Lorenz Henrique Helleis

Apresentação do estudo de caso de uma rede de grande porte usando PF com OpenBSD para segurança de perímetro de rede com alta disponibilidade

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. Msc. Sandro Melo

Co-Orientador
Prof. Esp. Hermano Pereira

Lavras
Minas Gerais - Brasil
2008

Lorenz Henrique Helleis

Apresentação do estudo de caso de uma rede de grande porte usando PF com OpenBSD para segurança de perímetro de rede com alta disponibilidade

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em *28/08/2008*

Prof. Msc. Denilson Vedoveto Martins

Prof. Esp. Heitor Augustus Xavier Costa

Prof. Msc. Sandro Melo
(Orientador)

Prof. Esp. Hermano Pereira
(Co-Orientador)

Lavras
Minas Gerais - Brasil

Dedico este trabalho primeiramente a Deus, pois sem Ele, nada seria possível e não estaríamos aqui reunidos, desfrutando, juntos, destes momentos que nos são tão importantes. Dedico também a minha mãe, a qual sempre me apoiou em tudo o que eu fiz na minha vida. Em especial, ao meu grande amigo e professor Hermano Pereira, por sua confiança e credibilidade em minha pessoa, o qual sempre esteve me ajudando a crescer profissionalmente e sempre me desafiando a ir cada vez mais longe.

Agradecimentos

Agradeço a Deus por sempre estar comigo em todas as circunstâncias. Sempre esta me ensinando a ser uma pessoa melhor.

Agradeço a minha mãe por ter me apoiado em todos meus estudos, muitas vezes se sacrificando financeiramente pra me dar o melhor possível.

Agradeço ao meu amigo Hermano Pereira pelo grande incentivo em minha vida profissional e pela grande amizade que tem demonstrado.

Agradeço ao Professor Sandro Melo por compartilhar seu conhecimento.

Agradeço aos amigos e colegas da empresa onde trabalho, os quais me incentivaram e me apoiaram no projeto realizado.

Agradeço a todos meus amigos que são uma bênção em minha vida e que sempre estiveram comigo, sendo momentos felizes ou momentos tristes.

Sumário

Lista de Figuras

Lista de Abreviaturas

ARP *Address Resolution Protocol*

CARP *Common Address Redundancy Protocol*

DMZ *Demilitarized Zone*

GUI *Graphical User Interface*

HSRP *Hot Standby Routing Protocol*

IETF *Internet Engineering Task Force*

IMAP *Internet Message Access Protocol*

IP *Internet Protocol*

IPS *Intrusion Prevention System*

LAN *Local Area Network*

MAC *Media Access Control*

NAT *Network Address Translation*

PF *Packet Filter*

TCP *Transmission Control Protocol*

VPN *Virtual Private Network*

VRRP *Virtual Router Redundancy Protocol*

XML *EXtensible Markup Language*

Resumo

Receber notificações de usuários insatisfeitos; administradores reclamando incessantemente da indisponibilidade de serviços; e, ainda mais, ser responsabilizado por arquivos de cópia de segurança corrompidos. Este é um cenário comum na vida de um administrador de rede que não projetou o seu *firewall* como deveria. Portanto, neste trabalho, será apresentada uma solução de *firewall* em *software* livre que provê alta disponibilidade em caso de falhas,

Palavras-chave: Firewall, Carp, Pfsync, OpenBSD, disponibilidade.

Capítulo 1

Introdução

Uma empresa que investe no crescimento da economia digital e disponibiliza serviços e vendas através da Internet requer disponibilidade contínua do que está oferencedo. Se a loja virtual não está funcionando, potenciais ganhos são anulados. Para evitar que a empresa leve prejuízo, seus serviços deverão estar sobre uma estrutura de rede redundante que provê alta disponibilidade.

Um dos equipamentos que não pode ficar de fora de um projeto de rede é o *firewall*. Além da função do *firewall* em manter conexões ativas e, ao mesmo tempo, proteger contra ataques; é adicionada uma nova responsabilidade em manter as sessões ativas e replicadas em um ambiente que provê alta disponibilidade.

Para que os *firewalls* possam se comunicar e manter sessões ativas, torna-se necessária a implementação de protocolos de redundância. Alguns protocolos são comerciais, como é o exemplo do *Hot Standby Routing Protocol* (HSRP) da Cisco e *Virtual Router Redundancy Protocol* (VRRP). O VRRP é alvo de críticas em segurança e possui uma série de discordâncias com o *Internet Engineering Task Force* (IETF) sem contar os conflitos com a patente do HSRP. Enquanto se discutiam problemas, a equipe do OpenBSD decidiu criar um novo protocolo de redundância livre de patentes, o que resultou no protocolo *Common Address Redundancy Protocol* (CARP).

