos e também, não são intuitivos e nem traduzem a real finalidade do endereço, ao contrário dos nomes de domínios.

4.1 CONFIGURAÇÃO DE *SOFTWARE* PARA OS TESTES REALIZADOS

- Xen Hypervisor versão 3.2;
- Debian GNU/Linux versão 5.0 (codinome Lenny), utilizando o *kernel* 2.6.26 compilado com suporte ao Xen;
- Squid versão 2.7 STABLE3;
- Bind versão 9.5.1.

4.2 RESOLUÇÃO LOCAL VS. RESOLUÇÃO REMOTA

Em geral, a resolução de nomes costuma ser muito rápida, abaixo de 100 ms, dependendo do número de saltos (equipamentos de rede entre o servidor DNS e a máquina cliente, como

firewalls, roteadores, etc.) até chegar ao servidor de DNS.

A vantagem da utilização de um servidor DNS local[1] é que esse servidor pode armazenar um *cache* dos nomes de domínio mais acessados, o que não proporciona uma economia considerável de banda de rede do servidor Squid (que normalmente não chega a 5% do consumo total de banda), porém, tão importante quanto, proporciona uma economia no tempo final de resposta ao usuário.

Quando utilizado um servidor DNS remoto, supondo que se leve 5 ms para resolver cada nome, multiplicando-se esse tempo por, por exemplo, 32172 acessos realizados em um minuto no servidor 2, toma-se, a grosso modo, aproximadamente, 160 segundos, ou quase 3 minutos, de espera pela resolução de nomes. Obviamente, esse valor, na prática, não tem todo esse impacto, ou seja, isso não quer dizer que em um minuto de acessos realizados pelos usuários, haverá 3 minutos de espera e as consultas ficarão enfileiradas até que esses nomes sejam resolvidos. Entretanto, esse valor serve para ilustrar a relevância do servidor DNS para um servidor *proxy*.

As consultas de nomes de domínio em servidores DNS costumam ser muito rápidas. Entretanto, uma consulta de apenas 5 ms, como no exemplo anterior, só é conseguida na prática quando o servidor DNS está a poucos saltos do cliente e também apenas quando esse servidor já possui o nome do domínio em *cache*, ou seja, quando o servidor DNS não precisa consultar um outro servidor que conheça esse nome de domínio para isso.

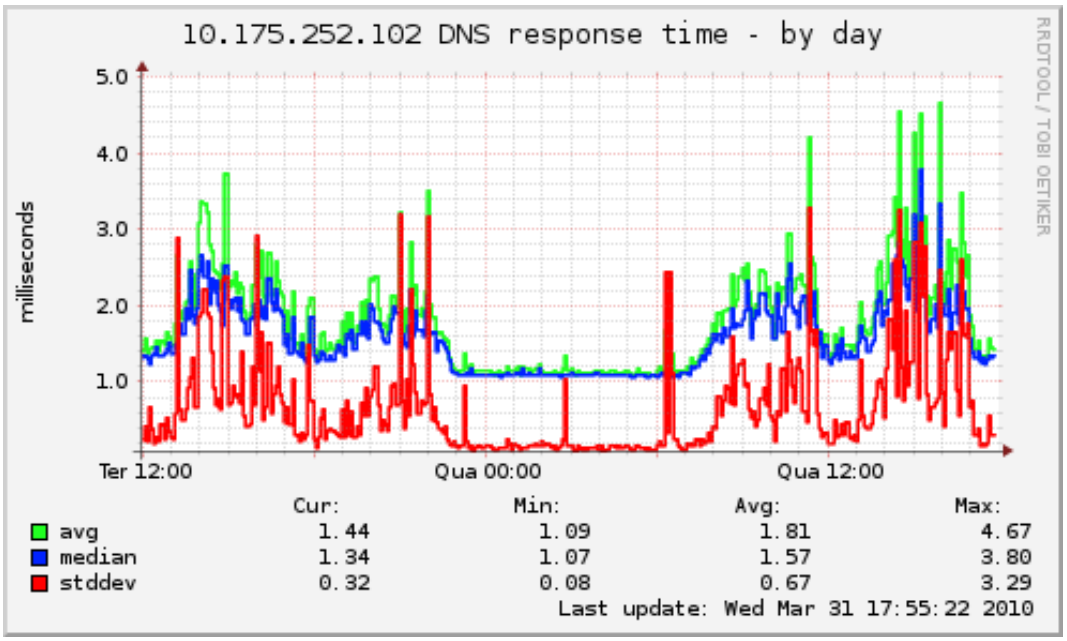
Um fator que em muito influencia o tempo de resolução dos nomes é a distância geográfica entre o cliente e o servidor DNS, ou seja, em tese, evidentemente que para um servidor bem configurado e fisicamente dimensionado de acordo com a demanda existente nele, é muito mais rápido resolver um nome em um servidor que esteja há poucos quilômetros de distância, como um servidor que esteja fisicamente localizado na mesma cidade do cliente, do que utilizar um servidor que esteja em outro continente, localizado há milhares de quilômetros de distância. Obviamente, não há sequer necessidade de explicar os motivos disso, já que são aparentes. Entretanto, neste capítulo, utilizar-se-á também alguns servidores DNS de outros continentes para levantar dados ainda mais precisos acerca do assunto.

Nos casos de análise de DNS localizados a longas distâncias, serão utilizados um servidor DNS no Estado da Califórnia nos Estados Unidos (10 mil quilômetros de distância), outro na Alemanha (10 mil quilômetros de distância) e ainda outro na Índia (14 mil quilômetros de distância), para demonstrar a relação entre o tempo de resolução dos nomes em relação à distância geográfica entre cliente e servidor.

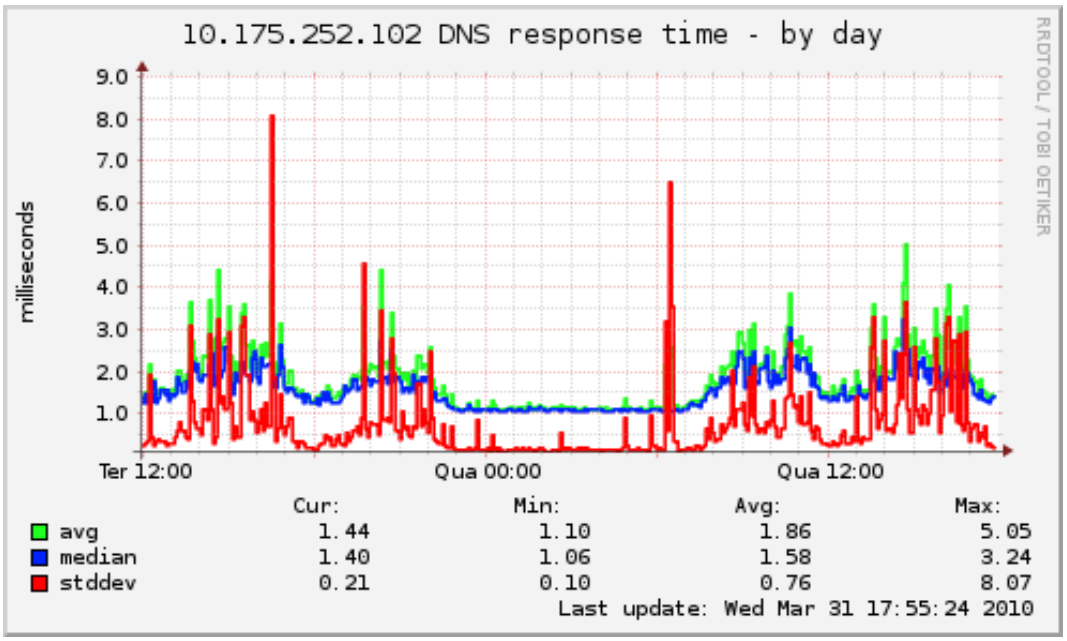
[1] DNS local nesse contexto é um servidor DNS sendo executado na mesma máquina do servidor Squid.

4.2.1 Testes e Comparativos de Desempenho de DNS de Curta Distância

O primeiro caso analisado, é de um DNS localizado há apenas um salto do servidor proxy, ou seja, ele está numa sub-rede na qual é necessário aos pacotes passarem por apenas um equipamento de roteamento até chegar ao servidor DNS. Esse caso está representado na Figura 8.



(a) Proxy 1



(b) Proxy 2

Figura 8: Tempo de resposta de DNS, um salto de distância.

Como se pode observar na Figura 8, o tempo médio de resposta (representado pela linha azul, nomeada “median”, e o valor “Avg”, abreviação de *average*, ou médio, em português),

em ambos os casos (dos *proxies* 1 e 2), é baixo, pouco maior que 1,5 ms. Em um caso real, um tempo médio de resposta menor que 2 ms é praticamente desprezível. Para uma pessoa que esteja acessando um site, esse tempo nem mesmo será percebido.

Como neste trabalho a situação procurada é a mais perfeita possível, apesar de o valor obtido na Figura 8 ser muito baixo, mais alguns casos de tempo de resposta de DNS serão analisados, levando-se em consideração o número de saltos necessários entre o cliente e o servidor DNS ou sua distância geográfica (para casos de DNS em outros continentes).

A distância geográfica não é usual para ser usada como referência, entretanto, devido às características das redes atuais, como bloqueios por parte de seus administradores, torna-se impossível traçar a rota entre origem e destino e estabelecer o número de saltos, sendo assim, a distância geográfica será utilizada como referência meramente ilustrativa neste trabalho.

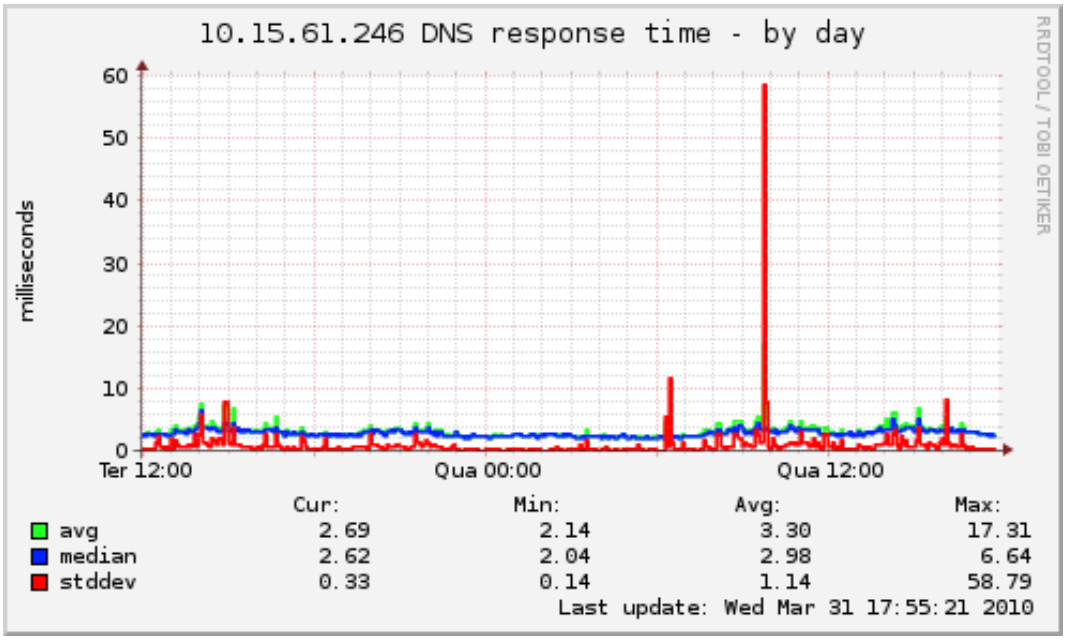
O próximo caso analisado, é de um DNS localizado a dois saltos do servidor proxy. Obviamente, sem nem antes fazer a análise dos dados obtidos, já é possível presumir que ele seja um pouco mais lento que aquele mostrado pela Figura 8, em que apenas um salto era necessário entre cliente e servidor. O caso analisado agora, está representado pela Figura 9.

Como já era previsível, a Figura 9 comprovou que a tendência é de que o tempo de resposta aumente, à medida que mais saltos existem entre cliente e servidor DNS. No caso observado, o tempo de resposta cresceu mais de 1 ms em relação àquele observado na Figura 8, ou seja, esse tempo aumentou em mais de 50%.

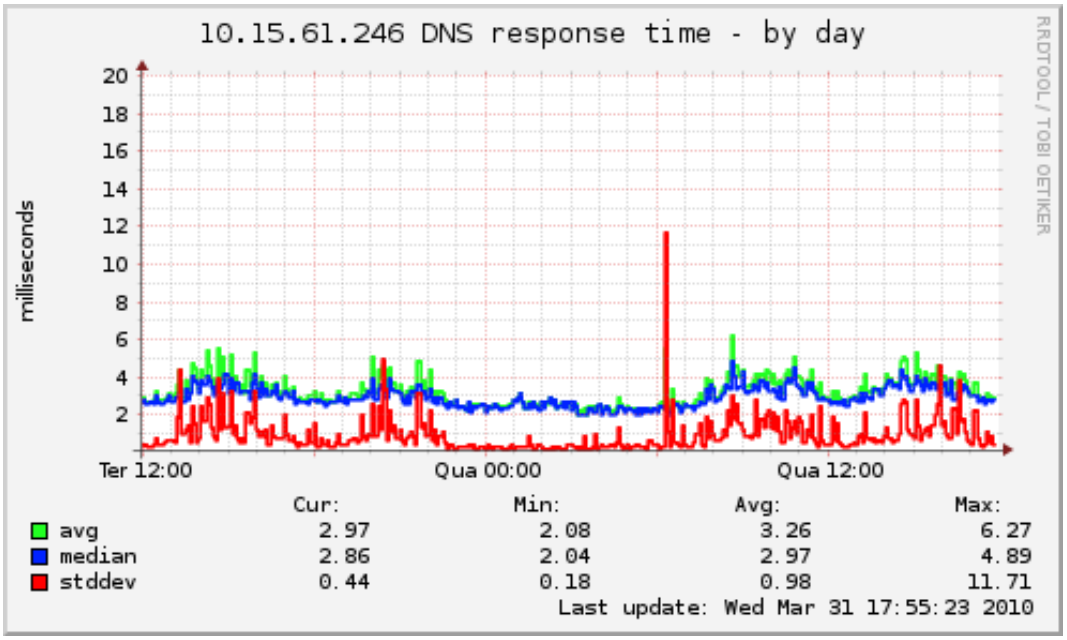
Nos dois casos já analisados, os servidores DNS estavam localizados fisicamente em uma mesma rede privativa, portanto, com poucos metros de distância geográfica. Entretanto, devido ao número de equipamentos pelos quais as requisições necessitam trafegar, esse tempo de resposta aumentou em mais de 50% (na realidade esse tempo chegou quase a dobrar). Essa diferença demonstra claramente que há vantagens reais ao se utilizar um DNS cada vez mais próximo de seu cliente, tanto fisicamente, quanto logicamente (quando considerada a quantidade de equipamentos entre os mesmos).

4.2.2 Testes e Comparativos de Desempenho de DNS de Longa Distância

Nesta subseção, testes com DNS localizados a longas distâncias serão realizados, com objetivo maior de demonstrar o real problema em se utilizar DNS externos, sobretudo quando o DNS está realmente muito distante. É muito comum observar-se clientes utilizando os chamados *Root* DNS (ou DNS Raízes, em português), que são os DNS dos quais todos os outros são dependentes, ou seja, eles estão no topo da cadeia de DNS.



(a) Proxy 1



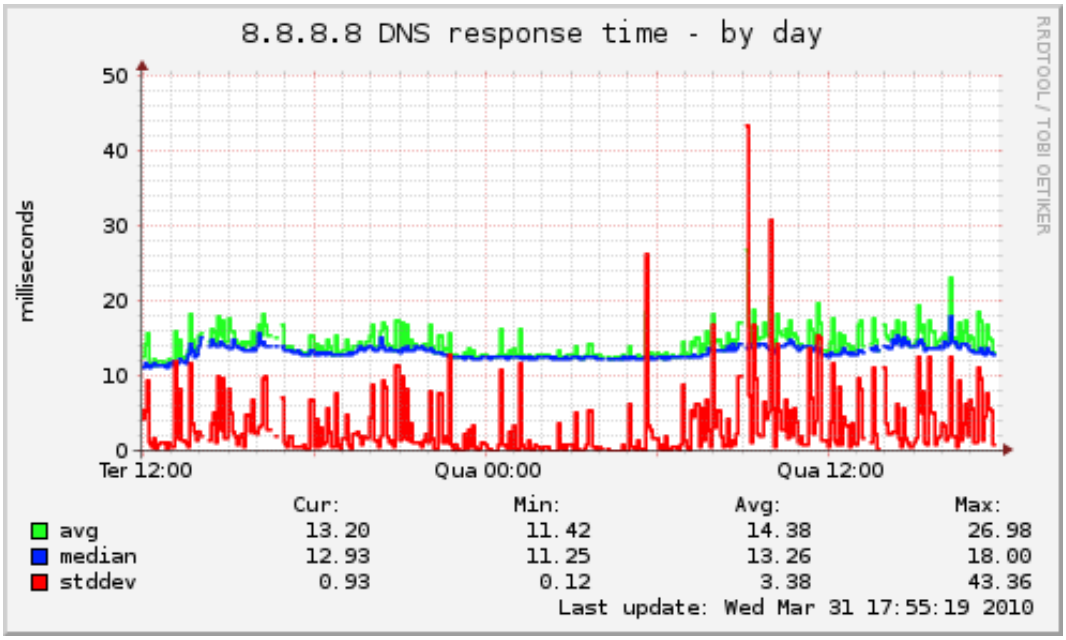
(b) Proxy 2

Figura 9: Tempo de resposta de DNS, dois saltos de distância.

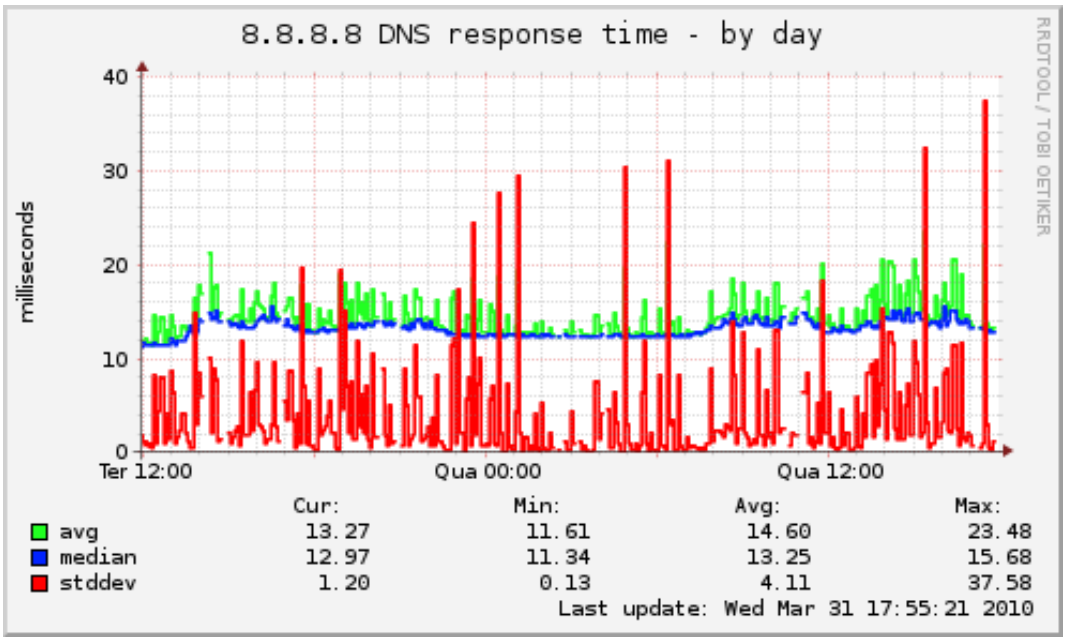
Muitas pessoas desinformadas acreditam que, devido à importância desses *Root* DNS, é mais vantajoso conectar-se diretamente a eles. Entretanto, isso nem sempre é verdade, visto que eles estão espalhados por alguns países do mundo, e se utilizado um DNS que esteja a milhares de quilômetros, seu tempo de resposta será muito elevado para o cliente, sobretudo para os padrões de velocidade hoje atingidos pela Internet, com bandas largas residenciais de alta velocidade.

Nesta subseção, a distância considerada entre servidor e cliente será meramente geográfica e medida em quilômetros e não mais em saltos, visto que não se possui a gerência de toda a rede da Internet e não é possível determinar o número de saltos entre cliente e servidor DNS.

O primeiro DNS analisado está localizado no estado da Califórnia, nos Estados Unidos, e os gráficos do tempo de resposta deste DNS estão representados na Figura 10.



(a) Proxy 1



(b) Proxy 2

Figura 10: Tempo de resposta de DNS, na Califórnia, Estados Unidos, 10 mil quilômetros de distância.

Como pode-se observar, o tempo de resposta desse DNS é altíssimo, se comparado aos

DNS localizados em uma rede local. Naqueles casos, o tempo de resposta era menor que 3 ms, e nesse servidor DNS, já se aumentou para mais de 13 ms, ou seja, um tempo mais de quatro vezes maior, o que, sem dúvida alguma, justifica a existência de um serviço importante como esse, senão na mesma máquina que um servidor *proxy*, ao menos um servidor DNS muito próximo, se possível na mesma sub-rede desse *proxy*.