1.1 Definição dos objetivos

Esta monografia tem como objetivo propor uma implementação para um ambiente real de segurança de perímetro com alta disponibilidade para *firewall*, utilizando PFSync e CARP sobre o sistema operacional OpenBSD. Esta solução tem como principal finalidade manter os serviços disponibilizados pela empresa o máximo de tempo possível, onde, para isso, utilizam-se técnicas de compartilhamento de sessões e redundância de *hardware*.

1.2 Motivação

A falha de *links* em uma rede *Internet Protocol* (IP) ocasiona a atualização das tabelas de roteamento para refletir a mudança da topologia. Essas mudanças frequentemente afetam a rede, causando instabilidade e prejudicando, por algum tempo, o desempenho dos equipamentos. Isso ocorre também na atualização da tabela arp dos roteadores caso seja necessário fazer uma mudança de um equipamento primário para um secundário.

A incorporação de redundância em roteadores e *firewalls* reduz a probabilidade de falha do *link*. A remoção desses tipos de problemas, aumenta a estabilidade e o desempenho da rede, existindo benefício aos usuários.

O ambiente funcionava com a utilização de *firewalls* utilizando o sistema operacional Debian juntamente com iptables. Esse ambiente tinha por objetivo colocar um certo nível de segurança no acesso a servidores em uma *Demilitarized Zone* (DMZ). Existiam dois *firewalls* fazendo redundância, eles eram balanceados por dois *switches* de balanceamento e toda a estrutura de regras estava sendo gerenciada com a ferramenta fwbuilder.

A motivação principal é a busca de um ambiente com alta disponibilidade. Além disso, é necessário liberar os *switches* de balanceamento dos *firewalls*; por isso, a redundância não deveria mais ser feita pelo switch de balanceamento, que seria destinado a outros fins.

1.3 Estado da arte

Atualmente, encontra-se como proposta de protocolos para soluções de alta disponibilidade em *firewall* a implementação do VRRP ou CARP. Ambos têm por

objetivo garantir a redundância de *hosts*[1], fazendo com que vários *hosts* sejam responsáveis por um único endereço em uma rede (BOTELHO, 2006).

O VRRP, protocolo padrão do IETF, é patenteado pela CISCO[2]. Este é protegido por patente e direitos autorais. Por esse motivo, a comunidade *open source* criou o padrão livre CARP, um protocolo FOSS[3], nativo do OpenBSD (BOTELHO, 2006).

Quando se utiliza a tecnologia VRRP, cria-se um endereço virtual vinculado a uma interface de rede; assim, os *hosts* pertencentes ao mesmo grupo compartilham somente um mesmo endereço IP. A desvantagem deste protocolo é a atualização de todo o *cache Address Resolution Protocol* (ARP) sempre que um *host* mudar de estado [RFC 826] (BOTELHO, 2006).

Ao contrário do VRRP, o CARP trabalha com uma interface virtual, nomeada carpN, onde N é o número da interface vinculada a uma interface real. A interface virtual tem um endereço *Media Access Control* (MAC) Virtual, bem como os *hosts* que estão compartilham o mesmo endereço. Assim, quando um *host* muda de estado, o *cache* não precisa ser atualizado, pois o novo *host* mestre irá assumir o endereço de IP e o endereço MAC. A única exigência para a utilização do CARP é a necessidade de, no mínimo, 3 endereços lógicos na mesma rede (BOTELHO, 2006).

O projeto OpenBSD incluiu o *Packet Filter* (PF) a partir da versão 3.0, a qual pode ser usada para criar *statefull firewalls*. Mas, usando somente o CARP para garantir *firewall* redundante, existe um problema. Quando o *firewall* primário pára de funcionar por algum motivo, os estados das conexões são perdidos e as conexões ativas não serão reconhecidas pelo firewall secundário. A solução para esse tipo de problema é o pfsync, o qual é utilizado para manter sincronizada a tabela de estados entre os *firewalls* (ATTEBURY; RAMAMURTHY, 2008).