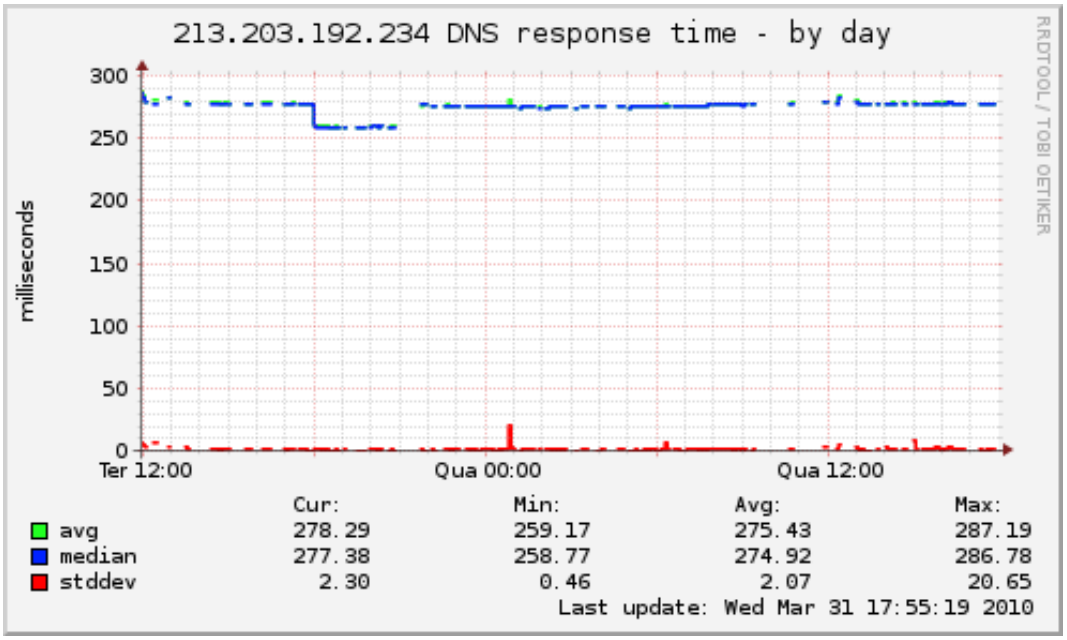
O próximo caso é o de um DNS localizado na Alemanha, a uma distância muito similar ao DNS localizado na Califórnia, nos Estados Unidos, e seus dados estão contidos nos gráficos da Figura 11.

Nesse caso, analisado na Figura 11, já é possível se descartar a possibilidade e vantagem na utilização de um DNS intercontinental, localizado a essa distância. Seu tempo médio de resposta foi de aproximadamente 275 ms, valor esse que já produz um atraso significativo e altamente perceptível ao usuário quando ele tenta acessar um *website* ou fazer o *download* de um arquivo.

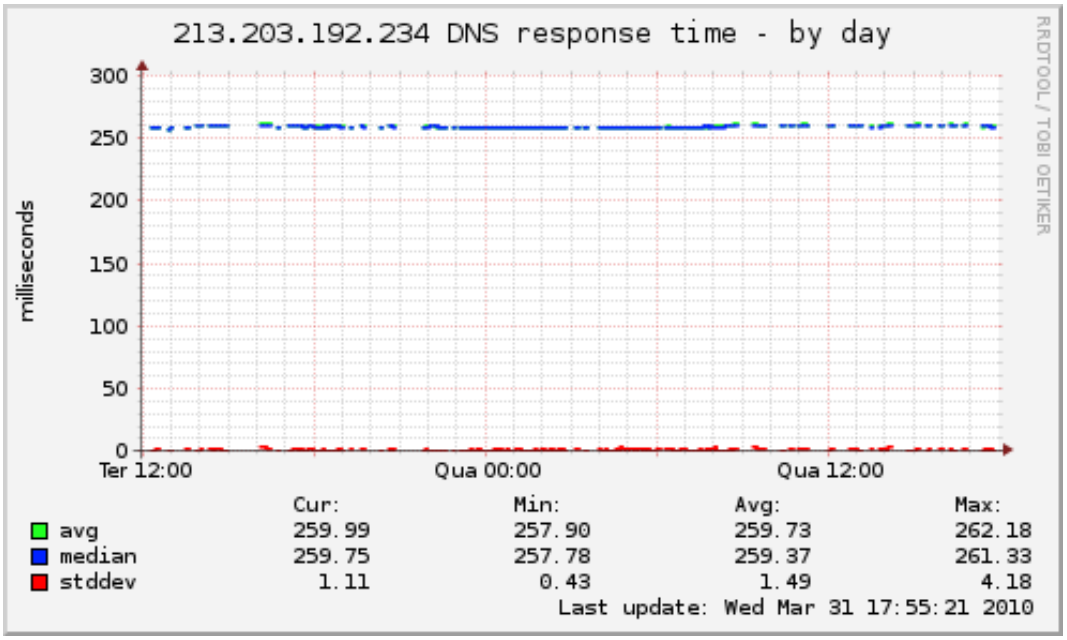
Percebe-se que, apesar de a distância do DNS localizado nos Estados Unidos e esse localizado na Alemanha serem bastante similares, há uma enorme diferença entre os tempos de resposta (maior que 20 vezes). Para esse caso especificamente não se pode chegar a uma conclusão exata do motivo de tamanha diferença. Entretanto, pode-se chegar a três prováveis conclusões, sendo elas as seguintes:

- A quantidade de saltos entre cliente e servidor é muito maior para o caso do DNS na Alemanha do que para aquele localizado nos Estados Unidos;
- Devido ao número total de saltos, a distância física no trajeto entre servidor e cliente é muito superior à distância física direta entre os dois pontos geográficos, isto é, não podemos considerar que a distância que o pacote de rede percorrerá é igual à distância de ponto a ponto, podendo ser muito maior que essa. Fazendo uma analogia, o tráfego do pacote pode ser comparado a uma escala de vôos, em que o avião não faça uma linha reta entre o ponto de origem e o destino final;
- O meio de transmissão utilizado para o segundo caso pode ser um satélite de órbita geo-estacionária.

Como não se possui gerência e nem mesmo informações precisas sobre todas as redes que conectam o cliente ao servidor, é impossível determinar qual a real causa dessa grande diferença.



(a) Proxy 1

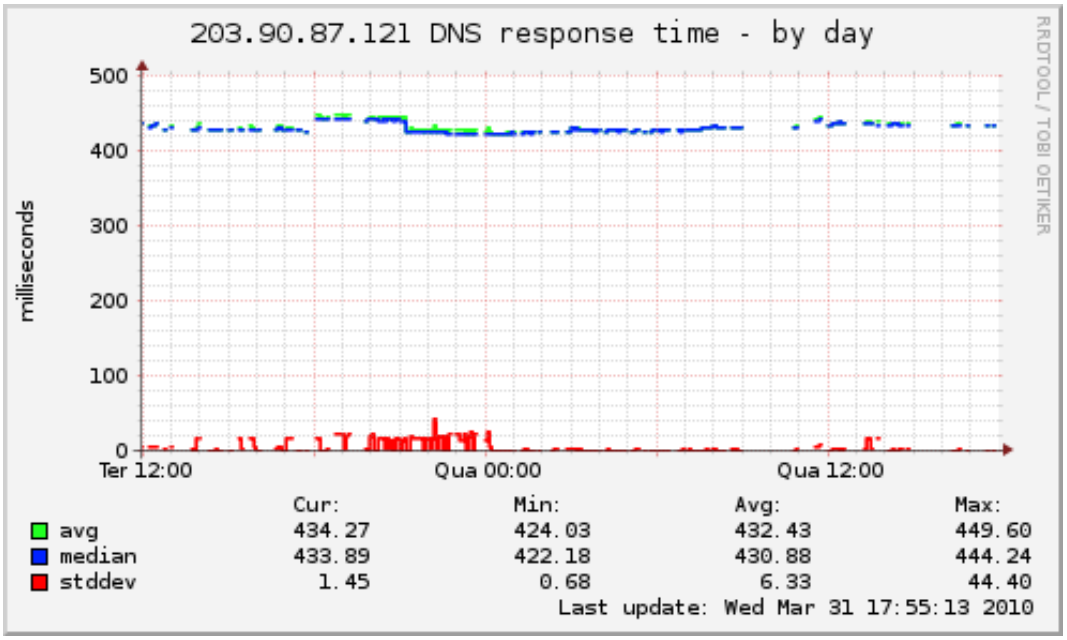


(b) Proxy 2

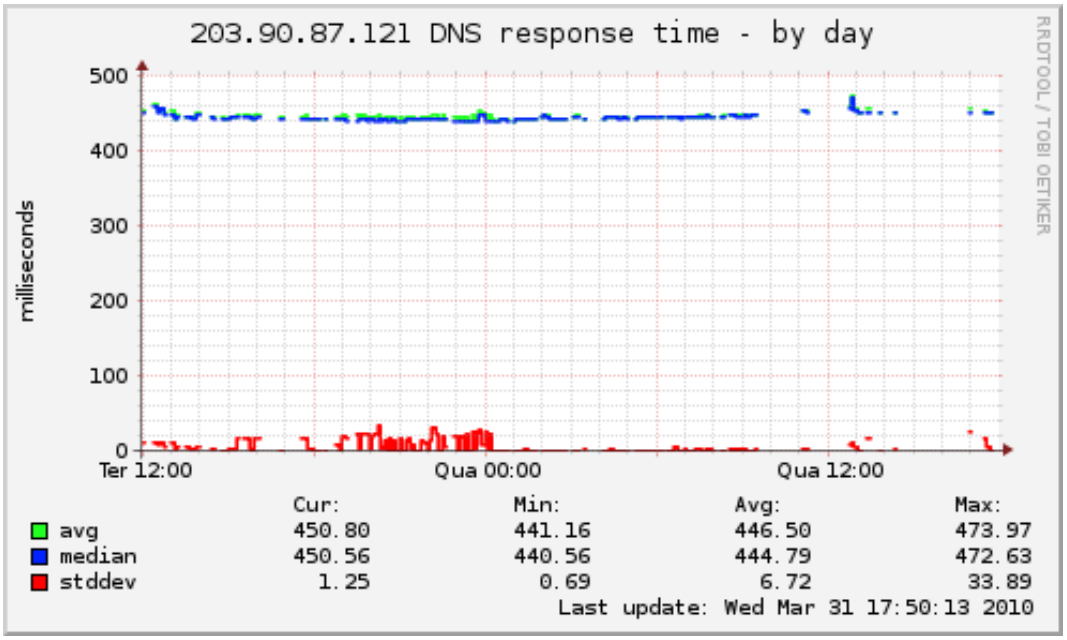
Figura 11: Tempo de resposta de DNS, na Alemanha, 10 mil quilômetros de distância.

O último caso de longa distância analisado trata de um servidor DNS localizado fisicamente na Índia, a 14 mil quilômetros de distância e os gráficos com os dados obtidos estão presentes na Figura 12.

Observando-se os gráficos da Figura 12, percebe-se que o tempo médio de resposta é de 430 ms em um dos servidores *proxy* e de 444 ms no outro, ambos maiores que todos os outros valores médios já analisados.



(a) Proxy 1



(b) Proxy 2

Figura 12: Tempo de resposta de DNS, na Índia, 14 mil quilômetros de distância.

Através do estudo dos dados obtidos com a análise dos três servidores DNS examinados nesta subseção, fica claro que um DNS localizado a uma longa distância de seu cliente não traz vantagem alguma, muito pelo contrário, seu tempo de resposta tende a aumentar quanto maior a distância.

No último e pior caso, o DNS localizado na Índia, com tempo de resposta médio de 437 ms, fazendo-se um comparativo com o pior caso de DNS na rede local, analisado na Figura 9, que

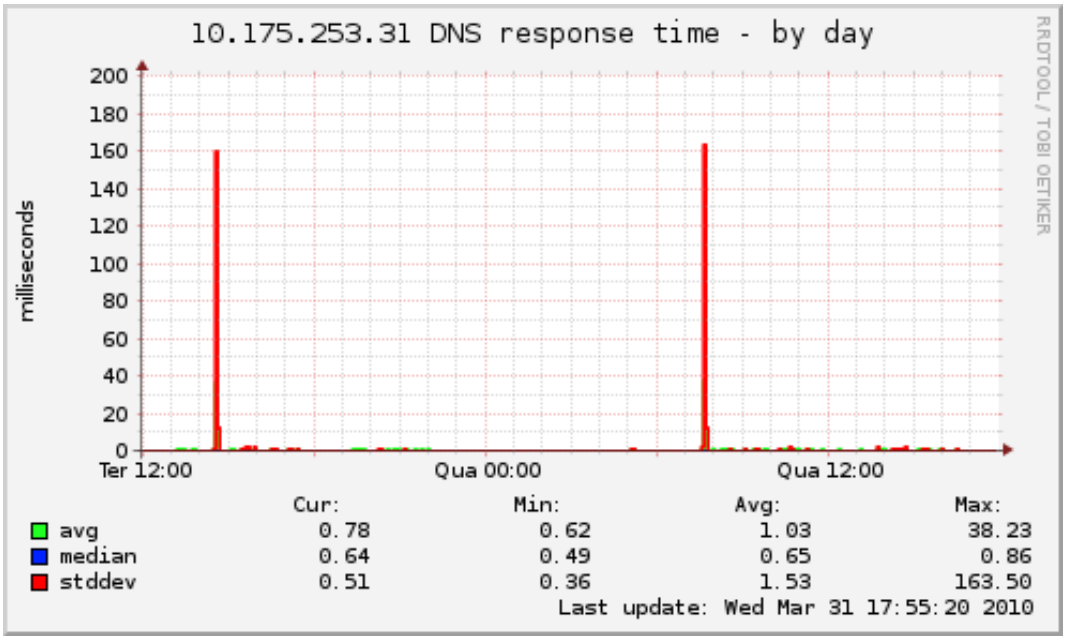
possuía tempo de resposta médio inferior a 3 ms, a relação é de aproximadamente 150 vezes, relação que pode trazer muitas reclamações de usuários ao administrador dos *proxies* ou da rede como um todo.

4.2.3 Testes e Comparativos de Desempenho de um DNS Local

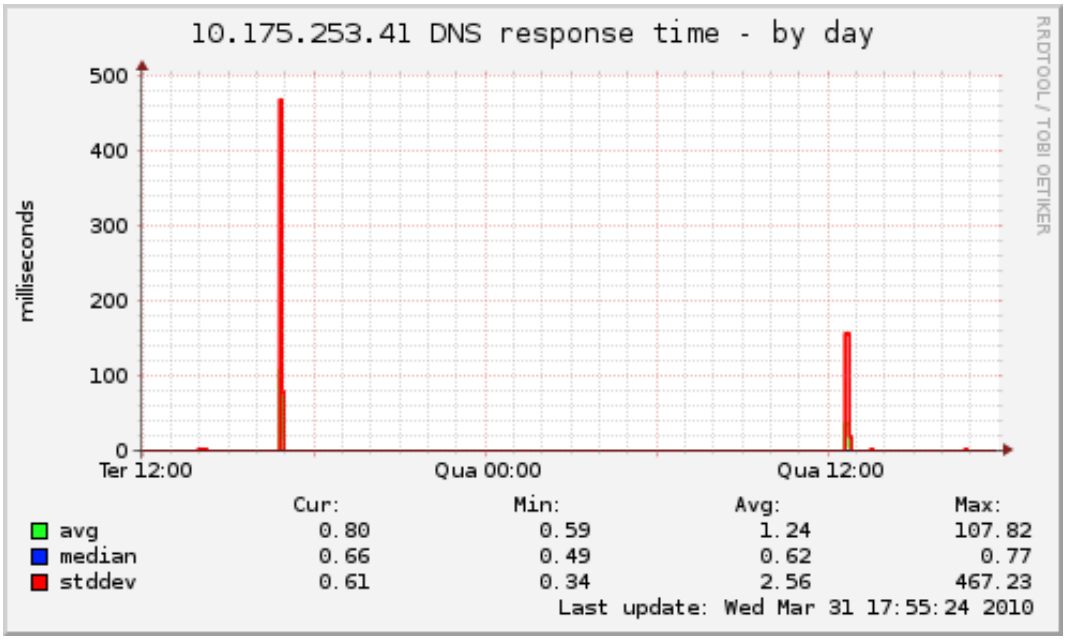
O último estudo de caso de DNS aqui realizado, será feito em virtude de um DNS local, ou seja, o serviço estará em execução na mesma máquina em que o serviço de *proxy*. A Figura 13 possui os gráficos com os dados obtidos no servidor com DNS local.

A Figura 13 mostra que o tempo de resposta médio é inferior a 1 ms, tempo esse que para qualquer aplicação atual, é imperceptível para o usuário.

Durante os outros comparativos, em nenhum caso, obteve-se um tempo médio inferior ou igual a esse, mostrando que um DNS local é realmente eficiente para qualquer serviço que constantemente necessite de resolução de nomes, como é o caso de um servidor *proxy*.



(a) Proxy 1



(b) Proxy 2

Figura 13: Tempo de resposta de DNS local.

5 SQUIDGUARD

O Squid (ou um serviço de *webproxy*, de uma forma geral) possui uma função secundária que é filtrar os acessos dos usuários a determinados tipos de *sites*. Determinados tipos de conteúdos são normalmente impróprios para o ambiente no qual se encontra um *proxy*. Num ambiente empresarial, seus gestores podem achar conveniente não permitir que seus funcionários acessem conteúdos adultos ou mensageiros instantâneos, ou num ambiente educacional, como é o caso do Paraná Digital, também é essencial que não se permita o acesso a conteúdos adultos ou também de intolerância, por exemplo.

Os filtros, entretanto, têm um grande inconveniente: sempre que há alguma tentativa de acesso a alguma página ou arquivo através do *proxy*, uma lista com tudo que se deseja filtrar tem que ser consultada, analisando-se filtro a filtro se o acesso desejado condiz com o filtro para que o Squid possa permitir ou não o acesso a este destino desejado.

As listas que contêm esses filtros, tendem a possuir milhares ou até milhões de entradas, pois, a maioria dos endereços de destino que possuem conteúdos impróprios ao acesso não podem ser filtrados apenas por uma palavra ou expressão porque seus endereços usualmente não possuem nenhuma característica de fácil compreensão. Devido a este fato, os destinos são normalmente bloqueados um a um.

O Squid possui um sistema de ACL (*Access Control Lists*, ou Listas de Controle de Acesso, em português) que permite que ele próprio realize esse filtro. Entretanto, o SquidGuard promete ser muito mais eficiente e rápido. Uma das características do SquidGuard que permite maior eficiência é a utilização de um banco de dados, diferentemente do Squid, que armazena nomes de domínios, IPs e expressões em forma de texto.

5.1 CONFIGURAÇÃO DE *SOFTWARE* PARA OS TESTES REALIZADOS

- Xen Hypervisor versão 3.2;
- Debian GNU/Linux versão 5.0 (codinome Lenny), utilizando o *kernel* 2.6.26 compilado

com suporte ao Xen;

- Squid versão 2.7 STABLE3;
- SquidGuard versão 1.4.

5.2 AMBIENTE UTILIZADO

Para se avaliar o desempenho entre os dois servidores, ambos utilizarão uma lista com mais de 950 mil regras de nomes domínios, IPs e expressões bloqueadas ou liberadas, um deles com SquidGuard e outro com ACL diretamente no serviço do Squid.

Uma lista desse porte, ocupa quase 20 MB em disco, e cada uma das instâncias do Squid (ou do SquidGuard) a carrega em memória. Apesar disso, cada vez que um usuário requisita acesso a algum *site*, o Squid tem que verificar domínio por domínio se o acesso ao *site* em questão é permitido ou não, passando assim por até quase um milhão de regras de acesso a cada uma das requisições, cada servidor necessitando realizar essa tarefa mais de 20 milhões de vezes diariamente, conforme os dados já analisados no Capítulo 3.

Para utilizar o SquidGuard, é necessário fazer com que o Squid redirecione suas requisições para o SquidGuard, ou seja, ele será tratado como um processo externo (um aplicativo à parte, e não embutido no Squid).

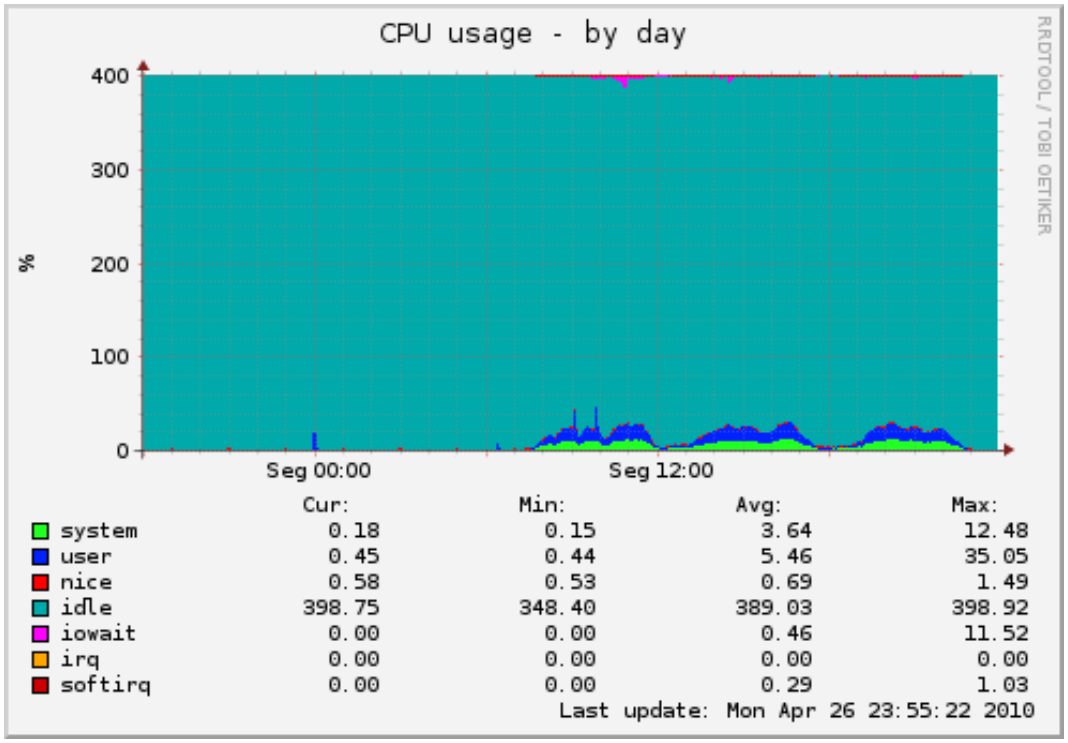
5.3 TESTES E COMPARATIVOS

Durante um dia, foram coletados os dados que serão observados nesta seção, sempre contando com as mais de 950 mil regras já citadas na Seção 5.2, em complementação às menos de 5 mil regras que existiam anteriormente.

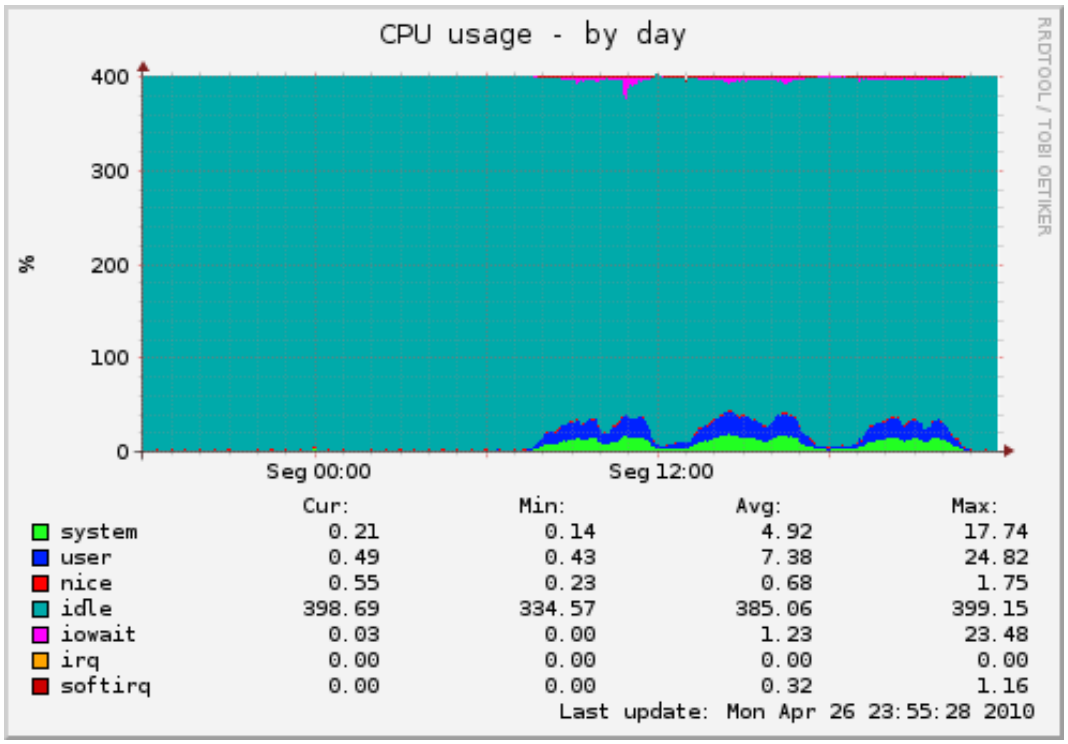
O primeiro recurso avaliado é a utilização de CPU, que no caso do SquidGuard, é o que tende a apresentar maior diferença, visto que é o processador que executa a validação dos *sites* que são acessados pelo usuário, se estes são permitidos dentro da rede ou não.

Como observa-se na Figura 14, o consumo médio de processamento do servidor que utiliza o SquidGuard é maior do que aquele que não utiliza o SquidGuard. Pela avaliação da característica *idle*, percebe-se que a média de utilização foi aproximadamente 4% superior no servidor com SquidGuard, valor que representa apenas 1% da capacidade total do servidor.

Por si só, a análise do consumo da capacidade de processamento mostra que a utilização do SquidGuard não é benéfica para um servidor *proxy*, o que torna necessário que haja ganho de



(a) Análise do servidor sem SquidGuard



(b) Análise do servidor com SquidGuard

Figura 14: Percentual de consumo de processamento do servidores *proxy*, comparativo da utilização do SquidGuard.

desempenho em outros aspectos do serviço, aspectos que serão abordados a seguir.

Além dos dados já observados sobre a utilização de CPU, outro ponto importante a ser

avaliado para os efeitos de desempenho sendo estudados, é o tempo decorrido médio para o acesso às páginas.

O que se espera com a utilização do SquidGuard, não é a redução de processamento do servidor, mas sim a melhoria no tempo de resposta, que é o fator relevante para o usuário que está requisitando acesso a um determinado objeto em um *site*, pois o aumento observado no consumo da capacidade de processamento pode ser ainda justificado porque há um novo aplicativo sendo executado em paralelo ao Squid, que gera um consumo de recursos também para a comunicação, ou seja, o envio das requisições do Squid para o SquidGuard e a posterior resposta fornecida ao Squid.

Tabela 6: Quantidade de arquivos obtidos do *cache* no Squid no período de 24 horas, com 512 MB de *cache* em memória, comparativo com o SquidGuard.

	Total de Acessos	Tempo Médio TCP_MEM_HIT	Tempo Médio TCP_HIT	Tempo Médio TCP_DENIED	Tempo Médio TCP_MISS
Servidor 1	24.829.548	26 ms	292 ms	46 ms	125 ms
Servidor 2	26.459.149	6 ms	198 ms	7 ms	148 ms

Como pode-se perceber pela Tabela 6, o servidor 2 (que está configurado para redirecionar as requisições para o SquidGuard) possui tempo médio de acesso para os objetos obtidos da memória, do disco e para os acessos negados consideravelmente menor que o servidor 1, que utiliza os mesmos filtros, porém, diretamente configurados no Squid. Observe ainda que a Tabela 6 introduz dois outros estados que são o TCP_DENIED e o TCP_MISS, sendo que o primeiro identifica os acessos negados e o segundo identifica os acessos que não possuíam os objetos requisitados no *cache* da máquina.

Apesar de a média de consumo de processamento ter sido superior para o servidor que utiliza o SquidGuard, o tempo de resposta foi consideravelmente menor nos casos em que o *cache* era consultado e, apesar de o tempo de resposta do servidor que utiliza apenas o Squid para controle de acesso já ser muito baixo para a percepção humana, se esse tempo de resposta for associado a outras características da rede, como casos de usuários com conexão via satélite, que já possuem um atraso da ordem de 600 ms, pode-se, com pequenas mudanças como essa em paralelo com outros serviços de rede como os DNS, proporcionar um ganho de desempenho bastante perceptível, mesmo para os menos exigentes usuários dos serviços.

6 SISTEMAS OPERACIONAIS

Neste capítulo serão analisadas as diferenças de desempenho entre dois sistemas operacionais para demonstrar como um mesmo *software* (no caso, o Squid) pode comportar-se apenas pela diferença do sistema operacional, que é a base que o executa. Apesar de o *software* ser o mesmo em sua concepção, em sua forma de funcionamento e em seu resultado final, há sempre várias linhas de seu código que sofrem modificações para que este torne-se compatível com mais de um sistema operacional.

Os sistemas diferem em desempenho e forma de funcionamento de diversas maneiras, especialmente pela forma como seu código é escrito (apesar de ser possível escrever um código de diversas maneiras para executar exatamente a mesma coisa, há sempre uma forma mais rápida e mais elegante de se fazer isso) e pela sua finalidade (os desenvolvedores dos sistemas operacionais sempre têm um objetivo primário, como maior facilidade para uso, melhor desempenho para aplicações em rede, maior segurança ou confiabilidade, dentre outros). Todas essas diferenças influenciam diretamente o funcionamento dos aplicativos.

Apesar de o sistema operacional influenciar diretamente no desempenho, normalmente o próprio *software* que possui essa significativa diferença foi inicialmente desenvolvido para funcionar em um sistema e em seguida foi adaptado para funcionar em outros, o que também gera grande influência em seu resultado final. Em geral, tem um melhor desempenho naquele sistema para o qual foi inicialmente desenvolvido.

Antes de continuar com os testes, é importante introduzir uma ferramenta muito comum em sistemas Unix e em seu clone Linux, o sysctl. Essa ferramenta permite que diversas configurações do sistema sejam alteradas, incluindo configurações de rede, de sistemas de arquivos e do *hardware* da máquina, podendo algumas das configurações serem alteradas até mesmo em tempo real. O sysctl será bastante utilizado no decorrer deste capítulo e o próximo parágrafo traz sua definição oficial encontrada em seu manual.

O sysctl é um utilitário para obter ou alterar os estados do *kernel* e permitir que processos com os privilégios apropriados alterem os estados do *kernel*. Os estados a serem obtidos ou

alterados são descritos usando um estilo de nomes “MIB” (Management Information Base, em português, Base de Gerenciamento de Informações), descrito como um grupo de componentes utilizando pontos como caractere separador (THE FREEBSD DOCUMENTATION PROJECT, 2007).

6.1 VIRTUALIZAÇÃO COM XEN

6.1.1 Configuração de *Software* para os Testes Realizados

A seguir estão todas as possíveis configurações de *software* utilizadas ao longo deste capítulo. As configurações específicas estão descritas no decorrer das seções.

Na máquina Linux:

- Xen Hypervisor versão 3.2;
- Debian GNU/Linux versão 5.0 (codinome Lenny), utilizando o *kernel* 2.6.26 compilado com suporte ao Xen;
- Squid 2.7 STABLE3.

Na máquina FreeBSD:

- Xen Hypervisor, versão 4.0;
- Xen Hypervisor, versão 3.2;
- KVM, versão 0.12.4;
- FreeBSD versão 8.0 e Squid 2.7 STABLE7;
- FreeBSD versão 7.3 e Squid 2.7 STABLE7.

Na máquina NetBSD:

- Xen Hypervisor, versão 3.2;
- KVM, versão 0.12.4;
- NetBSD versão 5.0.2 e Squid 2.7 STABLE7.

6.1.2 Xen

Para avaliar as diferenças de desempenho entre dois sistemas operacionais neste trabalho, serão comparados o Linux ao FreeBSD ou o Linux ao NetBSD, procurando-se sempre manter os dois sempre o mais similares quanto seja possível, tanto no quesito de versões de *software*, quanto no quesito de configurações.

Após alguns testes, foi constatado que não seria possível utilizar o Xen versão 3.2 para este trabalho, pois ele possui algumas limitações. O maior empecilho para o desenvolvimento do trabalho foi em relação ao QEMU[1], visto que no Debian 5.0 a versão disponível do QEMU é a 0.9.1, a qual não possui uma série de funcionalidades, como a emulação de interfaces de rede Gigabit, emulação essa que só tornou-se disponível a partir da versão 0.10.0, através da emulação da interface "Intel® PRO/1000 PCI-Express Gigabit Ethernet".

Para as necessidades da análise aqui tratada foi então necessária a versão mais atual do Xen que é a 4.0. Essa versão já possui os módulos e bibliotecas do QEMU que possuem compatibilidade com a placa de rede Intel® PRO/1000, uma necessidade básica para os *proxies* do Paraná Digital, que em horários de pico possuem utilização de banda maior que 100 Mbps.

Como o suporte a interfaces gigabit fez-se necessário para a conclusão da análise e por tratarem-se os *proxies* de ambientes já em produção, é necessário usar sempre versões estáveis dos sistemas operacionais, aplicativos e todo o *software* que se fizer necessário para permitir o funcionamento deles. Sendo assim, o Debian 5.0 (última versão estável do sistema operacional) era necessário para a utilização como hospedeiro do Xen na máquina física.

Quando o código-fonte do Xen 4.0 foi obtido e em seguida compilado, este utilizava por padrão o *kernel* do Linux versão 2.6.31.13. Entretanto, após todo o processo ter sido concluído com sucesso, o sistema apresentou o erro abaixo enquanto a *blade* era reiniciada com seu novo *kernel*:
(XEN) Panic on CPU 0:
(XEN) IO-APIC + timer doesn't work! Boot with apic=debug and send a report. Then try booting with the 'noapic' option.

Com a análise do problema e pesquisa nas documentações do Xen, constatou-se que a versão 2.6.31.13 do *kernel* não suporta chamadas ao novo IOAPIC com versões do Xen superiores à 3.4.2 (CITRIX SYSTEMS, 2010b), ou seja, a partir da versão 3.4.3 é necessário utilizar-se um *kernel* mais novo para que o sistema possa funcionar como hospedeiro, e a versão atualmente

[1] O QEMU é um emulador de *hardware* do qual o Xen é dependente para algumas aplicações, dentre elas a emulação de *hardware* para o FreeBSD, já que o Xen não possui suporte nativo à emulação de *hardware* para esse sistema operacional.

disponível nos repositórios de desenvolvimento é a 2.6.32.12 (CITRIX SYSTEMS, 2010a).

Então o *kernel* do Linux versão 2.6.32.12, última versão dada como estável pelos desenvolvedores do Xen com suporte a esse sistema de virtualização, foi compilado, mantendo-se todo o suporte ao *hardware* da máquina física em questão (Blade DELL, modelo PowerEdge M600), e em seguida, o Xen 4.0, última versão estável do *software* de virtualização.

Após o Xen 4.0 ter ficado estável na máquina e funcional para as necessidades do FreeBSD, o sistema operacional foi instalado, mantendo-se, dentro do possível, as configurações de sistema e *software* as mais parecidas possíveis com as da máquina com Linux, para evitar que mudanças muito significativas afetem o comparativo feito, o qual deve ater-se exclusivamente ao desempenho do sistema operacional.

6.1.3 FreeBSD Versão 8.0

Num primeiro momento, o FreeBSD versão 8.0 foi utilizado, pois trata-se de sua versão mais atual e em teoria, com suporte a um maior número de dispositivos físicos e com maior estabilidade do sistema.

Durante algumas horas o servidor foi deixado ativo, com as dez instâncias do Squid atendendo às conexões dos clientes. Porém, depois de algum tempo, o servidor "congelou", não permitindo que nada mais fosse feito nele. Nesse momento, o servidor físico ficou completamente parado, consequentemente parando também o servidor virtual que possuía o Squid. Nesse primeiro caso, o "congelamento" de todo o sistema foi tratado apenas como uma falha comum, ou seja, acreditava-se que a falha pudesse ter sido ocasionada por um evento fora do padrão, pois não havia nenhum registro em nenhum *log* da máquina que pudesse indicar a causa do problema.

Em seguida a máquina foi reiniciada, mas o problema voltou a ocorrer alguns minutos depois disso, descartando a hipótese de que um evento aleatório pudesse ter gerado o "congelamento" da máquina física, então monitoramentos do sistema mais aprofundados e soluções foram buscadas para o problema.

A primeira característica percebida foi que, além da máquina virtual ficar com a carga bastante alta desde o momento em que o sistema foi iniciado, a máquina física, hospedeira do FreeBSD, também ficou com a carga bem elevada, chegando a consumir toda a memória RAM disponível, algo incomum para a máquina hospedeira. No caso das máquinas físicas que hospedam apenas máquinas virtuais com Linux, essas ficam praticamente 100% ociosas durante todo o tempo, como é o caso do servidor 1.

O alto consumo dos recursos de *hardware* da máquina hospedeira justificam-se por duas razões básicas. A primeira delas é que as máquinas virtuais que utilizam sistema operacional Linux no caso desse trabalho sempre utilizam o modelo de paravirtualização. Atualmente não é possível utilizar o modelo de paravirtualização para o FreeBSD com máquinas com o padrão x86_64, sendo necessário utilizar-se o modelo de virtualização total para isso. A segunda razão é que a virtualização total necessita de um *software* de terceiros para ser efetuada, que apesar de suportada pelo Xen, não é desenvolvida pela mesma comunidade, sendo o *software* em questão o QEMU.

Em suma, o modelo de paravirtualização faz com que o sistema hóspede acesse o *hardware* da máquina física diretamente, através de um dispositivo virtual criado pelo sistema operacional hóspede. Para que o hóspede crie o dispositivo virtual em questão é necessário que o sistema que se deseja hospedar possua suporte a esse tipo de virtualização, ou seja, o sistema terá que sofrer algum tipo de modificação para estar habilitado a realizar esse tipo de virtualização (VMWARE, 2007).

O modelo de virtualização total (também conhecido como virtualização completa) emula determinados dispositivos e a máquina virtual os reconhecerá como algum dos modelos de dispositivo que ele possui suporte e em alguns casos, os dispositivos emulados podem não ser suficientes para prover toda a necessidade de recursos da máquina virtual (no caso do Xen, através desse modelo faz-se impossível, por exemplo, o uso de placas de rede 10 Gbps, por atualmente não existir suporte a nenhum dispositivo desse tipo). A virtualização total faz a intermediação da máquina virtual com o *hardware* através da máquina hospedeira, ou seja, todo e qualquer recurso de que ela necessite terá obrigatoriamente que ser processado também pela máquina física. Sendo assim, sempre haverá consumo de *hardware* nas duas máquinas, tanto na física quanto na virtual, ocasionando perda no desempenho. A grande vantagem desse modelo é a possibilidade de utilizar sistemas operacionais virtualizados sem nenhuma necessidade de modificação, o que permite que praticamente qualquer sistema operacional seja um hóspede em potencial, desde que ele forneça suporte ao *hardware* emulado e que sua arquitetura seja suportada pela máquina hospedeira (VMWARE, 2007).

O Xen utiliza o QEMU para realizar a intermediação entre *hardware* e máquina virtual no modelo de virtualização completa. Sempre que se utiliza um *software* externo para realizar uma tarefa, a tendência é que ele torne-se mais lento, pois além de todos os problemas já citados anteriormente, ainda haverá a comunicação entre os dois processos executados, o que também consumirá recursos dos dispositivos.

6.1.4 FreeBSD Versão 7.3

Após todos os problemas encontrados com o FreeBSD versão 8.0 sendo virtualizado através do Xen, a alternativa foi a tentativa da utilização do FreeBSD versão 7.3 que é atualmente a versão legada do FreeBSD, ou seja, a versão a que ainda se fornece suporte e atualizações para manter a compatibilidade especialmente com aquelas máquinas que não podem simplesmente ser paradas para efetuar uma atualização desse porte tão frequentemente.

As configurações usadas nos *proxies* na versão 7.3 do FreeBSD foram mantidas idênticas às utilizadas na versão 8.0, inclusive as duas utilizando a arquitetura amd64 e a versão 2.7 STABLE7 do Squid.

Num primeiro momento, a versão 7.3 mostrou-se mais estável por não ter travado em nenhum momento durante todos os testes. Entretanto, com o passar do tempo e uma rápida análise de desempenho da máquina, um limitante foi descoberto, pois a taxa de transferência ao chegar a determinado pico, acabou por estagnar, conforme a Figura 15.

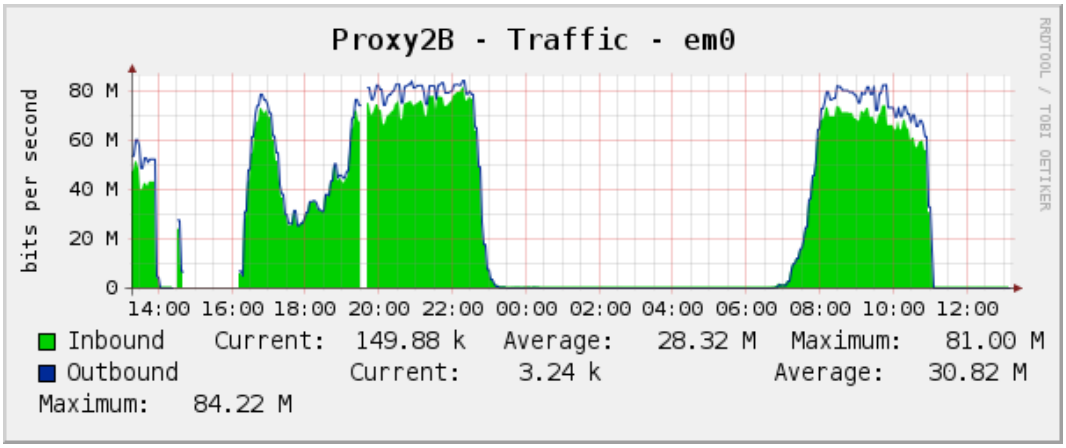


Figura 15: Tráfego de Rede no Freebsd versão 7.3 virtualizado com Xen 4.0.