Algumas tecnologias relacionadas:

- IPTABLES: é um *firewall* em nível de pacotes e funciona baseado no endereço/porta de origem/destino do pacote, prioridade, etc. Ele funciona através da comparação de regras para saber se um pacote tem ou não permissão para passar. Em *firewalls* mais restritivos, o pacote é bloqueado e registrado para

[1] *Hosts* é todo e qualquer equipamento conectado a uma rede e que possua um endereço IP (BOTELHO, 2006).

[2] http://www.cisco.com

[3] (Free and Open-Source Software) Software Livre e código aberto

que o administrador do sistema tenha conhecimento sobre o que está acontecendo em seu sistema (SILVA, 2006).

Ele também pode ser usado para modificar e monitorar o tráfego da rede, fazer *Network Address Translation* (NAT) (*masquerading, source nat, destination nat*), redirecionar pacotes, marcar pacotes, modificar a prioridade de pacotes que chegam/saem do sistema, contar bytes, dividir tráfego entre máquinas, criar proteções *anti-spoofing*, contra *syn flood*, DoS, etc. O tráfego vindo de máquinas desconhecidas da rede pode também ser bloqueado/registrado através do uso de simples regras. As possibilidades oferecidas pelos recursos de filtragem *iptables* garantem flexibilidade na manipulação das regras de acesso ao sistema, precisando apenas conhecer quais interfaces o sistema possui, o que deseja bloquear, o que tem acesso garantido e quais serviços devem estar acessíveis para cada rede, e iniciar a construção do *firewall* (SILVA, 2006).

O iptables é um *firewall* com estado, ou seja, um *firewall stateful*. Os anteriores eram *stateless*. O modo de filtragem '*Stateless*' tende a tratar cada pacote roteado pelo *firewall* como pacotes individuais e podem ser usados para obter melhor desempenho em determinadas situações onde existem regras de nível de rede bem simples.(THOMAZ, 2005).

Enfim, o *iptables* é um *firewall* que agradará pessoas que desejam uma segurança básica no sistema e quando administradores de grandes redes que querem ter controle minucioso sobre o tráfego que passa entre suas interfaces de rede (controlando o que pode passar de uma rede a outra) (SILVA, 2006).

- IPFW é um filtro de pacotes nativo no *kernel*[4] de alguns sistemas operacionais, entre eles o Linux e o FreeBSD (sistemas abertos bem conhecidos). Um administrador pode elevar o nível de proteção de sua rede interna caso o IPFW seja instalado no seu *gateway*.Além disso um usuário preocupado com segurança pode utilizá-lo para proteger o computador de conexões indesejáveis. Por ser parte de pacotes de *software* de livre distribuição, não há ônus financeiro por utilizar este filtro (FORSTER, 1998).

  Por ser um filtro de pacotes, o IPFW só pode aceitar ou rejeitar um determinado pacote baseando-se nos dados a que tem acesso. O IPFW não pode, por exemplo, perceber a passagem de um applet maligno em Java, ou detectar que uma conexão *Internet Message Access Protocol* (IMAP) está normal

[4] *Kernel* é o núcleo de um sistema operacional. Ele representa a camada mais baixa de interface com o *hardware* (FORSTER, 1998).

enquanto outra está sendo feita para explorar um *bug* do servidor. Ele analisa os dados presentes no cabeçalho dos pacotes, e mas não a sua área de dados (FORSTER, 1998).

Tendo feito a análise dos dados do pacote, o IPFW decide, baseado nas regras inseridas pelo administrador, se o pacote será aceito, rejeitado, aceito e "logado"ou rejeitado e "logado". Se a opção de log estiver ativada, uma mensagem será enviada ao syslogd, registrando os dados do pacote (FORSTER, 1998).

Um detalhe importante é ao contrário do *Transmission Control Protocol* (TCP) *Wrapper*, que registra apenas conexões iniciadas corretamente. O IPFW registra qualquer tentativa, mesmo que o TCP *3-Way Handshake* não tenha sido completado. Desta forma, mesmo os *Stealth Scans* podem ser detectados. Outra vantagem é o IPFW, estando corretamente configurado, registrar pacotes de qualquer protocolo direcionado a qualquer porta e apenas a serviços controlados pelo inetd. A desvantagem é ele fornecer informações sobre os pacotes logados de forma independente (FORSTER, 1998).

- IPFilter é uma aplicação *open source* e tem sido portada para os sistemas operacionais FreeBSD, NetBSD, OpenBSD, SunOS, HP/UX e Solaris (PROJECT, 2007).

  O IPFilter utilizava regras do tipo *stateless*, mas, atualmente, utiliza regras do tipo *stateful*, a qual modernizou a lógica de processamento de regras (PROJECT, 2007).