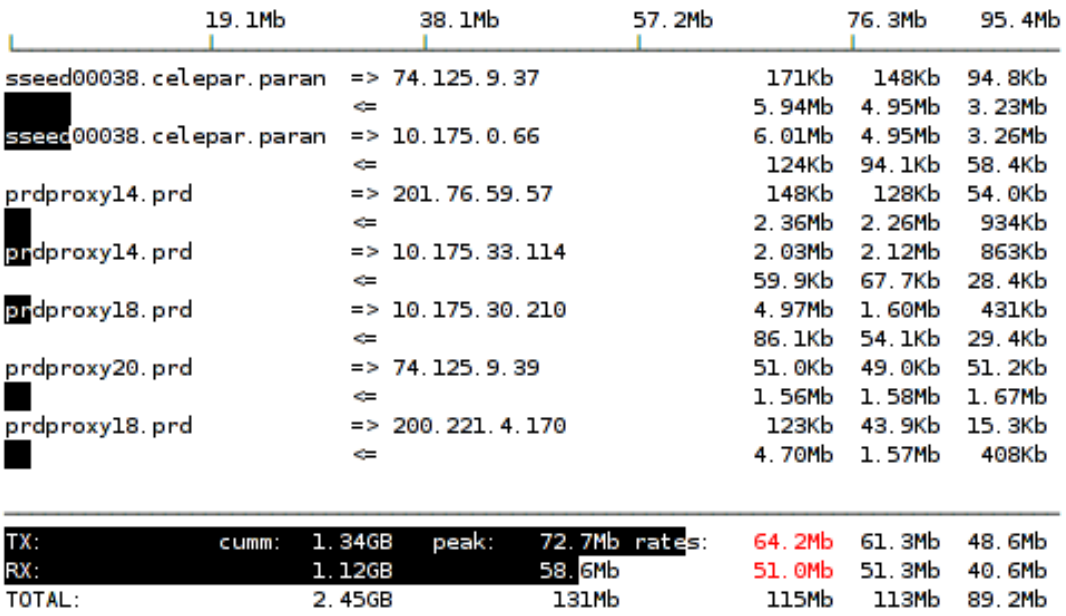
Na Figura 15, percebe-se que por várias horas (como o intervalo observado durante as 20h e 22h) o gráfico fica praticamente estável, próximo à taxa de 80 Mbps, tanto para a banda de entrada quanto de saída.

Após a avaliação desse gráfico, uma investigação minuciosa começou a ser realizada com o objetivo de identificar qual recurso da máquina estava em seu limite.

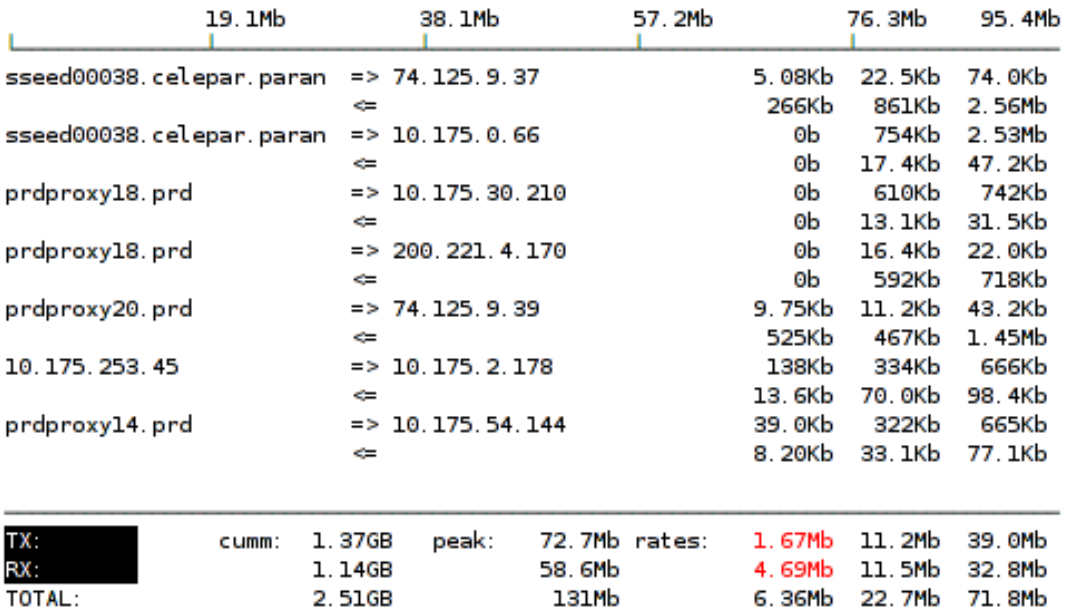
Num primeiro momento, a observação de uma ferramenta de análise do consumo de banda foi utilizada, chamada iftop. O iftop mostra informações como consumo de banda entre origem e destino na interface de rede especificada, total de consumo de banda de entrada e saída das interfaces de rede. Entretanto, diferentemente do Cacti, esta ferramenta permite que essas in-

formações sejam atualizadas para intervalos de tempo mais curtos, e o tempo padrão tanto para o Debian 5.0 quanto para o FreeBSD versão 7.3 é de 1 segundo.

Com a avaliação das informações apresentadas pelo iftop, dois estados diferentes foram observados, conforme a Figura 16.



(a) Primeiro estado



(b) Segundo estado

Figura 16: Consumo de banda no FreeBSD versão 7.3 virtualizado com Xen 4.0.

A Figura 16 mostra dois estados distintos, o primeiro deles pode ser considerado como a situação esperada, indicada pelos valores em vermelho. Esses valores indicam que a taxa de envio é de aproximadamente 64 Mbps, enquanto o consumo de banda de recebimento é de 51

Mbps. No segundo estado, os valores apresentados são muito distintos da situação esperada, e o consumo de banda de saída é de apenas 1,67 Mbps e o consumo de banda de entrada é 4,69 Mbps, valores que estão muito abaixo do real consumo nos *proxies*, muito diferentes daqueles valores da Figura 15.

Os dados obtidos geraram desconfiança sobre as configurações de *software* e *hardware* utilizados na máquina virtual e após minuciosas observações do sistema, constatou-se que realmente havia um limitante que tornava o acesso à máquina lento e suscetível às bruscas variações de consumo de banda pela máquina observadas na Figura 15.

O limitante encontrado nesse ponto foi o acesso ao disco, que através do Xen 4.0 e QEMU 0.10.2 só permitia a emulação de discos IDE no padrão ATA-4 que permite taxas de transferência máximas de 33 MBps, valor muitíssimo inferior às taxas máximas de transferência do disco real, que é de 3 GBps.

A Figura 17 mostra também dois estados. O primeiro deles coincide com o momento em que o consumo de banda mostrado na Figura 16 é de 64 Mbps de saída e 51 Mbps de entrada, já o segundo estado representa o momento em que a banda de saída é de 1,67 Mbps e a de entrada é 4,69 Mbps.

```
dT: 1.016s  w: 1.000s
 L(q)  ops/s    r/s   kBps   ms/r    w/s   kBps   ms/w   %busy Name
    0     66     52    675   13.3     14    864   31.7   42.0| ad0
    0     66     52    675   13.6     14    864   32.3   42.1| ad0s1
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1a
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1c
    0     66     52    675   13.8     14    864   32.4   42.3| ad0s1d
```

(a) Primeiro estado

```
dT: 1.006s  w: 1.000s
 L(q)  ops/s    r/s   kBps   ms/r    w/s   kBps   ms/w   %busy Name
 3313    518      1      2   2653    517   4688   2053  100.3| ad0
 3314    517      0      0    0.0    517   4688   2063  100.2| ad0s1
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1a
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1c
 3314    517      0      0    0.0    517   4688   2071  100.2| ad0s1d
```

(b) Segundo estado

Figura 17: Taxas de leitura e gravação em disco no FreeBSD versão 7.3 virtualizado com Xen 4.0.

Os dois estados constantes na Figura 17 mostram a configuração do sistema para dispositivos ATA com *cache* de escrita desabilitado, ou seja, a escrita em disco não é síncrona, fazendo com que os dados que deverão ser gravados efetivamente no disco fiquem armazenados em *buffer* até que o sistema operacional execute a função que sincronizará os dados. No primeiro estado, o sistema operacional não está executando a sincronização para o disco, já o estado se-

guinte mostra o exato momento em que os dados estão sendo sincronizados, o que faz com que o disco esteja 100% ocupado.

Com a observação do sistema e das Figuras 16 e 17, foi possível constatar que o momento da sincronização dos dados no disco faz com que o sistema sofra uma queda de desempenho muito alta, o que torna inviável a sua utilização com essas propriedades.

Por fim, uma última modificação de configuração do sistema operacional foi realizada, com o objetivo de tornar a escrita em disco síncrona, ou seja, o *cache* de escrita em disco foi habilitado, através da opção "hw.ata.wc" (THE FREEBSD DOCUMENTATION PROJECT, 2010).

Após a habilitação do *cache* de escrita em disco no FreeBSD, a taxa de gravação em disco ficou constante, ou seja, não sofria variações bruscas no decorrer do tempo, como mostra a Figura 18.

```
dT: 1.004s  w: 1.000s
 L(q)  ops/s    r/s   kBps   ms/r    w/s   kBps   ms/w   %busy Name
    8    706      0      0    0.0    706   9932    9.3  100.0| ad0
    9    705      0      0    0.0    705   9916   10.1   99.9| ad0s1
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1a
    0      0      0      0    0.0      0      0    0.0    0.0| ad0s1c
    9    705      0      0    0.0    705   9916   10.2   99.9| ad0s1d
```

Figura 18: Taxa de leitura e gravação em disco com *cache* de escrita em disco habilitado no Freebsd versão 7.3 virtualizado com Xen 4.0.

Apesar das taxas de gravação em disco terem ficado constantes após a habilitação da variável "hw.ata.wc" do sysctl do FreeBSD, o disco ficou com uso de 100% durante todo o tempo, fazendo também com que o consumo de banda ficasse muito baixo, como apresentado na Figura 19.

Com base nas análises feitas, o desempenho com esse modelo de configuração da máquina virtual ficou muito abaixo do esperado, inclusive não sendo possível sua utilização de forma prática, visto que as limitações foram muito grandes, impedindo o bom desempenho dos *proxies*.

6.1.5 FreeBSD Paravirtualizado com Xen

Uma última tentativa de utilização do FreeBSD (versão 8.0) com o Xen foi a utilização do modelo paravirtualizado, que tende a um desempenho melhor que o modelo de virtualização total.

Anteriormente, o Xen havia sido atualizado para a versão 4.0 para prover a compatibilidade

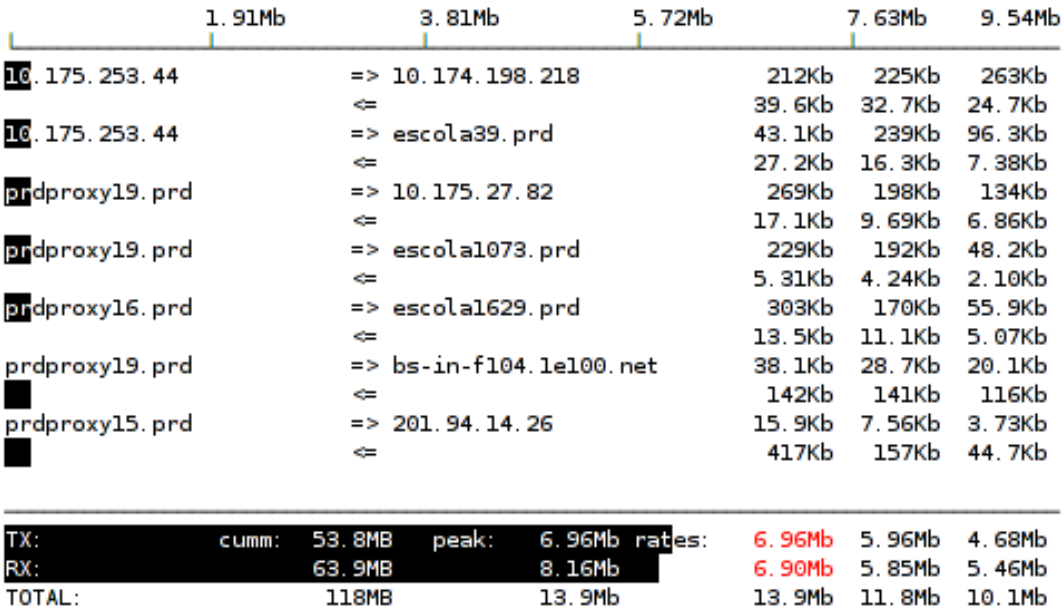


Figura 19: Consumo de banda com *cache* habilitado no Freebsd versão 7.3 virtualizado com Xen 4.0.

com uma placa de rede Gigabit emulada pelo QEMU (a Intel® PRO/1000 PCI-Express Gigabit Ethernet, como citado na Subseção 6.1.2). Porém, o FreeBSD versão 8.0 não dispõe de suporte ao Xen 4.0 para máquinas virtuais, sendo necessária novamente a utilização da versão 3.2, nativa do Debian Lenny, para efetuar os testes.

No primeiro caso, foi testado o FreeBSD versão 8.0 para arquitetura i386, que possui suporte para o modelo de paravirtualização, através da ativação da opção “XEN” no seu *kernel* (necessitando ser recompilado para que ele possua esse suporte) ou da utilização da configuração disponível juntamente com sua distribuição para fazer isso (quando instalado o pacote “sys” da árvore “src” do FreeBSD), a configuração pode ser encontrada dentro da própria distribuição do FreeBSD, no arquivo /usr/src/sys/i386/conf/XEN).

Com a utilização dessa configuração, é necessário também habilitar a opção “PAE” do *kernel*, uma modificação do *kernel* i386 para suportar o endereçamento de mais de 4GB de RAM, que é uma limitação dos sistemas 32 bits, sendo que sistemas 64 bits podem endereçar até 16 EB de RAM, desde que o sistema operacional também possua suporte. A Equação 1 demonstra o cálculo necessário para se obter o tamanho máximo de endereçamento de memória para um sistema de 32 bits, e a Equação 2 demonstra o cálculo para um sistema de 64 bits.

$$2^{32} = 4294967296 = 4GB \quad (1)$$

$$2^{64} = 18446744073709551616 = 16EB \quad (2)$$

Após o sistema ter sido instalado como uma máquina hóspede e o *kernel* compilado de acordo com as necessidades citadas anteriormente, a máquina foi iniciada no modelo paravirtualizado, entretanto, o suporte ao multiprocessamento não funcionava, sendo apenas um núcleo reconhecido pelo sistema operacional, e após algumas pesquisas, um relato foi encontrado de que o suporte a multiprocessamento para o FreeBSD como máquina virtual do Xen é problemático (CHADD, 2010), então os testes foram descontinuados.

Por fim, mais alguns testes foram feitos com o FreeBSD versão 8.0 para arquitetura amd64. Porém, o FreeBSD versão 8.0 não possui suporte a paravirtualização, apenas suporte a virtualização total com QEMU (ou HVM - *Hardware-assisted virtualization*). Existe uma configuração de *kernel* específica para isso (encontrada no arquivo /usr/src/sys/amd64/conf/XENHVM, quando instalado o pacote "sys" da árvore "src" do FreeBSD), mas nos testes realizados, não foi possível colocá-lo em execução por diversos problemas para os quais soluções ou explicações não foram encontradas.

6.1.6 NetBSD Paravirtualizado com Xen

Como não foi possível virtualizar o FreeBSD com o Xen para atender às necessidades dos *proxies*, uma outra alternativa foi testada: um sistema operacional também da família BSD e *software* livre, chamado NetBSD.

O NetBSD desde a versão 3.0 possui suporte nativo à paravirtualização com Xen (na época em que foi lançado, o suporte era ao Xen versão 2.0) (THE NETBSD PROJECT, 2009) e a versão 5.0.2, a mesma utilizada nos testes realizados nesta subseção e também mais atual, possui também suporte ao Xen versão 3.x.

Após a instalação do sistema em uma máquina hóspede, também constatou-se que o problema do multiprocessamento encontrado no FreeBSD repetia-se para o caso do NetBSD, ou seja, não há suporte funcional para executar tal tarefa atualmente (BOUYER, 2009), e devido à carga esperada no servidor, os demais testes foram descontinuados.

6.2 VIRTUALIZAÇÃO COM KVM

Devido aos inúmeros problemas encontrados ao se virtualizar as máquinas utilizando o Xen como solução, não foi possível completar os testes com outros sistemas operacionais que não o

próprio Linux. Portanto, uma outra solução de virtualização teve de ser testada para que fosse possível avaliar as diferenças de desempenho entre o Linux e outro sistema operacional com características similares, como o FreeBSD.

O KVM (Kernel-based Virtual Machine ou, em português, Máquina Virtual baseada no Kernel) é uma solução de virtualização total para Linux com *hardware* x86 contendo extensões de virtualização (Intel VT ou AMD-V). Ele consiste de um módulo carregável do *kernel*, chamado kvm.ko, que provê o núcleo de infraestrutura de virtualização e um módulo específico para o processador, kvm-intel.ko ou kvm-amd.ko. O KVM também requer uma versão modificada do QEMU. Entretanto, há um trabalho sendo feito para se disponibilizar as modificações necessárias na versão oficial do QEMU (REDHAT, 2010a).

O KVM pode executar múltiplas máquinas virtuais rodando vários sistemas operacionais sem necessidades de modificação, dentre eles o próprio Linux, o Windows e o FreeBSD. Cada máquina virtual possui *hardware* virtualizado privado, como placas de rede, discos, etc (REDHAT, 2010a).

6.2.1 FreeBSD versão 8.0 virtualizado com KVM

Assim como na virtualização total com o Xen, o KVM, por utilizar também o QEMU como base para virtualização do *hardware*, possui as mesmas limitações do primeiro. No Debian Lenny, a versão disponível do KVM é a 72, que é também baseada no QEMU 0.9.1, e conforme os estudos realizados na subseção 6.1.2, suas limitações não permitem que os testes sejam feitos com essa versão.

Para resolver os problemas do KVM e do QEMU, o Debian Backports, repositório que possui um conjunto de *software* mais atual que o da versão estável do Debian (como já citado anteriormente, atualmente é a 5.0, codinome Lenny), normalmente recompilado das versões teste ou instável do Debian, foi utilizado para a instalação de uma versão mais nova do KVM. O Backports do Debian 5.0 possui a versão 0.12.4 do KVM, agora disponível através do pacote chamado “qemu-kvm”, pois o KVM teve o nome de sua árvore mudado após a versão 88, ou seja, o KVM e suas atualizações foram integradas ao QEMU. A mais nova versão do “qemu-kvm” é também a versão que será utilizada no trabalho, versão 0.12.4. Para efeitos práticos, nesse trabalho, o “qemu-kvm” será chamado apenas de KVM.

Quando a máquina virtual foi configurada, foi também necessário especificar quais componentes seriam virtualizados, como o caso da placa de rede, utilizando-se o módulo “e1000” da Intel® PRO/1000 PCI-Express Gigabit Ethernet, bem como a especificação do tipo de disco,

utilizando-se o módulo para dispositivos SCSI, para que a velocidade de leitura e gravação em disco não seja comprometida.

O KVM possui um módulo próprio para virtualizar os dispositivos de entrada e saída, como placas de rede e discos, chamado “virtio”, que foi escolhido para ser a plataforma principal para a virtualização de *I/O* (REDHAT, 2010b). Entretanto, atualmente não existem módulos para dispostivos do tipo “virtio” para o FreeBSD e portanto é necessário utilizar módulos genéricos para dispositivos de entrada e saída, como o “scsi” para discos e “e1000” para interfaces de rede.

Para a virtualização com o KVM, é possível especificar o tipo do disco virtual que será utilizado. Os mais comuns são o “qcow2”, que é o formato padrão do QEMU, e o “raw” (ou cru, em português) que é apenas um arquivo formatado como se fosse um disco físico.

Nesse teste, a escolha do tipo de disco virtual ideal é importante para que não haja perda significativa de desempenho. Portanto, antes de continuar com testes mais profundos, os dois tipos de arquivos foram devidamente comparados para que essa característica não fosse afetada.

Além do tipo do disco virtual, existe também uma propriedade chamada de *cache* para esses discos, que possui três opções de configuração, sendo elas “none”, “writeback” e “write-through”.

Com o “writethrough” (padrão no QEMU para o tipo “raw”), sempre que o sistema grava dados, a máquina virtual envia os dados ao *cache* do hospedeiro e aguarda até que ele indique que a gravação no dispositivo de armazenamento tenha sido concluída. No caso do “writeback” (padrão do QEMU para o tipo “qcow2”), o sistema entenderá como completa a gravação assim que os dados estiverem presentes no *cache* do hospedeiro, ou seja, assim que os dados estiverem aguardando na fila para serem sincronizados no dispositivo. Por fim, quando o *cache* é desativado, através da opção “none”, este é completamente ignorado, fazendo com que a máquina virtual grave diretamente no disco, não dependendo do hospedeiro para intermediar tal ação (BELLARD, 2010).

Os testes foram realizados com os tipos de disco “raw” e “qcow2”, as três configurações de *cache* foram testadas e, sendo a maior parte do acesso ao disco para gravação, como estudado anteriormente na subseção 3.2.2, um arquivo de 512 MB foi gravado em disco para testar-se a velocidade dos mesmos, conforme as Tabelas 7 e 8.

As Tabelas 7 e 8 mostram os tempos e taxas de gravação atingidas para o FreeBSD virtualizado com o KVM. Através desses dados, pode-se justificar a escolha do tipo de disco “raw”, configurado com a opção “writeback” para o *cache*, pois mostrou-se a mais eficaz em termos

Tabela 7: Tempo médio para gravação em disco de um arquivo de 512 MB no FreeBSD virtualizado com KVM.

Tipo de Disco	Writethrough	Writeback	None
Raw	130 s	2.2 s	60 s
Qcow2	578.5 s	3.7 s	298.3 s

Tabela 8: Taxa média de gravação em disco de um arquivo de 512 MB no FreeBSD virtualizado com KVM.

Tipo de Disco	Writethrough	Writeback	None
Raw	3.94 MBps	230 MBps	8.53 MBps
Qcow2	885 KBps	138 MBps	1.72 MBps

gerais. Obviamente, esse foi um teste simples e rápido apenas para concluir qual deles tende a ser mais rápido, o que não quer dizer necessariamente que para determinadas situações, como a de gravação de vários arquivos pequenos ou mesmo a leitura do disco seja também a mais eficiente.

Com as definições necessárias já feitas, deste ponto em diante serão realizado os testes e comparativos de desempenho.

Após as configurações da máquina virtual terem sido feitas, utilizando-se o tipo de disco "raw" com a opção de *cache* "writeback", juntamente com a emulação de disco SCSI e interface de rede Intel® PRO/1000 PCI-Express Gigabit Ethernet, o FreeBSD versão 8.0 para arquitetura amd64 foi instalado, e logo em seguida, algumas limitações foram percebidas.

A primeira limitação é relativa à quantidade permitida de arquivos abertos, ou seja, quantos arquivos o sistema permite que sejam acessados simultaneamente (o que não quer dizer que eles estejam sendo lidos ou escritos ao mesmo tempo, mas sim o fato de apenas estarem abertos pelos aplicativos). Por padrão, esse limite é de 12328 arquivos e foi logo aumentado para 65535, através da variável "kern.maxfiles" para que o Squid não seja afetado por essa limitação.

Uma segunda limitação encontrada foi a quantidade máxima de portas abertas, que não é definida por um único valor como é o caso do "kern.maxfiles". Neste caso, o limite é definido através dos limites mínimos e máximos. No IPv4 (versão utilizada atualmente do protocolo IP na maioria das redes, incluindo a utilizada nesse trabalho), o número das portas possui como limite inferior a porta 0 e como limite superior a 65535. Portanto, numa mesma máquina, pode-se ter no máximo 65536 portas abertas, ou 65536 conexões simultâneas estabelecidas. Dessa forma, tendo-se, por exemplo, 20 mil clientes conectados, se precisaria de, no mínimo, 20 mil portas abertas. No caso do FreeBSD, o limite inferior padrão é a porta 49152, dado pela variável "kern.inet.ip.portrange.first" e o limite superior padrão é a porta 65535, dado pela

variável “kern.inet.ip.portrange.last”. Para um servidor mais robusto e que possa atender a demanda existente, a porta inferior foi definida como a de número 1024, permitindo assim mais de 64 mil clientes simultâneos. Como comparativo, o Debian versão 5.0 limita, através da variável “net.ipv4.ip_local_port_range”, a porta inferior a 32768 e a superior a 61000.

Apenas a título de informação, nos horários de pico, cada um dos servidores atinge mais de 10 mil conexões simultâneas, ou seja, um total de mais de 20 mil conexões para os serviços de *proxy*.

Já com as configurações citadas anteriormente, o *proxy* foi ativado e testado durante um dia para que seu desempenho fosse avaliado e o resultado obtido está representado na Tabela 9.

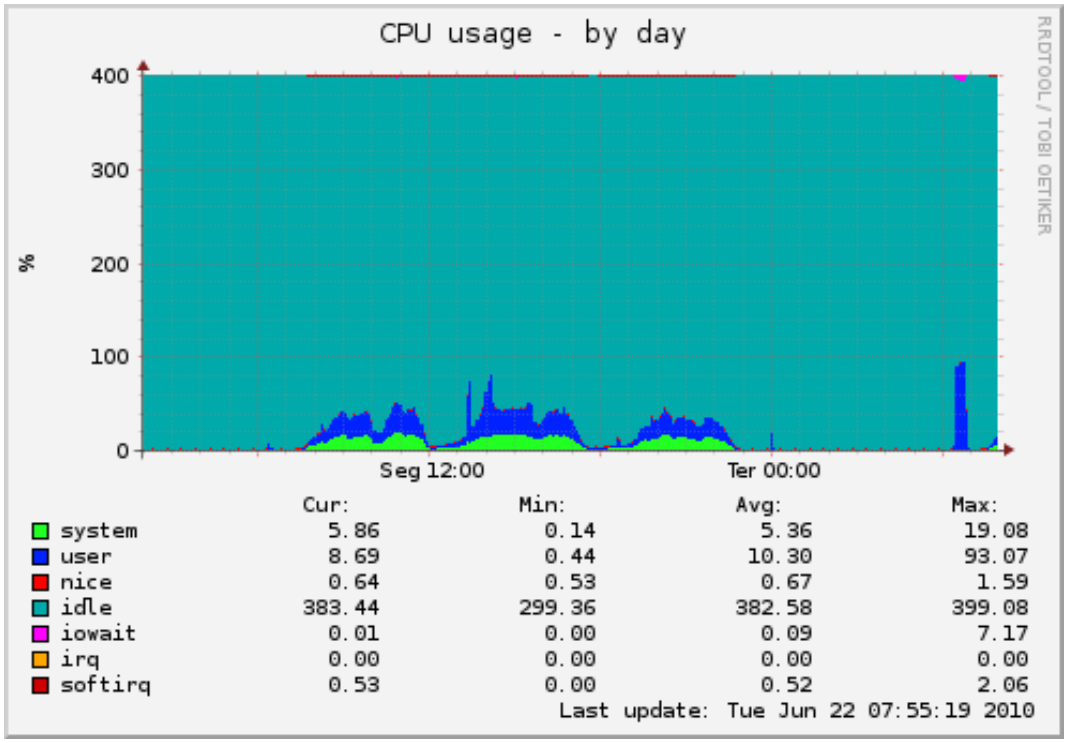
Tabela 9: Tempo médio dos acessos do Squid comparando-se o Linux ao FreeBSD versão 8.0, servidores 1 e 2, respectivamente, virtualizado com KVM, no período de 24 horas, com 512 MB de *cache* em memória.

	Total de Acessos	Tempo Médio TCP_MEM_HIT	Tempo Médio TCP_HIT	Tempo Médio TCP_DENIED	Tempo Médio TCP_MISS
Servidor 1	31.611.402	88 ms	120 ms	36 ms	381 ms
Servidor 2	30.679.468	7 ms	130 ms	10 ms	206 ms

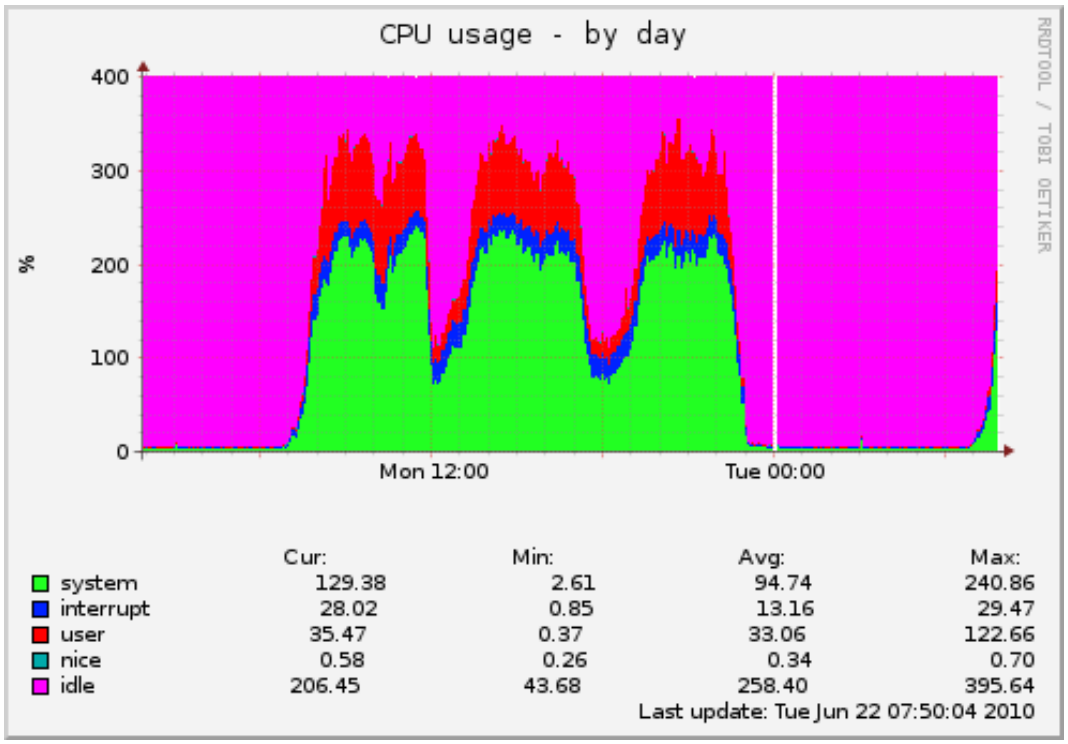
Conforme os dados apresentados na Tabela 9, o número total de acessos dos dois servidores *proxy* é muito similar. Entretanto, o tempo médio de acesso do servidor 2, com o FreeBSD versão 8.0, em quase todas as situações é muito inferior ao do servidor 1 que está utilizando o Linux. O valor que mais impressiona é relativo ao tempo médio de acesso ao *cache* em memória, que se mostrou mais de 10 vezes mais rápido ao servidor com Linux. Além do *cache* em memória, o tempo médio de acesso ao conteúdo proibido teve um ganho de quase 3 vezes e o acesso ao conteúdo que não estava em *cache* teve ganho de desempenho de 85%, um ganho também considerável.

No caso do conteúdo armazenado no *cache* em disco, o servidor 2, que utiliza o FreeBSD, apresentou um tempo médio de acesso maior que o servidor com Linux, ainda que a diferença do tempo tenha sido baixa. Entretanto, percebeu-se que a quantidade de objetos recuperados desse *cache* foi muito maior no servidor 2 do que no servidor 1. No servidor com Linux, o número total de arquivos recuperados do *cache* em disco foi de 1.275.021 e no servidor com FreeBSD foi de 3.198.583, ou seja, o FreeBSD teve 1,5 vezes mais arquivos recuperados que o Linux, justificando o tempo de acesso mais elevado.

Como visto anteriormente, o ganho de desempenho no FreeBSD foi considerável, se comparado ao servidor com Linux. Apesar desse ganho de desempenho, o consumo de processamento do servidor FreeBSD foi muito superior ao do servidor Linux, como mostra a Figura 20.



(a) Servidor com Linux



(b) Servidor com FreeBSD

Figura 20: Comparativo de consumo de processamento entre o Linux e o FreeBSD versão 8.0 virtualizado com KVM.

O consumo total de processamento mostrado na Figura 20, em horários de pico, chega a quase 400% de uso (que é o equivalente a 100% multiplicado pelo número total de 4 núcleos

para esse caso, conforme explicado anteriormente na Subseção 3.2.2), sendo que no servidor Linux chega a pouco mais de 90% (menos de 25% do total de processamento disponível).

Deve-se lembrar que o servidor Linux está utilizando o modelo de paravirtualização e é virtualizado utilizando-se o Xen 3.2. Já o FreeBSD está virtualizado utilizando-se o modelo de virtualização total com o KVM, portanto é natural que ele apresente maior consumo de recursos da máquina, ainda que esse maior consumo seja revertido em um ganho de desempenho no quesito tempo de resposta.

Considerando-se o consumo de processamento médio durante o dia, o servidor com FreeBSD possui consumo de mais de 140%, já o servidor com Linux consome pouco mais de 15% no total e apesar de existir um ganho no tempo de resposta, houve um reflexo muito alto no quesito custo. Sendo assim, é necessário que uma modificação como essa, que possui um reflexo negativo de tal escala no consumo de processamento, seja cautelosamente estudada pelo administrador antes de sua efetiva implementação.

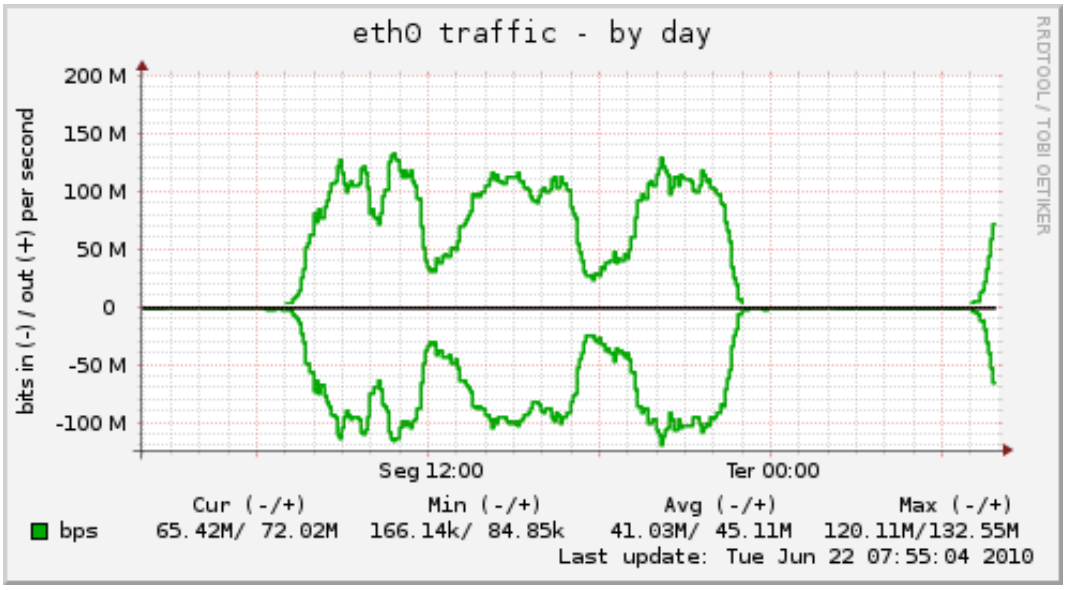
Como pode ser observado na Figura 20, o maior consumo de processamento do servidor com FreeBSD é do sistema (*system*). O consumo do sistema abrange leitura e gravação em disco, tráfego de entrada e saída de rede, dentre outras características do sistema. Apesar de não haver nenhum dado concreto de qual recurso estava consumindo todo o processamento do sistema, os mais prováveis seriam uso de disco e de rede, baseando-se no entendimento da finalidade do serviço de *proxy*.

O tráfego de saída de rede mostrado na Figura 21 está parecido para os dois servidores. Entretanto, no caso do servidor com FreeBSD, o consumo médio de entrada de rede é superior em 40%. O motivo disso ocorrer também não é claro, mas há duas hipóteses que se sugere aqui como esclarecimento. A primeira hipótese é de que poderia haver grande perda de pacotes de rede que chegam ao servidor, portanto gerando retransmissão e consequente elevação do consumo de rede. A segunda hipótese é de que haja comunicação interprocessos (relativo a comunicação do Squid com processos do sistema ou ainda, dos processos do sistema entre si).

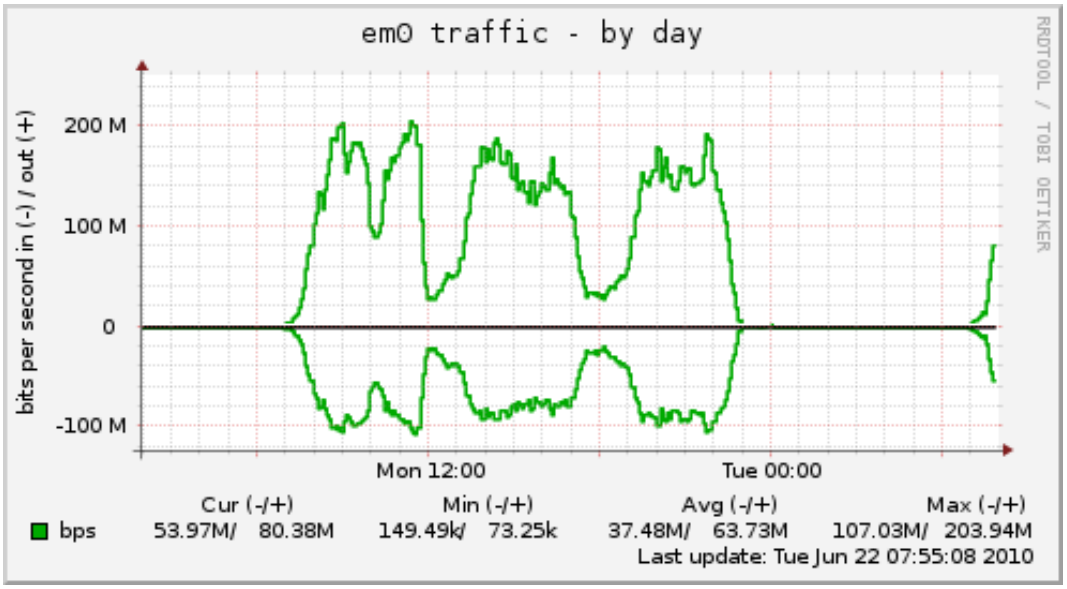
Para tentar reduzir o quadro de maior utilização de processamento do servidor com FreeBSD, utilizando-se dos conhecimentos obtidos no Capítulo 4, um serviço de DNS foi instalado no servidor com FreeBSD, visando a redução do consumo de rede e, consequentemente, de processamento.

A Tabela 10 mostra os resultados do tempo médio de acesso para o servidor, utilizando-se de um servidor DNS também no mesmo servidor.

Apesar do tempo médio de acesso ao *cache* em memória ter sido maior com o DNS local



(a) Servidor com Linux



(b) Servidor com FreeBSD

Figura 21: Comparativo de consumo de banda de rede entre o Linux e o FreeBSD versão 8.0 virtualizado com KVM.

(18 ms, conforme mostra a Tabela 10, contra os 7 ms medidos com DNS externo, como anteriormente visto na Tabela 9), que é o tempo médio de acesso que melhor reflete o ganho pela existência de um DNS local devido ao seu baixo tempo de resposta, pode-se ressaltar que 84% dos acessos ao *cache* em memória levaram menos de 2 ms para serem respondidos, enquanto que durante o dia em que o servidor utilizava um DNS externo, apenas 74% desses acessos foram respondidos em menos de 2 ms.

Quanto ao consumo de processamento, a Figura 22 mostra que não houve grande diferença de quando o servidor não possuía o DNS local, como já foi visto na Figura 22.

Tabela 10: Tempo médio dos acessos do Squid comparando-se o Linux ao FreeBSD versão 8.0, servidores 1 e 2, respectivamente, virtualizado com KVM, no período de 24 horas, com 512 MB de *cache* em memória e DNS local.

	Total de Acessos	Tempo Médio TCP_MEM_HIT	Tempo Médio TCP_HIT	Tempo Médio TCP_DENIED	Tempo Médio TCP_MISS
Servidor 1	29.055.419	23 ms	251 ms	33 ms	469 ms
Servidor 2	28.675.787	18 ms	134 ms	12 ms	322 ms

Para o consumo de banda de rede, também não houve grande diferença, especialmente na banda de saída de rede, que continua sendo aproximadamente 100% maior, como se pode ver na Figura 23.

6.2.2 FreeBSD Versão 7.3 Virtualizado com KVM

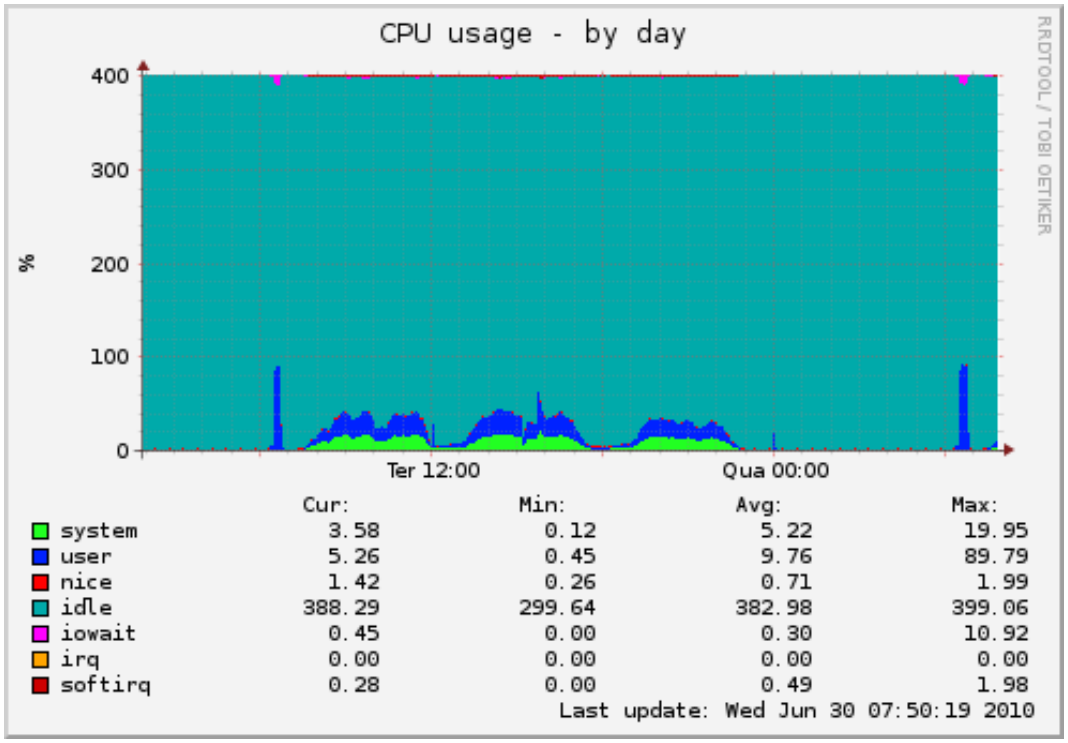
Na tentativa de avaliar diferenças no desempenho do FreeBSD, testes com as arquiteturas i386 e amd64 do sistema foram realizadas, conforme os relatos no decorrer dessa subseção.