- PFSense é um dos mais conhecidos e provavelmente mais rico em recursos entre os sistemas para *appliance* pré-configurado. Ele é amistoso de fácil uso por interface Web sendo uma versão customizada do FreeBSD que oferece inúmeros recursos e focado para ambiente de roteamento e *firewalling* e segurança de rede, com excelente solução para *Virtual Private Network* (VPN), entre outros diversos recursos. O PFSense é um projeto decendente do conhecido m0n0wall. Ao lado do FreeNAS, são os principais projetos derivados do m0n0wall(FUGBR, 2002).

  O pfSense chegou em um nível de desenvolvimento que não há muito mais o que fazer, o que integrar, a não ser manter a atualização e estabilidade, além de melhoria dos recursos atuais. Talvez o único recurso que o pfSense não integre - por opção dos criadores do Projeto - é um *Intrusion Prevention System* (IPS) (FUGBR, 2002).

- MonoWall é baseado em uma versão do bare-bones de FreeBSD, junto com um *web server*, de PHP e de algumas outras utilidades. A configuração

do sistema inteiro é armazenado em uma única linha de texto de *EXtensible Markup Language* (XML) para manter a configuração transparente(KASPER, 2007).

O m0n0wall é um *firewall open source* desenvolvido por Manuel Kasper, ele oferece muitas funções encontradas em *firewalls* comerciais como Check Point e Cisco Pix, incluindo filtragem de pacote *stateful*. Pode-se usar o m0n0wall como *gateway* VPN, então é possível acessar uma *Local Area Network* (LAN) através da *Internet*(VINGAARD, 2005).

O m0n0wall tem uma boa interface Web para configuração do *firewall*. Os valores podem ser gravados em um único arquivo XML. A configuração pode ser salva em algum *drive* para que possa ser utilizado para instalação de regras em outro *firewall*(VINGAARD, 2005).

1.4 Estrutura dos capítulos

O capítulo 2 objetiva fundamentar teoricamente sobre as tecnologias envolvidas com essa proposta, descrevendo conceitos das tecnologias.

O capítulo 3 descreve o estudo de caso utilizado para a criação desse trabalho, descrevendo o cenário em que estava, mostrando o objetivo da mudança e como a rede ficou estruturada após a migração.

O capítulo 4 se refere a implementação do novo cenário, mostrando como foi feita toda a configuração do novo ambiente.

O capítulo 5 descreve algumas ferramentas para plano de capacidade, essas ferramentas tem como objetivo se manter informado do funcionamento do ambiente e para descoberta de possíveis alterações na rede.

O capítulo 6 apresenta resultados obtidos com a implementação desta proposta, demonstrando os problemas ocorridos e algum tuning que foi necessário ser feito.

O capítulo 7 é a conclusão final deste trabalho, descrevendo os resultados obtidos e a proposta de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo tem por objetivo apresentar as ferramentas utilizadas no capítulo 3, com fundamentação teória em OpenBSD, CARP, PFsync, PF e Fwbuilder.

2.1 OpenBSD

O projeto OpenBSD produz um sistema operacional livre, multi-plataforma, do tipo UNIX baseado no 4.4BSD. O OpenBSD suporta emulação binária da maioria dos programas do SVR4 (Solaris), FreeBSD, Linux, BSD/OS, SunOS e HP-UX (OPENBSD, 2006a).

O OpenBSD é desenvolvido por voluntários. Os fundos para o desenvolvimento do projeto são adquiridos a partir da venda de CDs e Camisetas e por intermédio de doações. Organizações e Indivíduos realizam doações e garantem que o OpenBSD continuará a existir e permanecerá livre para a utilização de todos e para todos os fins (OPENBSD, 2006a).

2.2 CARP

O CARP é o substituto para o VRRP, é um protocolo nativo do OpenBSD e foi portado para o FreeBSD (OLIVEIRA, 2006). O VRRP é um padrão do IETF[1] e é

[1]Comunidade internacional cuja missão é identificar problemas relacionados a utilização da Internet, além de propor soluções e a padronização das tecnologias e protocolos envolvidos.

patenteado pela CISCO. Com isso, o Projeto OpenBSD criou um padrão livre, o CARP (OLIVEIRA, 2006).

O objetivo do CARP é o mesmo do VRRP, garantir a redundância de *hosts*, fazendo com que vários *hosts* sejam responsáveis por um único endereço em uma rede, caso um falhe, o outro assume. Porém, o CARP age de forma diferente do VRRP, o VRRP cria um endereço virtual vinculado a uma interface de rede, como o criado com o parâmetro "ifconfig_interface_alias0"do rc.conf. Assim os *hosts* pertencentes ao mesmo grupo compartilha somente o mesmo IP e cada vez que um host muda de estado o *cache* ARP da rede deve ser atualizado (OLIVEIRA, 2006).

A figura 2.1 mostra a infra-estrutura da rede utilizando o carp, a qual mostra dois *firewalls* compartilhando apenas um IP virtual e um MAC virtual e quanto um está assumindo o tráfego, o outro fica de *backup*.

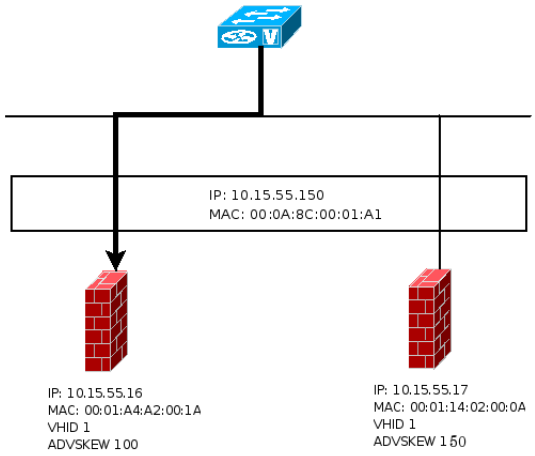

Figura 2.1: Infra-estrutura CARP em estado normal

Caso a máquina master falhe, a máquina slave irá assumir o tráfego. A figura 2.2 mostra como ficará o acesso caso a máquina master falhe.

Caso a máquina *master* volte a funcionar, ela será *slave*. Assim a eleição para máquina irá começar e ela voltará a ser a máquina *master* por possuir o menor advskew.

Para o grupo de redundância, é atribuído um endereço IP compartilhado entre os membros do grupo. Dentro do grupo, um *host* é designado o "*master*"e o restante sendo "*backup*". O host master é o que atualmente mantém o IP compartilhado sendo o responsável por responder a qualquer tráfego ou requisições ARP

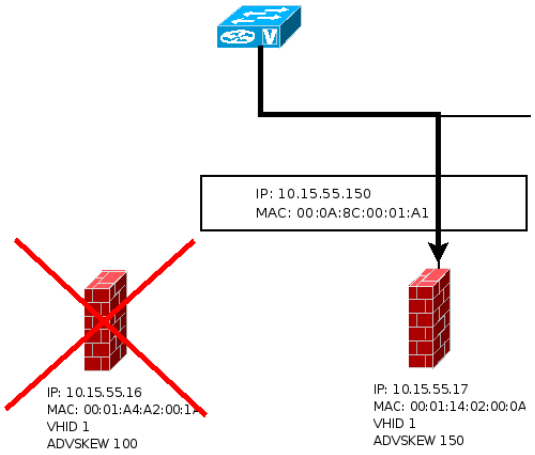

Figura 2.2: Infra-estrutura CARP quando o master falha

direcionadas para o IP do grupo. Cada *host* pode pertencer mais de um grupo de redundância por vez (BOTELHO, 2006).

O *host master* no grupo envia regularmente anúncios à rede local para informar aos *hosts backup* que o mesmo está ativo. Se os *hosts backup* não receberem um anúncio do *master* por um período de tempo pré-determinado, então um dos *hosts backup* que tenha configurado os valores advbase e advskew baixos será promovido a *master* do agrupamento de *firewalls*. A Figura 2.3 mostra o tráfego do CARP capturado por tcpdump. (BOTELHO, 2006).

É possível múltiplos grupos CARP existirem no mesmo segmento de rede. Anúncios CARP contêm o *Virtual Host* ID que permite membros do grupo identificar qual grupo de redundância pertence (BOTELHO, 2006).

Para prevenir que um usuário malicioso no segmento de rede falsifique anúncios CARP, cada grupo pode ser configurado com uma senha. Cada pacote CARP enviado ao grupo é protegido por um HMAC SHA11[2] (BOTELHO, 2006).