A primeira arquitetura testada é a i386 e após a instalação e configuração de todo o sistema com sucesso, havia ainda a necessidade de se compilar o *kernel* com a opção PAE habilitada para fornecer suporte ao endereçamento de mais de 4 GB de memória RAM, visto que sistemas de 32 bits não conseguem endereçar mais do que esses 4 GB de memória, como já anteriormente justificado na subseção 6.1.5.

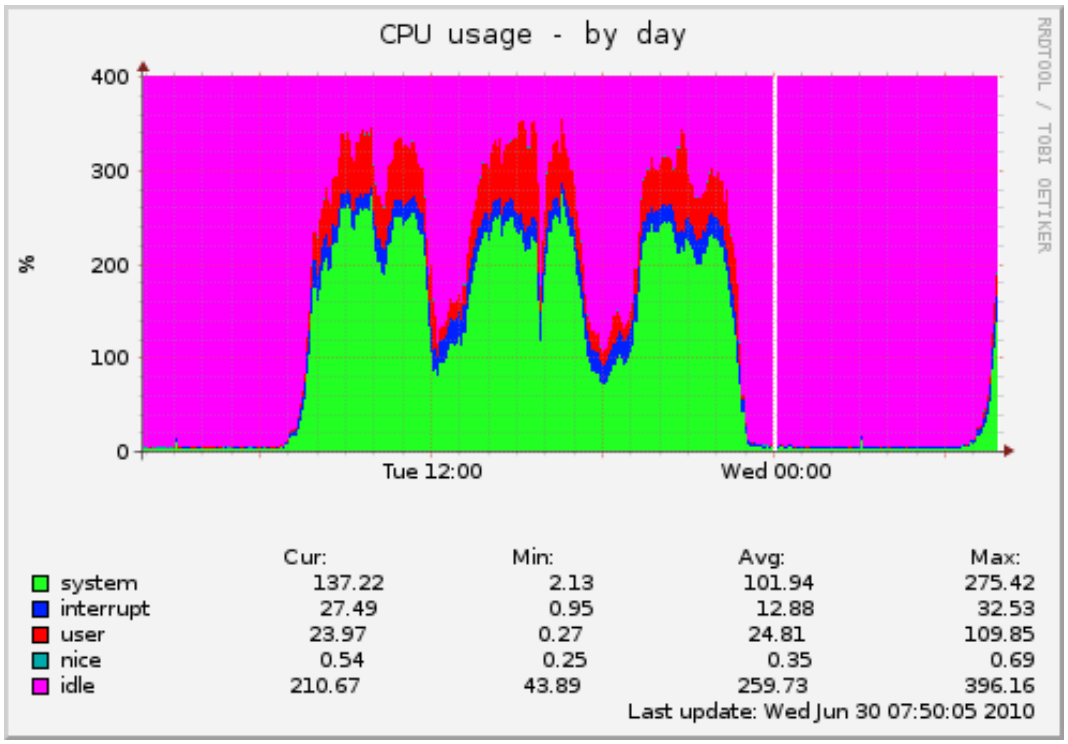
Ao se abrir o arquivo de configuração do *kernel* com suporte a PAE, um comentário com o seguinte alerta foi encontrado: Abaixo há uma lista de drivers que normalmente estão na configuração GENERIC, porém eles não funcionam ou não foram testados com o PAE. Seja muito cuidadoso antes de habilitar qualquer um destes drivers. Drivers que usam DMA e não gerenciam endereços físicos de 64 bits corretamente podem causar corrompimento de arquivos quando usados em uma máquina com mais de 4 GB de memória (SMITH, 2010).

Entre os drivers que constavam do alerta estava o driver para o disco SCSI emulado pelo KVM, através do dispositivo do *kernel* nomeado “sym”. Como o objetivo do trabalho é justamente testar a melhor opção, o teste foi realizado, utilizando-se o driver “sym”, mas sem sucesso, pois apesar do reconhecimento do disco, não era possível montar nenhum dos sistemas de arquivos contidos no mesmo.

Já o FreeBSD versão 7.3, arquitetura amd64, foi instalado diversas vezes na tentativa de avaliar diferenças no desempenho para a versão 8.0 com o KVM. Entretanto, durante a configuração do servidor, os sistemas de arquivos sofriam corrompimento por causas que não puderam ser constatadas. Acredita-se que o mesmo driver “sym” não funcione corretamente com mais



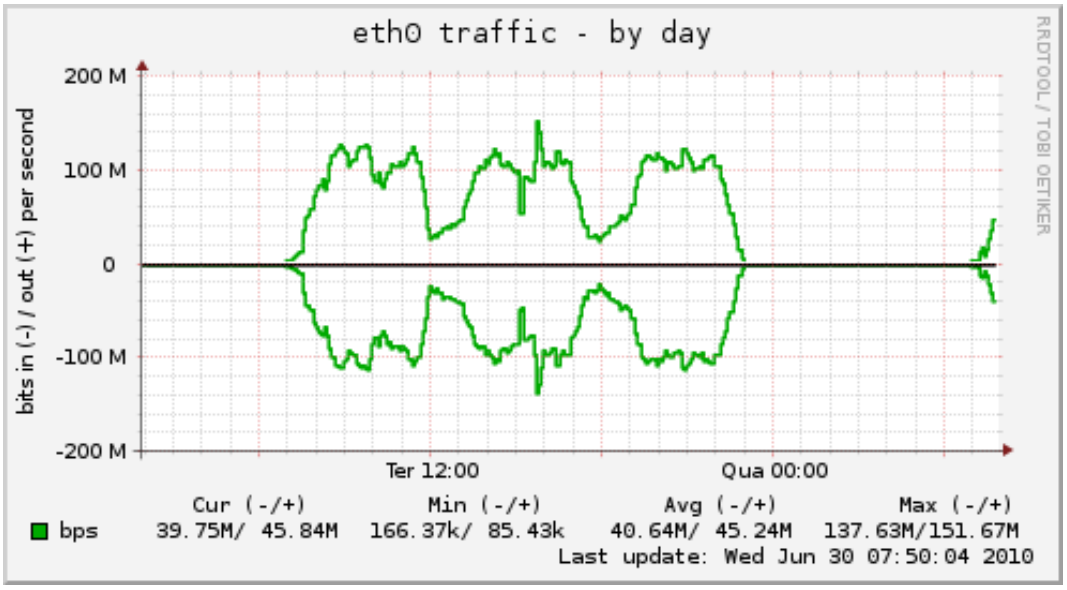
(a) Servidor com Linux



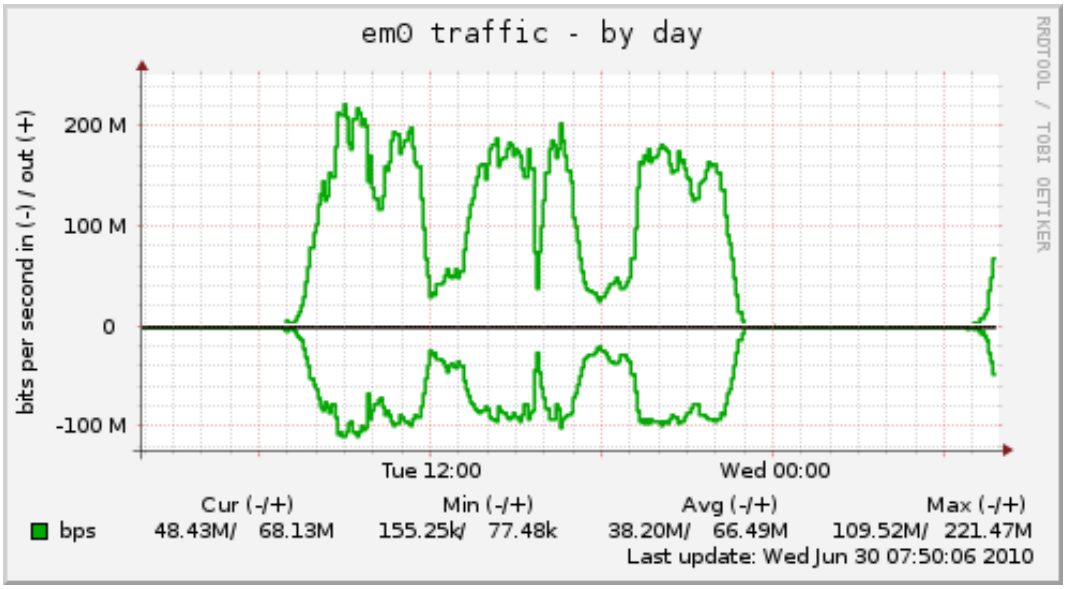
(b) Servidor com FreeBSD

Figura 22: Comparativo de consumo de processamento com DNS local entre o Linux e o FreeBSD versão 8.0 virtualizado com KVM.

de 4 GB de RAM mesmo na arquitetura amd64 do FreeBSD na versão 7.3.



(a) Servidor com Linux



(b) Servidor com FreeBSD

Figura 23: Comparativo de consumo de banda de rede com DNS local entre o Linux e o FreeBSD versão 8.0 virtualizado com KVM.

6.2.3 NetBSD Versão 5.0.2

Apesar da instalação e configuração do NetBSD 5.0.2 para arquitetura amd64 ser bem sucedida, incluindo a compilação do *kernel*, elevando-se o número máximo de usuários simultâneos e arquivos abertos, de 64 para 1024 e de 128 para 131072, respectivamente, não houve sucesso durante os testes dos *proxies* devido a dois problemas que ocorreram. O primeiro deles é que, durante a inicialização dos processos das dez instâncias do Squid, o processamento da máquina era totalmente consumido pelos processos dos Squid, como mostra a Figura 24, na maioria das

vezes ocasionando até o “congelamento” da máquina, e o segundo, como pode ser observado na Figura 25 é que a banda de saída ficava muito baixa (menos de 10% do total da banda de entrada), ocasionando lentidão na resposta aos clientes.

```
load averages:  5.04,  1.36,  0.51;               up 0+00:04:26       13:54:18
59 processes: 7 runnable, 38 sleeping, 10 zombie, 4 on CPU
CPU0: 27.7% user,  0.0% nice, 70.1% system,  2.2% interrupt,  0.0% idle
CPU1: 89.2% user,  0.0% nice, 10.8% system,  0.0% interrupt,  0.0% idle
CPU2: 79.0% user,  0.0% nice, 21.0% system,  0.0% interrupt,  0.0% idle
CPU3: 72.3% user,  0.0% nice, 27.7% system,  0.0% interrupt,  0.0% idle
Memory: 839M Act, 390M Inact, 1416K Wired, 9388K Exec, 25M File, 6616M Free
Swap: 1024M Total, 1024M Free

  PID USERNAME PRI NICE   SIZE   RES STATE      TIME   WCPU    CPU COMMAND
  562 root      26    0    99M   86M RUN/2      0:08 45.47% 27.00% squid
  406 root      25    0   100M   88M CPU/0      0:08 35.99% 24.02% squid
  793 root      77    0    97M   84M RUN/0      0:07 35.13% 22.85% squid
  450 root      79    0   100M   88M RUN/0      0:07 36.52% 22.41% squid
  780 root      78    0   101M   89M RUN/0      0:07 37.66% 22.36% squid
  160 root      28    0    59M   45M RUN/0      0:06 35.17% 21.58% squid
  799 root      26    0    56M   41M CPU/3      0:06 33.97% 20.85% squid
  765 root      26    0    59M   45M RUN/2      0:06 30.79% 18.90% squid
  892 root      25    0    47M   33M CPU/1      0:04 51.02% 18.51% squid
  536 root      26    0    47M   33M RUN/3      0:04 42.40% 16.70% squid
  154 root      43    0    66M   34M parked/1   0:00  0.05%  0.05% named
```

Figura 24: Consumo de processamento no NetBSD com KVM.

A Figura 24 mostra o consumo de processamento da máquina virtual com o NetBSD no momento em que os processos das dez instâncias do Squid foram iniciadas, ficando então os 4 núcleos 0.0% ociosos. Desse momento em diante, a máquina apresentava lentidão para executar quaisquer outras tarefas, além de muitas vezes ser “congelada”, não sendo mais possível executar nenhuma ação, forçando o seu desligamento.

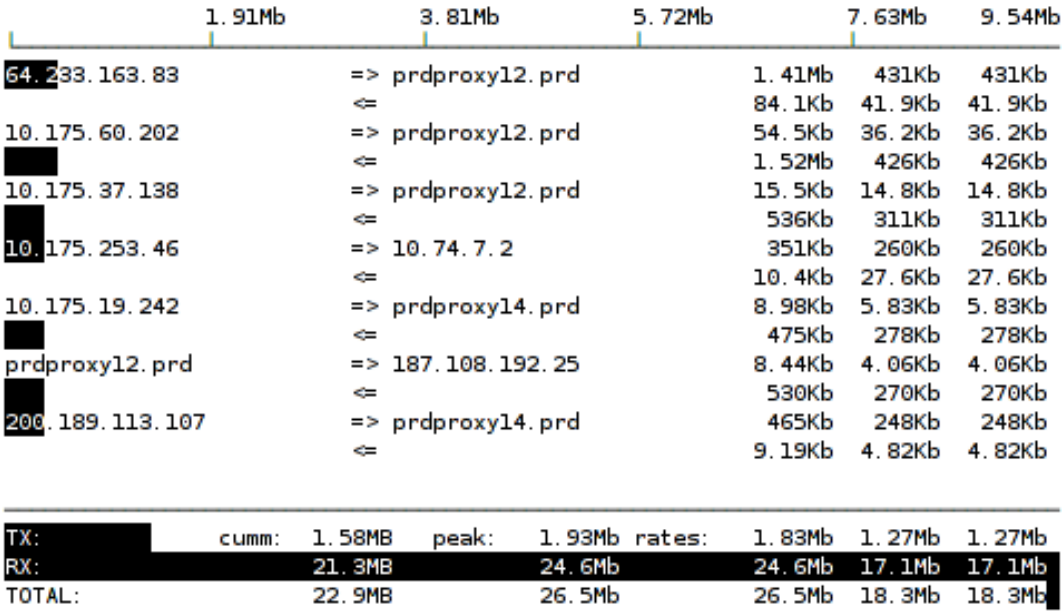


Figura 25: Consumo de banda de entrada e saída no NetBSD com KVM.

7 ECHOPING PROXY - MODELO DE COLETA DE DADOS PARA O CACTI

Para um controle ainda maior dos servidores *proxies* e melhor visualização do desempenho da rede, uma ferramenta na forma de um *template* foi desenvolvida no decorrer deste trabalho com base no *script* em Perl originalmente desenvolvido por Sébastien Desse (o script está no Apêndice B. Essa ferramenta trata-se na realidade de um modelo de coleta de dados, pronto para ser utilizado com os mesmos *proxies* que são objetos de estudo neste trabalho. Entretanto, o Echoping Proxy pode ser usado com qualquer servidor *proxy* e não apenas com os que são objetos de estudo deste trabalho.

Essa ferramenta é baseada no aplicativo echoping, que permite mensurar o tempo para um determinado acesso TCP ou UDP. O modelo de coleta desenvolvido para o Cacti, nada mais é do que uma interface gráfica para o echoping, que além de mensurar o tempo de acesso utilizando-se um servidor *proxy* para acessar o *site* de destino, também mantém registros do comportamento do servidor *proxy* ao longo do tempo, função que é o real objetivo do monitoramento de equipamentos com o Cacti.

O echoping é um pequeno programa desenvolvido por Stéphane Bortzmeyer para testar (aproximadamente) o desempenho de um computador remoto através do envio de requisições no formato HTTP (BORTZMEYER, 2008).

A ferramenta desenvolvida não tem o objetivo direto de melhorar o desempenho de um servidor *proxy*, trata-se apenas de uma ferramenta de monitoramento, ferramentas essas que são muito importantes na administração e gerenciamento de redes de médio e grande porte.

Apesar desse modelo não objetivar diretamente a melhoria de desempenho, ele fornece auxílio durante a mensuração do tempo de acesso, transformando os dados numéricos obtidos através do echoping em gráficos que poderão auxiliar o gerente da rede a avaliar o que nela ocorre. Como exemplo, se em dado momento há um aumento no tempo médio de acesso aos *sites* externos à rede, o gerente poderá ser avisado pelo Cacti que um determinado limite foi excedido, e partindo dessa importante informação, começar a investigação do problema, ou então, se o problema for recorrente, acontecendo sempre em um determinado horário do dia, o

gerente poderá avaliar o que ocorre na rede no horário em que o problema costuma ocorrer.

O funcionamento desse modelo de coleta de dados é muito simples: no Cacti, são configurados o site de destino, ou seja, aquele para o qual se deseja testar o tempo de resposta, o dispositivo que se objetiva avaliar e sua respectiva porta de entrada, no caso, o servidor *proxy* e a porta que o serviço está escutando, o número de tentativas, que é a quantidade de acessos que o *proxy* realizará para calcular o tempo médio.

O modelo desenvolvido é baseado em duas partes, a primeira delas e mais importante, é um *script* na linguagem Perl, originalmente desenvolvido por Sébastien Desse e modificado para os fins buscados para o modelo do Cacti, e a segunda parte, é o modelo de coleta de dados desenvolvido através da interface de administração do Cacti que posteriormente é convertido automaticamente para o XML (Extensible Markup Language).

7.1 INTERPRETAÇÃO DO GRÁFICO

Os gráficos gerados por esse modelo são similares ao da Figura 26.

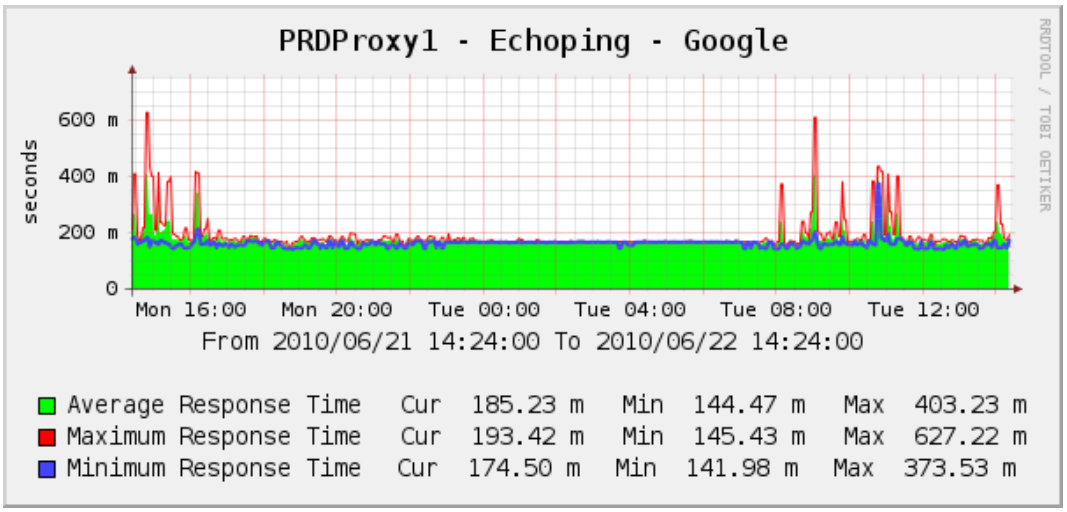


Figura 26: Gráfico gerado pelo Echoping Proxy.

Como pode-se observar na Figura 26, há três valores de tempo de resposta coletados, o valor médio ou *average*, o valor máximo ou *maximum* e o valor mínimo ou *minimum*. Dos três valores coletados pelo modelo, o mais importante e confiável é o valor médio, pois os valores mínimos e máximos podem ter sido afetados por alguma condição temporária da rede, que naquele exato momento acabou por gerar uma informação não tão precisa. Apesar de o valor médio ser o mais importante e trazer uma informação mais confiável, os outros valores foram mantidos no gráfico para mostrar o comportanto do *proxy* em todos os aspectos.

Para entender o gráfico da Figura 26 deve-se saber que ele possui três cores, azul, verde e vermelho. O valor mais importante do gráfico, que é o valor do tempo de resposta médio é representado por uma área de cor verde, a cor azul representa o tempo mínimo de resposta e a linha vermelha representa o tempo máximo de resposta naquele dado instante.

Nos detalhes do gráfico, existem 9 valores que sempre são mostrados, ou seja, 3 valores para os tempos médio, mínimo e máximo, que indicam o tempo calculado na última coleta mostrada no gráfico (*Cur*), o tempo mínimo (*Min*) e o tempo máximo (*Max*). Esses três valores nada mais são que os dados mais significativos observados naquele determinado período mostrado pelo gráfico.

7.2 INSTALAÇÃO DO MODELO

Para instalar o modelo, basta entrar na página *Import Templates* e colar ou digitar o conteúdo do Apêndice A no campo *Import Template from Text* e selecioná-lo no campo *Import Template from Local File*, por fim pressionando o botão *save* no rodapé da página. Quando a importação for concluída, uma mensagem de conclusão da importação deverá ser exibida, como na Figura 27.

Figura 27: Importação do modelo de coleta de dados Echoping Proxy.

Assim que a importação tiver sido realizada com sucesso, é importante verificar na página *Data Input Methods* se o comando de entrada foi importado corretamente, pois foi observado que na versão 0.8.7e do Cacti, os delimitadores das variáveis (os caracteres que indicam início e fim das variáveis, '<' e '>', respectivamente) não são importados, ficando então o comando de entrada como abaixo:
perl path_cacti/scripts/echoping.pl numberoftestssiteproxyproxyport

Quando o correto seria:
perl <path_cacti>/scripts/echoping.pl <numberoftests> <site> <proxy> <proxyport>

Se tudo estiver correto após a importação, as seções *Data Input Methods* e *Input Fields* da página *Data Input Methods*, devem mostrar as informações como na Figura 28.

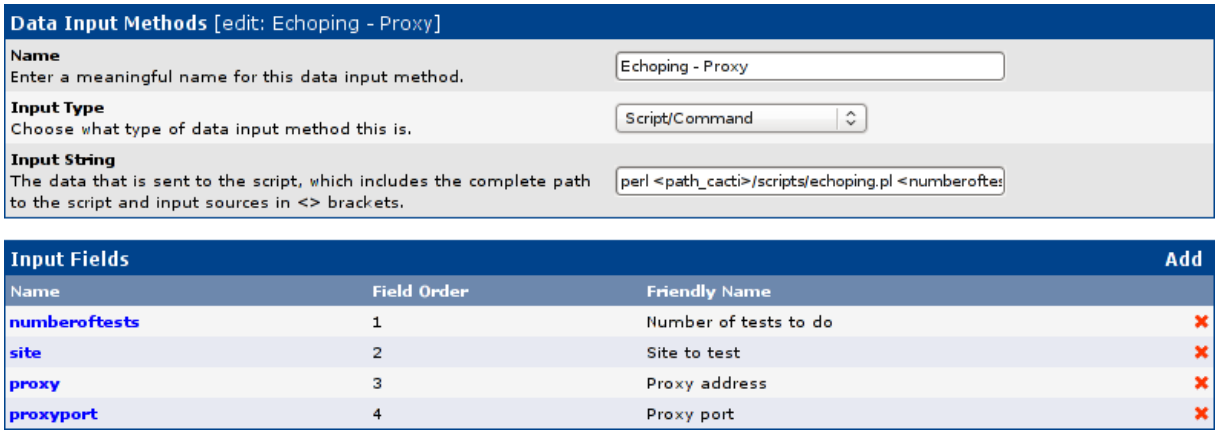


Figura 28: Configurações dos métodos de entrada de dados do modelo de coleta de dados Echoping Proxy.

Além da inclusão do modelo no Cacti, é necessário adicionar o *script* em Perl que efetivamente executará o aplicativo echoping e conectará ao servidor *proxy* para efetuar o teste e mensurar seu tempo de resposta.

Para adicionar o script, basta copiar o conteúdo do Apêndice B para o diretório de *scripts* do Cacti, que está localizado por padrão na raiz da instalação do Cacti, no subdiretório “scripts”, e nomear o arquivo como “echoping.pl”.

7.3 CONFIGURAÇÃO DO MODELO

O modelo possui quatro variáveis, são elas: numberoftests (configura o número de acessos que serão feitos em cada coleta de dados, configurada para três por padrão), site (endereço do site ao qual se deseja medir o tempo de acesso, por padrão http://www.google.com.br), proxy (endereço IP ou nome de domínio do servidor *proxy*, por padrão utiliza o IP ou nome de domínio do dispositivo para o qual a coleta de dados foi configurada) e proxyport (porta utilizada pelo servidor *proxy* para receber conexões dos clientes, 8080 por padrão). Essas quatro variáveis são configuráveis pelo administrador do Cacti.

As variáveis podem ser configuradas na página *Data Templates* da administração do Cacti, através da seção *Custom Data*, como mostrado na Figura 29.

Como já citado anteriormente, o primeiro dado variável passado ao Echoping Proxy é o número de testes que serão realizados (*Number of tests to do*), valor que é passado ao echoping e é encarregado de fazer com que o echoping execute o número de consultas especificado, para

Figura 29: Configuração das variáveis personalizadas do modelo de coleta de dados Echoping Proxy.

então calcular os valores mínimo, máximo e médio do tempo de resposta. O segundo dado é o endereço a ser testado (*Site to test*), sendo que o padrão é "http://www.google.com.br" (é importante que o endereçamento do protocolo http:// seja mantido para que o echoping funcione corretamente) por ser bastante difundido atualmente. A variável seguinte é o endereço do *proxy* (*Proxy address*) e ele pode ser o IP ou o nome de domínio do servidor ou ainda pode ser deixado em branco para que o gráfico seja gerado para o endereço do servidor para o qual o modelo de coleta de dados esteja sendo configurado. Por fim, o último dado é a porta na qual o servidor *proxy* está aguardando pela conexão dos seus clientes (*Proxy port*) e por padrão é a "8080", porém outro valor comum para servidores *proxy* é o valor "3128".

7.4 DISTRIBUIÇÃO DO MODELO EM FORMATO DIGITAL

Para obter o modelo em formato digital através da Internet, basta seguir o *link* abaixo:

http://www.entschev.com/files/echoping-proxy/echoping-proxy.tar.gz

Este *link* possui os dois arquivos necessários, comprimidos no formato Tarball-GZip, para fazer a instalação e a configuração do modelo, como indicados ao longo deste capítulo. Seguindo este endereço, sempre será encontrada a versão mais atual dos arquivos.

8 CONCLUSÃO

Este trabalho testou maneiras de se obter um melhor desempenho do *hardware* de um servidor *proxy* bem como do serviço de *proxy* através do refinamento de *software*. O conjunto dos servidores aqui estudados possui mais de 40 mil usuários simultâneos e chega a atingir taxas de consumo de banda de rede de mais de 200 Mbps em horários de pico, pois esse conjunto de servidores e serviços é utilizado pelo Projeto Paraná Digital, que fornece computadores e acesso à Internet às mais de duas mil escolas estaduais do Paraná.

Pode-se destacar no trabalho o uso de ferramentas totalmente livres e gratuitas para atingir os objetivos, dentre elas, os sistemas operacionais Linux, FreeBSD e NetBSD, o Xen e o KVM para realizar a virtualização dos servidores, o Munin e o Cacti para monitorar e gerar gráficos de consumo das máquinas, o Bind para a disponibilização de um DNS local e é claro, o Squid, que é o principal *software* estudado e provê o serviço de *proxy*.

O estudo do *cache* do Squid no Capítulo 3 mostrou que aquele localizado em disco, apesar do tamanho total de *cache* configurado para ser armazenado em disco ser razoavelmente alto, possui uma taxa de leitura muito baixa. Essa baixa taxa de leitura acaba nem mesmo aparecendo em gráficos de leitura e gravação em disco, visto que a taxa de gravação é alta devido à grande quantidade de *logs* de acesso gerada pelo Squid.

Entre os sistemas de arquivos XFS e ReiserFS também não se observou nenhuma diferença significativa nas taxas de leitura e gravação ou mesmo no consumo de processamento, o que leva à conclusão de que as máquinas não sofreram nenhuma influência negativa ou positiva direta apenas pela diferença do sistema de arquivos, não gerando os resultados que eram esperados no início do trabalho.

Ainda com os dados observados no Capítulo 3, pode-se concluir que, à medida que o número de acessos ao disco é reduzido (devido ao aumento do tamanho do *cache* em memória e o consequente crescimento da quantidade de acessos a ele), o tempo decorrido médio é também reduzido. Com relação ao tempo decorrido médio dos acessos em disco do caso estudado na Tabela 1, em que o *cache* em memória era de 256 MB, esse tempo foi reduzido em 20% a partir

do momento em que o espaço reservado em memória para o *cache* foi dobrado, permanecendo em 512 MB por instância do Squid e essa foi a única característica positiva encontrada na análise do *cache*, mas que também gera um alto custo, visto que memórias RAM são mais caras que espaço de armazenamento em disco.

O Capítulo 4 estudou o impacto dos servidores de resolução de nomes de domínios, mostrando que o tempo de resposta desse serviço é bastante afetado levando-se como característica a distância física (geográfica) e lógica (quantidade de equipamentos de rede) entre o servidor e o cliente.

Para serviços como o de um *proxy*, que constantemente necessita de resolução de nomes, o tempo gasto com esse serviço torna-se bastante relevante, pois não é a única ação que custará tempo de processamento ao servidor. Devido a isso, e como analisado durante todo o estudo realizado no Capítulo 4, conclui-se que o menor tempo de resposta foi de um serviço DNS utilizado na própria máquina onde o Squid era executado. Além disso, à medida que se aumentou a distância física e lógica entre os servidores *proxy* (que nesse caso é o cliente da resolução de nomes) e o servidor DNS, o tempo de resposta também se elevou. Os dados obtidos mostraram como menor tempo médio de resposta um período menor do que 1 ms, chegando até aproximadamente 444 ms, no caso de um servidor localizado a 14 mil quilômetros de distância. A utilização de um DNS local é uma das características que mostraram um grande benefício durante os estudos.

As listas de controle de acesso quando utilizadas com o SquidGuard estudadas no Capítulo 5 reduziram consideravelmente o tempo de resposta para as requisições de objetos que se encontravam em *cache*, tanto na memória quanto no disco, além das requisições que tiveram acesso negado. Apesar da redução do tempo de resposta, o consumo de processamento da máquina foi elevado em aproximadamente 1% no total, valor não muito significativo, mas que se associado a outros fatores, dependendo do *hardware* disponível e da quantidade de acessos, pode tornar-se um problema. A utilização do SquidGuard deve ser bem analisada antes de sua implementação, se o que se deseja é menor tempo de resposta, ainda que isso custe maior consumo do poder de processamento do *hardware*, não haverá problemas. Entretanto, se não há *hardware* suficiente disponível, há de se deixar as listas de controle de acesso para serem processadas pelo próprio Squid, com o custo de o tempo de resposta ser maior. Devido à essas características, o administrador do serviço *proxy* deverá avaliar bem as necessidades do ambiente e a infraestrutura disponível antes de tomar a decisão de usar o SquidGuard.

O Xen, apesar de ser uma ferramenta muito robusta, em constante processo de melhorias e ascensão do número de utilizadores como ferramenta de virtualização, possui ainda sérias

limitações em relação ao suporte a *hardware* fornecido para a virtualização total, mesmo em sua versão mais atual, como foi observado nas subseções 6.1.2 a 6.1.4.

Com relação à paravirtualização, o Xen já possui suporte bastante avançado no Linux, como observado no decorrer dos capítulos 3 a 5. Entretanto, para a maioria dos sistemas operacionais, o suporte ao Xen ainda está em processo de desenvolvimento, como visto nas subseções 6.1.5 e 6.1.6, em que falta suporte funcional para a arquitetura amd64 no caso do FreeBSD e também não há suporte a múltiplos núcleos para a máquina virtual tanto no FreeBSD quanto no NetBSD.

Com todas as informações que foram adquiridas na seção 6.1, percebe-se que o Xen é suficientemente estável para ser utilizado em máquinas hóspedes com o sistema Linux, mas para sistemas operacionais da família BSD, como os dois estudados neste trabalho (o NetBSD e o FreeBSD) para servidores de elevado consumo, que necessitam de *hardware* rápido e confiável, ainda há muito trabalho a ser feito antes que isso se torne uma realidade.

Na tentativa de continuar os testes comparativos entre o Linux e os sistemas operacionais da família BSD, uma outra alternativa testada foi o KVM, mostrando-se um pouco mais avançado em termos de suporte para esses sistemas se comparado ao Xen.

O que se pode constatar, analisando os estudos da subseção 6.2.1 é que em termos de tempo de resposta, o FreeBSD de uma forma geral mostrou-se muito superior ao Linux, mesmo sendo utilizado o modelo de virtualização total. Novamente, apesar desse elevado ganho no tempo de resposta, há um alto custo de *hardware* a se pagar, como maior consumo de processamento, de memória RAM e de disco, recursos esses que podem não estar disponíveis. Portanto, de forma geral, os testes realizados possuem mais pontos contra do que a favor da utilização do FreeBSD. Porém, novamente lembrando que o modelo de virtualização total foi usado, sendo que possivelmente numa máquina física ou no futuro, quando o FreeBSD puder ser paravirtualizado, talvez esse demasiado consumo de *hardware* seja muito mais baixo.

Devido ao *hardware* específico que é emulado pelo QEMU para discos SCSI e também às limitações do FreeBSD versão 7.3, muitos problemas foram encontrados, desde a falta de suporte ao dispositivo até o corrompimento de arquivos sem causa aparente, impossibilitando testes mais aprofundados.

Para o NetBSD, a virtualização e emulação de *hardware* também não teve sucesso, gerando alto consumo de recursos, "congelamento" da máquina e baixas taxas de transferência, mais uma vez impedindo que testes mais precisos fossem feitos para comparar seu desempenho ao desempenho obtido no Linux.

Devido aos problemas encontrados com a virtualização dos sistemas operacionais FreeBSD

e NetBSD não foi possível atingir o objetivo de realizar testes precisos de desempenho com o Squid em sistemas operacionais distintos ao Linux.

O monitoramento de serviços de rede é uma parte importante do refinamento de desempenho, não só de *proxies* mas de quaisquer serviços de rede. Muitas ferramentas foram apresentadas e com elas, essa importância foi constatada. Entretanto, hoje ainda há muito o que se explorar nessa área, e o Echoping Proxy que foi desenvolvido no decorrer do trabalho, descrito com detalhes no capítulo 7, é apenas uma pequena parte do que pode ser feito em relação ao monitoramento de serviços de rede. Nesse mesmo capítulo ainda foram fornecidas informações detalhadas sobre sua utilização e a parte construtiva do modelo.

A conclusão maior a que se chegou é de que toda configuração utilizada possui vantagens e desvantagens, e cada caso deve ser minuciosamente analisado até que se chegue a uma solução definitiva e de alto desempenho. No decorrer do trabalho, os pontos de maior relevância foram a ampliação do tamanho do *cache* armazenado em memória, da utilização do DNS local e do SquidGuard para listas de controle de acesso, sendo que cada uma dessas opções trouxe um benefício. O SquidGuard foi o único que refletiu negativamente se relacionado a outras características do sistema, sendo no caso o consumo de processamento, mas de qualquer forma trouxe também um benefício, permitindo que a relação custo/benefício seja analisada pelo administrador, possibilitando a ele optar por algum destes parâmetros.

Outro teste comparativo realizado que trouxe grande redução no tempo de resposta foi a utilização do FreeBSD versão 8.0 em substituição ao Linux. Entretanto, da mesma forma que trouxe um grande benefício, teve um alto custo, consumindo muito mais processamento do que o servidor em Linux, mas sempre lembrando que o FreeBSD utilizou a virtualização total com o KVM, que tende a consumir mais recursos de *hardware*. Dessa forma, mais uma vez a relação custo/benefício deverá decidir qual o melhor sistema a se utilizar.

Ainda há muito a ser estudado no que diz respeito a refinar *software* para aproveitar ao máximo todo os recursos disponíveis de *hardware*, não só no que diz respeito ao Squid ou qualquer outro *software* de *proxy*, tampouco referindo-se apenas aos sistemas operacionais, mas sim de todo *software* que executa tarefas exaustivas (no caso do *proxy*, procurar em *cache* ou na Internet o objeto solicitado pelo usuário e enviar ao solicitante).

Os testes com outros sistemas operacionais são os que deixaram maior dúvida durante o desenvolvimento do trabalho. Sendo assim, trabalhos futuros poderão aproveitar a ideia sugerida aqui para realizar novamente os testes quando as ferramentas de virtualização livres estejam mais estáveis em relação aos sistemas operacionais da família BSD ou ainda, quando houver a disponibilidade de máquinas que não utilizem virtualização, mas sim apenas o sistema BSD

instalado na máquina.

Por fim, esse trabalho serviu como base para um assunto que ainda hoje possui pouca exploração e documentação que é o refinamento do *software* e melhor aproveitamento de *hardware*. Esse trabalho mostra como é possível, com análises minuciosas, procurar caminhos para se obter o melhor desempenho com as ferramentas que se tem em mãos através da utilização adequada das mesmas.

REFERÊNCIAS

APESTEGUIA, Fernando. **Linux Performance Tuning**. 2006. Disponível em: <http://www.linuxforums.org/desktop/linux_performance_tuning.html>. Acesso em: 22 mar. 2009.

APPLE INC. **Open source overview**. 2007. Disponível em: <http://developer.apple.com/opensource/overview.html>. Acesso em: 18 mar. 2009.

BARROSO, Luiz A.; DEAN, Jeffrey; HÖLZLE, Urs. Web search for a planet: the Google cluster architecture. **IEEE Micro**, v. 23, n. 2, p. 22–28, mar./abr. 2003.

BELLARD, Fabrice. **QEMU Emulator User Documentation**. 2010. Disponível em: <http://manpages.ubuntu.com/manpages/lucid/man1/qemu.1.html>. Acesso em: 08 jun. 2010.

BORTZMEYER, Stéphany. **echoping Home Page**. 2008. Disponível em: <http://echoping.sourceforge.net/>. Acesso em: 14 jun. 2010.

BOUYER, Manuel. **DOM0 and DOMU work with only one cpu**. 2009. Disponível em: <http://mail-index.netbsd.org/netbsd-bugs/2009/12/11/msg014890.html>. Acesso em: 31 mai. 2010.

CHADD, Adrian. **Xen 8 & 9 DomU kernel panic with 2 or more virtual CPU**. 2010. Disponível em: <http://lists.freebsd.org/pipermail/freebsd-xen/2010-March/000472.html>. Acesso em: 31 mai. 2010.

CISCO SYSTEMS INC. **Internetworking technology handbook**. 4. ed. Indianapolis, IN, United States of America: Cisco Press, 2003.

CITRIX SYSTEMS. **Xen Kernel Git Development Tree**. 2010. Disponível em: <http://git.kernel.org/?p=linux/kernel/git/jeremy/xen.git;a=shortlog;h=xen/stable-2.6.32.x>. Acesso em: 10 mai. 2010.

CITRIX SYSTEMS. **XenParavirtOps: Bleeding edge work, including Xen dom0 support**. 2010. Disponível em: <http://wiki.xensource.com/xenwiki/XenParavirtOps>. Acesso em: 10 mai. 2010.

COMUNIDADE MUNIN. **Munin**. 2010. Disponível em: <http://munin-monitoring.org/>. Acesso em: 2 set. 2010.

FREE SOFTWARE FOUNDATION. **The free software definition**. 2007. Disponível em: <http://www.gnu.org/philosophy/free-sw.html>. Acesso em: 18 mar. 2009.

FREE SOFTWARE FOUNDATION. **What is free software and why is it so important for society?** 2008. Disponível em: <http://www.fsf.org/about/what-is-free-software>. Acesso em: 18 mar. 2009.

GRENNAN, Mark. **Firewall and proxy server howto**. 2000. Disponível em: <http://www.linux.org/docs/ldp/howto/Firewall-HOWTO-11.html#ss11.4>. Acesso em: 16 mar. 2009.

K12LINUX. **K12LINUX LTSP for Fedora**. 2009. Disponível em: <https://fedorahosted.org/k12linux/>. Acesso em: 19 mar. 2009.

LINUX KERNEL ORGANIZATION. **What is Linux?** 2008. Disponível em: <http://www.kernel.org/>. Acesso em: 18 mar. 2009.

NAMESYS. **Three reasons why ReiserFS is great for you**. 2007. Disponível em: <http://web.archive.org/web/20070711182813/http://www.namesys.com/X0reiserfs.html>. Acesso em: 2 set. 2010.

REDHAT. **Kernel Based Virtual Machine**. 2010. Disponível em: <http://www.linux-kvm.org/page/Main_Page>. Acesso em: 31 mai. 2010.

REDHAT. **Virtio**. 2010. Disponível em: <http://www.linux-kvm.org/page/Virtio>. Acesso em: 31 mai. 2010.

SECRETARIA DE ESTADO DA EDUCAÇÃO DO PARANÁ. **Paraná Digital - Saiba Mais**. 2010. Disponível em: <http://www.diaadiaeducacao.pr.gov.br/portals/portal/paranadigital/saiba_mais.php>. Acesso em: 1 set. 2010.

SGI. **XFS: A high-performance journaling filesystem**. 2010. Disponível em: <http://oss.sgi.com/projects/xfs/>. Acesso em: 2 set. 2010.

SHALLA SECURE SERVICES. **Welcome to SquidGuard**. 2009. Disponível em: <http://www.squidguard.org/>. Acesso em: 22 mar. 2009.

SMITH, Ken. **Generic kernel configuration file for FreeBSD/i386 PAE**. 2010. Disponível em: <http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/i386/conf/PAE?rev=1.32.2.1.4.1;content-type=text%2Fplain>. Acesso em: 05 jul. 2010.

SOFTWARE IN THE PUBLIC INTEREST, INC. **Debian Med: descrição do projeto**. 2009. Disponível em: <http://www.debian.org/devel/debian-med/>. Acesso em: 19 mar. 2009.

SYSTEMRESCUECD. **Welcome to SystemRescueCd**. 2009. Disponível em: <http://www.sysresccd.org/>. Acesso em: 19 mar. 2009.

THE CACTI GROUP. **About Cacti**. 2009. Disponível em: <http://www.cacti.net/>. Acesso em: 22 mar. 2009.

THE FREEBSD DOCUMENTATION PROJECT. **FreeBSD System Manager's Manual - SYSCTL(8)**. 2007. Disponível em: <http://www.freebsd.org/cgi/man.cgi?query=sysctl&sektion=8&manpath=FreeBSD+8.0-RELEASE>. Acesso em: 24 jun. 2010.

THE FREEBSD DOCUMENTATION PROJECT. **FreeBSD Handbook**. [S.l.]: FreeBSD Documentation Project, 2010. 305 p.

THE NETBSD PROJECT. **NetBSD/xen News**. 2009. Disponível em: <http://www.netbsd.org/ports/xen/>. Acesso em: 31 mai. 2010.

THOMAS, Keir; SICAM, Jaime. **Beginning Ubuntu Linux: from novice to professional**. 3. ed. United States of America: Apress, 2008.

VMWARE. **Understanding Full Virtualization, Paravirtualization and Hardware Assist**. 2007. Disponível em: <http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf>. Acesso em: 24 mai. 2010.

WESSELS, Duane. **Squid: the definitive guide**. Sebastopol, CA, United States of America: O'Reilly & Associates, Inc., 2004.

APÊNDICE A – CACTI_GRAPH_TEMPLATE_ECHOPING_-_GOOGLE.XML

```
<cacti>
<hash_00001994e42fa0c41bb550303e5a5db2597fe4>
<name>Echoping - Google</name>
<graph>
<t_title></t_title>
<title>|host_description| - Echoping - Google</title>
<t_image_format_id></t_image_format_id>
<image_format_id>1</image_format_id>
<t_height></t_height>
<height>120</height>
<t_width></t_width>
<width>500</width>
<t_slope_mode></t_slope_mode>
<slope_mode>on</slope_mode>
<t_auto_scale></t_auto_scale>
<auto_scale>on</auto_scale>
<t_auto_scale_opts></t_auto_scale_opts>
<auto_scale_opts>2</auto_scale_opts>
<t_auto_scale_log></t_auto_scale_log>
<auto_scale_log></auto_scale_log>
<t_scale_log_units></t_scale_log_units>
<scale_log_units></scale_log_units>
<t_auto_scale_rigid></t_auto_scale_rigid>
<auto_scale_rigid></auto_scale_rigid>
<t_auto_padding></t_auto_padding>
<auto_padding>on</auto_padding>
<t_export></t_export>
<export>on</export>
<t_upper_limit></t_upper_limit>
<upper_limit>100</upper_limit>
<t_lower_limit></t_lower_limit>
<lower_limit>0</lower_limit>
<t_base_value></t_base_value>
<base_value>1000</base_value>
<t_unit_value></t_unit_value>
<unit_value></unit_value>
<t_unit_exponent_value></t_unit_exponent_value>
<unit_exponent_value></unit_exponent_value>
<t_vertical_label></t_vertical_label>
<vertical_label>seconds</vertical_label>
</graph>
<items>
<hash_1000193d1972a802a9bba91805b295dd2b2d63>
```

```
<task_item_id>hash_08001937237a61d8d51718183da927ccd747cb</task_item_id>
<color_id>00FF00</color_id>
<alpha>FF</alpha>
<graph_type_id>7</graph_type_id>
<consolidation_function_id>1</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Average Response Time</text_format>
<hard_return></hard_return>
<sequence>1</sequence>
</hash_1000193d1972a802a9bba91805b295dd2b2d63>
<hash_1000190cd3041c0a53948f70b8b49c1328a06b>
<task_item_id>hash_08001937237a61d8d51718183da927ccd747cb</task_item_id>
<color_id>0</color_id>
<alpha>FF</alpha>
<graph_type_id>9</graph_type_id>
<consolidation_function_id>4</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Cur</text_format>
<hard_return></hard_return>
<sequence>2</sequence>
</hash_1000190cd3041c0a53948f70b8b49c1328a06b>
<hash_100019b7e99e4c620615b01501360c95c33b02>
<task_item_id>hash_08001937237a61d8d51718183da927ccd747cb</task_item_id>
<color_id>0</color_id>
<alpha>FF</alpha>
<graph_type_id>9</graph_type_id>
<consolidation_function_id>2</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Min</text_format>
<hard_return></hard_return>
<sequence>3</sequence>
</hash_100019b7e99e4c620615b01501360c95c33b02>
<hash_100019e32342525e1ba130ec3fbfee9181e6e2>
<task_item_id>hash_08001937237a61d8d51718183da927ccd747cb</task_item_id>
<color_id>0</color_id>
<alpha>FF</alpha>
<graph_type_id>9</graph_type_id>
<consolidation_function_id>3</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Max</text_format>
<hard_return></hard_return>
<sequence>4</sequence>
</hash_100019e32342525e1ba130ec3fbfee9181e6e2>
<hash_100019bf56949ed7937277ee017c7337747896>
<task_item_id>hash_08001959eb7d1a8bb5e2b8f3b935c09899a849</task_item_id>
<color_id>FF0000</color_id>
```

```
<alpha>FF</alpha>
<graph_type_id>4</graph_type_id>
<consolidation_function_id>1</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Maximum Response Time</text_format>
<hard_return></hard_return>
<sequence>5</sequence>
</hash_100019bf56949ed7937277ee017c7337747896>
<hash_100019aa56c108be36d3662352ddfd2a962488>
<task_item_id>hash_08001959eb7d1a8bb5e2b8f3b935c09899a849</task_item_id>
<color_id>0</color_id>
<alpha>FF</alpha>
<graph_type_id>9</graph_type_id>
<consolidation_function_id>4</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Cur</text_format>
<hard_return></hard_return>
<sequence>6</sequence>
</hash_100019aa56c108be36d3662352ddfd2a962488>
<hash_100019544b188d14c48eb8f6ecc4840799d924>
<task_item_id>hash_08001959eb7d1a8bb5e2b8f3b935c09899a849</task_item_id>
<color_id>0</color_id>
<alpha>FF</alpha>
<graph_type_id>9</graph_type_id>
<consolidation_function_id>2</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Min</text_format>
<hard_return></hard_return>
<sequence>7</sequence>
</hash_100019544b188d14c48eb8f6ecc4840799d924>
<hash_1000195159a3cc392b914614fba80dc44f55a6>
<task_item_id>hash_08001959eb7d1a8bb5e2b8f3b935c09899a849</task_item_id>
<color_id>0</color_id>
<alpha>FF</alpha>
<graph_type_id>9</graph_type_id>
<consolidation_function_id>3</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Max</text_format>
<hard_return></hard_return>
<sequence>8</sequence>
</hash_1000195159a3cc392b914614fba80dc44f55a6>
<hash_1000193fd6701320d0d946abfd39b4fc6e423f>
<task_item_id>hash_0800193ba5c552ca86f052fd9b3001e9ea3040</task_item_id>
<color_id>4444FF</color_id>
<alpha>FF</alpha>
<graph_type_id>5</graph_type_id>
```

```
<consolidation_function_id>1</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Minimum Response Time</text_format>
<hard_return></hard_return>
<sequence>9</sequence>
</hash_1000193fd6701320d0d946abfd39b4fc6e423f>
<hash_1000198072ea46de74c3b44ae29a986df50e35>
<task_item_id>hash_0800193ba5c552ca86f052fd9b3001e9ea3040</task_item_id>
<color_id>0</color_id>
<alpha>FF</alpha>
<graph_type_id>9</graph_type_id>
<consolidation_function_id>4</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Cur</text_format>
<hard_return></hard_return>
<sequence>10</sequence>
</hash_1000198072ea46de74c3b44ae29a986df50e35>
<hash_100019e17c002e3be49e42cde89187139f7679>
<task_item_id>hash_0800193ba5c552ca86f052fd9b3001e9ea3040</task_item_id>
<color_id>0</color_id>
<alpha>FF</alpha>
<graph_type_id>9</graph_type_id>
<consolidation_function_id>2</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Min</text_format>
<hard_return></hard_return>
<sequence>11</sequence>
</hash_100019e17c002e3be49e42cde89187139f7679>
<hash_1000196794e5ca4cd996dcc1db1a1dc91cc807>
<task_item_id>hash_0800193ba5c552ca86f052fd9b3001e9ea3040</task_item_id>
<color_id>0</color_id>
<alpha>FF</alpha>
<graph_type_id>9</graph_type_id>
<consolidation_function_id>3</consolidation_function_id>
<cdef_id>0</cdef_id>
<value></value>
<gprint_id>hash_060019e9c43831e54eca8069317a2ce8c6f751</gprint_id>
<text_format>Max</text_format>
<hard_return></hard_return>
<sequence>12</sequence>
</hash_1000196794e5ca4cd996dcc1db1a1dc91cc807>
</items>
<inputs>
<hash_0900199734464ce12fd774d3f6d20b84850ae8>
<name>Data Source [avg]</name>
<description></description>
<column_name>task_item_id</column_name>
<items>hash_0000193d1972a802a9bba91805b295dd2b2d63|
```

```
    hash_000019b7e99e4c620615b01501360c95c33b02|
    hash_0000190cd3041c0a53948f70b8b49c1328a06b|
    hash_000019e32342525e1ba130ec3fbfee9181e6e2</items>
</hash_0900199734464ce12fd774d3f6d20b84850ae8>
<hash_0900193d74cdde409e0cbf19d96c58d8fad065>
<name>Data Source [max]</name>
<description></description>
<column_name>task_item_id</column_name>
<items>hash_000019bf56949ed7937277ee017c7337747896|
    hash_000019544b188d14c48eb8f6ecc4840799d924|
    hash_000019aa56c108be36d3662352ddfd2a962488|
    hash_0000195159a3cc392b914614fba80dc44f55a6</items>
</hash_0900193d74cdde409e0cbf19d96c58d8fad065>
<hash_0900198e7f4df61c3bbb633f4ee66e25f83e96>
<name>Data Source [min]</name>
<description></description>
<column_name>task_item_id</column_name>
<items>hash_0000193fd6701320d0d946abfd39b4fc6e423f|
    hash_000019e17c002e3be49e42cde89187139f7679|
    hash_0000198072ea46de74c3b44ae29a986df50e35|
    hash_0000196794e5ca4cd996dcc1db1a1dc91cc807</items>
</hash_0900198e7f4df61c3bbb633f4ee66e25f83e96>
</inputs>
</hash_00001994e42fa0c41bb550303e5a5db2597fe4>
<hash_010019154f12a0b059bef479ba7dc4ac86b6dc>
<name>Echoping - Google</name>
<ds>
<t_name></t_name>
<name>|host_description| - Echoping - Google</name>
<data_input_id>hash_0300198469b732c74e4d9c3884080baa51b9cd</data_input_id>
<t_rra_id></t_rra_id>
<t_rrd_step></t_rrd_step>
<rrd_step>300</rrd_step>
<t_active></t_active>
<active>on</active>
<rra_items>hash_150019c21df5178e5c955013591239eb0afd46|
    hash_1500190d9c0af8b8acdc7807943937b3208e29|
    hash_1500196fc2d038fb42950138b0ce3e9874cc60|
    hash_150019e36f3adb9f152adfa5dc50fd2b23337e|
    hash_150019283ea2bf1634d92ce081ec82a634f513</rra_items>
</ds>
<items>
<hash_0800193ba5c552ca86f052fd9b3001e9ea3040>
<t_data_source_name></t_data_source_name>
<data_source_name>min</data_source_name>
<t_rrd_minimum></t_rrd_minimum>
<rrd_minimum>0</rrd_minimum>
<t_rrd_maximum></t_rrd_maximum>
<rrd_maximum>100</rrd_maximum>
<t_data_source_type_id></t_data_source_type_id>
<data_source_type_id>1</data_source_type_id>
<t_rrd_heartbeat></t_rrd_heartbeat>
<rrd_heartbeat>600</rrd_heartbeat>
<t_data_input_field_id></t_data_input_field_id>
```

```
<data_input_field_id>hash_07001968119cbe391786d31eea72d0e500ae37</data_input_field_id>
</hash_0800193ba5c552ca86f052fd9b3001e9ea3040>
<hash_08001959eb7d1a8bb5e2b8f3b935c09899a849>
<t_data_source_name></t_data_source_name>
<data_source_name>max</data_source_name>
<t_rrd_minimum></t_rrd_minimum>
<rrd_minimum>0</rrd_minimum>
<t_rrd_maximum></t_rrd_maximum>
<rrd_maximum>100</rrd_maximum>
<t_data_source_type_id></t_data_source_type_id>
<data_source_type_id>1</data_source_type_id>
<t_rrd_heartbeat></t_rrd_heartbeat>
<rrd_heartbeat>600</rrd_heartbeat>
<t_data_input_field_id></t_data_input_field_id>
<data_input_field_id>hash_0700193179837b7cc9d031a1e4d30a75b66c89</data_input_field_id>
</hash_08001959eb7d1a8bb5e2b8f3b935c09899a849>
<hash_08001937237a61d8d51718183da927ccd747cb>
<t_data_source_name></t_data_source_name>
<data_source_name>avg</data_source_name>
<t_rrd_minimum></t_rrd_minimum>
<rrd_minimum>0</rrd_minimum>
<t_rrd_maximum></t_rrd_maximum>
<rrd_maximum>100</rrd_maximum>
<t_data_source_type_id></t_data_source_type_id>
<data_source_type_id>1</data_source_type_id>
<t_rrd_heartbeat></t_rrd_heartbeat>
<rrd_heartbeat>600</rrd_heartbeat>
<t_data_input_field_id></t_data_input_field_id>
<data_input_field_id>hash_0700191d7e5f10d3cb98eb57f176dac14af50c</data_input_field_id>
</hash_08001937237a61d8d51718183da927ccd747cb>
</items>
<data>
<item_000>
<data_input_field_id>hash_070019c91bf8a10dbc4eb93718caa4e0bd824d</data_input_field_id>
<t_value></t_value>
<value>8080</value>
</item_000>
<item_001>
<data_input_field_id>hash_0700194245da73524d504536019cb7a69e34c3</data_input_field_id>
<t_value></t_value>
<value></value>
</item_001>
<item_002>
<data_input_field_id>hash_070019f77d8d3ece4d5a19cb6cb499cac4d43e</data_input_field_id>
<t_value></t_value>
<value>http://www.google.com.br</value>
</item_002>
<item_003>
<data_input_field_id>hash_070019219db38ef42a1a7f3ef1030e01240491</data_input_field_id>
<t_value></t_value>
<value>3</value>
</item_003>
</data>
</hash_010019154f12a0b059bef479ba7dc4ac86b6dc>
```

```
<hash_0300198469b732c74e4d9c3884080baa51b9cd>
<name>Echoping - Proxy</name>
<type_id>1</type_id>
<input_string>perl &lt;path_cacti&gt;/scripts/echoping.pl &lt;numberoftests&gt; &lt;site&gt;
    &lt;proxy&gt; &lt;proxyport&gt;</input_string>
<fields>
<hash_07001968119cbe391786d31eea72d0e500ae37>
<name>Minimum Request Time</name>
<update_rra>on</update_rra>
<regexp_match></regexp_match>
<allow_nulls></allow_nulls>
<type_code></type_code>
<input_output>out</input_output>
<data_name>Min</data_name>
</hash_07001968119cbe391786d31eea72d0e500ae37>
<hash_0700193179837b7cc9d031a1e4d30a75b66c89>
<name>Maximum Request Time</name>
<update_rra>on</update_rra>
<regexp_match></regexp_match>
<allow_nulls></allow_nulls>
<type_code></type_code>
<input_output>out</input_output>
<data_name>Max</data_name>
</hash_0700193179837b7cc9d031a1e4d30a75b66c89>
<hash_0700191d7e5f10d3cb98eb57f176dac14af50c>
<name>Average Request Time</name>
<update_rra>on</update_rra>
<regexp_match></regexp_match>
<allow_nulls></allow_nulls>
<type_code></type_code>
<input_output>out</input_output>
<data_name>Average</data_name>
</hash_0700191d7e5f10d3cb98eb57f176dac14af50c>
<hash_070019f77d8d3ece4d5a19cb6cb499cac4d43e>
<name>Site to test</name>
<update_rra></update_rra>
<regexp_match></regexp_match>
<allow_nulls></allow_nulls>
<type_code></type_code>
<input_output>in</input_output>
<data_name>site</data_name>
</hash_070019f77d8d3ece4d5a19cb6cb499cac4d43e>
<hash_0700194245da73524d504536019cb7a69e34c3>
<name>Proxy address</name>
<update_rra></update_rra>
<regexp_match></regexp_match>
<allow_nulls></allow_nulls>
<type_code>hostname</type_code>
<input_output>in</input_output>
<data_name>proxy</data_name>
</hash_0700194245da73524d504536019cb7a69e34c3>
<hash_070019c91bf8a10dbc4eb93718caa4e0bd824d>
<name>Proxy port</name>
<update_rra></update_rra>
```

```
<regexp_match></regexp_match>
<allow_nulls></allow_nulls>
<type_code></type_code>
<input_output>in</input_output>
<data_name>proxyport</data_name>
</hash_070019c91bf8a10dbc4eb93718caa4e0bd824d>
<hash_070019219db38ef42a1a7f3ef1030e01240491>
<name>Number of tests to do</name>
<update_rra></update_rra>
<regexp_match></regexp_match>
<allow_nulls></allow_nulls>
<type_code></type_code>
<input_output>in</input_output>
<data_name>numberoftests</data_name>
</hash_070019219db38ef42a1a7f3ef1030e01240491>
</fields>
</hash_0300198469b732c74e4d9c3884080baa51b9cd>
<hash_150019c21df5178e5c955013591239eb0afd46>
<name>Daily (5 Minute Average)</name>
<x_files_factor>0.5</x_files_factor>
<steps>1</steps>
<rows>600</rows>
<timespan>86400</timespan>
<cf_items>1|3</cf_items>
</hash_150019c21df5178e5c955013591239eb0afd46>
<hash_1500190d9c0af8b8acdc7807943937b3208e29>
<name>Weekly (30 Minute Average)</name>
<x_files_factor>0.5</x_files_factor>
<steps>6</steps>
<rows>700</rows>
<timespan>604800</timespan>
<cf_items>1|3</cf_items>
</hash_1500190d9c0af8b8acdc7807943937b3208e29>
<hash_1500196fc2d038fb42950138b0ce3e9874cc60>
<name>Monthly (2 Hour Average)</name>
<x_files_factor>0.5</x_files_factor>
<steps>24</steps>
<rows>775</rows>
<timespan>2678400</timespan>
<cf_items>1|3</cf_items>
</hash_1500196fc2d038fb42950138b0ce3e9874cc60>
<hash_150019e36f3adb9f152adfa5dc50fd2b23337e>
<name>Yearly (1 Day Average)</name>
<x_files_factor>0.5</x_files_factor>
<steps>288</steps>
<rows>797</rows>
<timespan>33053184</timespan>
<cf_items>1|3</cf_items>
</hash_150019e36f3adb9f152adfa5dc50fd2b23337e>
<hash_150019283ea2bf1634d92ce081ec82a634f513>
<name>Hourly (1 Minute Average)</name>
<x_files_factor>0.5</x_files_factor>
<steps>1</steps>
<rows>500</rows>
```

```
<timespan>14400</timespan>
<cf_items>1|3</cf_items>
</hash_150019283ea2bf1634d92ce081ec82a634f513>
<hash_060019e9c43831e54eca8069317a2ce8c6f751>
<name>Normal</name>
<gprint_text>%8.2lf %s</gprint_text>
</hash_060019e9c43831e54eca8069317a2ce8c6f751>
</cacti>
```

APÊNDICE B – ECHOPING.PL

```
#!/usr/bin/perl -w
#
# It is my first (dirty) perl script. GPL2.
# It use echoping to test responce time of servers for http, https, smtp and echo protocols
# 07/08/2002 version 0.2
# Please e-mail me optimisations...
# Sébastien Desse : sdesse@euresys.fr
#
# Customized for Cacti Graph Template/Data Input Method
# Echoping - Proxy created by
# Peter Andreas Entschev <peter@entschev.com>
# Last changed Jun 21 2010
#
# syntax : echoping.pl <number of tests> <url> <proxy> <port>
# number of tests > 0
# url             = @IP | FQDN
# proxy           = @IP | FQDN
# port            = Usually 3128 or 8080

@ARGV == 4 or die "Syntax : echoping.pl <number of tests> <url> <proxy> <port>\n";


$NumberOfTests = $ARGV[0];
$Url = $ARGV[1];
$Proxy = $ARGV[2];
$Port = $ARGV[3];

($Min1,$Max1,$Average1) = (`echoping -w 0.000001 -n $NumberOfTests -h $Url $Proxy:$Port`
    =~ m/Min.*(\d+\.\d+).*\nMax.*(\d+\.\d+).*\nAverage.*(\d+\.\d+)/);
($Min2,$Max2,$Average2) = (`echoping -w 0.000001 -n $NumberOfTests -h $Url $Proxy:$Port`
    =~ m/Min.*(\d+\.\d+).*\nMax.*(\d+\.\d+).*\nAverage.*(\d+\.\d+)/);

if ($Max1 <= $Max2) {
    print "Min:$Min1 Max:$Max1 Average:$Average1";
} else {
    print "Min:$Min2 Max:$Max2 Average:$Average2";
}
```