[2]HMAC SHA1 é um algoritmo de autenticação chaveada de mensagem codificada

```
# tcpdump proto carp
        tcpdump: listening on msk0, link-type EN10MB
        11:32:54.783354 CARPv2-advertise 36: vhid=1 advbase=1 advskew=0
        demote=0 (DF) [tos 0x10]
        11:32:55.793413 CARPv2-advertise 36: vhid=1 advbase=1 advskew=0
        demote=0 (DF) [tos 0x10]
        11:32:56.803408 CARPv2-advertise 36: vhid=1 advbase=1 advskew=0
        demote=0 (DF) [tos 0x10]
        11:32:57.814685 CARPv2-advertise 36: vhid=1 advbase=1 advskew=0
        demote=0 (DF) [tos 0x10]
        11:32:58.823568 CARPv2-advertise 36: vhid=1 advbase=1 advskew=0
        demote=0 (DF) [tos 0x10]
        11:32:59.833640 CARPv2-advertise 36: vhid=1 advbase=1 advskew=0
        demote=0 (DF) [tos 0x10]
        11:33:00.843595 CARPv2-advertise 36: vhid=1 advbase=1 advskew=0
        demote=0 (DF) [tos 0x10]
        11:33:01.853868 CARPv2-advertise 36: vhid=1 advbase=1 advskew=0
        demote=0 (DF) [tos 0x10]
        11:33:02.863837 CARPv2-advertise 36: vhid=1 advbase=1 advskew=0
        demote=0 (DF) [tos 0x10]
        ^C
        5679 packets received by filter
        0 packets dropped by kernel
#
```

Figura 2.3: Tráfego CARP capturado via tcpdump

2.3 PF

O *packet filter*[3] é o sistema usado pelo OpenBSD para fazer filtragem e tradução do endereço de rede NAT em TCP/IP. O PF também é capaz de fazer normalização e condicionar tráfego TCP/IP e fazer controle de banda e priorização de pacotes. O PF tem sido parte do *kernel generic* do OpenBSD desde o OpenBSD 3.0. Versões anteriores do OpenBSD usavam um pacote diferente para firewall/NAT não mais suportado (OPENBSD, 2006b).

O *firewall* PF foi desenvolvido por Daniel Hartmeier para o projeto OpenBSD em 2001. Ele substituiu o antigo *software de firewall* chamado IPFilter devido a preocupações com licença de redistribuição. O criador do IPFilter, Darren Reed,

[3] http://www.openbsd.org/faq/pf

havia incluído várias estipulações de licenciamento em certas partes de seu *software*. A equipe OpenBSD achou melhor remover seu *software* e desenvolver sua própria solução. O *firewall* PF é um filtro de pacotes *stateful* que fornece controle de acesso à rede verificando pacotes que chegam ou saem e chegam através do PF. Um *firewall* como o PF é configurado através de políticas as quais determinam como os serviços de protocolos da *Internet* devem ser tratados.

O PF manipula pacotes para ele mesmo, e pode atuar como um filtro entre as máquinas de cada lado das interfaces da rede, mediando o acesso entre as redes e fazendo respeitar a política estabelecida. Isso pode acontecer sendo feito uma intercepção roteada ou pode ser apenas uma ponte transparente onde os pacotes são interceptados de forma silenciosa (HUMPHREY, 2008).

2.4 PFSYNC

A interface de rede PFSync expõe certas alterações feitas na tabela de estados PF. Através da ferramenta de monitoramento tcpdump[4], pode-se observar mudanças na tabela de estado. A interface pfsync envia mensagens de alterações de estado para a rede de modo que outros nós rodando PF possam atualizar as alterações em suas próprias tabelas de estado (BOTELHO, 2006). A figura 2.4 mostra a passagens de pacotes pfsync sendo enviados via *multicast*.

Quando o PFSync está configurado para enviar e receber atualizações na rede, o comportamento padrão é enviar atualizações *multicast*[5] na rede local. As atualizações são enviadas sem autenticação. A prática mais comum neste procedimento é (BOTELHO, 2006):

- Conectar através de um cabo *crossover*[6] os dois nós trocando atualizações de estado das conexões e usar a interface como syncdev;
- Usar a opção ifconfig syncpeer, assim atualizações são direcionadas com *unicast* [7] para o nó, configurando o ipsec entre os *hosts* para proteger o tráfego PFSync.

[4] http://www.tcpdump.org

[5] Multicast é a entrega de informação para múltiplos destinatários simultaneamente onde as mensagens só passam por um link uma única vez.

[6] Crossover consiste na interligação de 2 computadores pelas respectivas placas de rede sem ser necessário a utilização de um concentrador.

[7] *Unicast* é quando um endereçamento para um pacote é feito a um único destino.

```
# tcpdump -v proto pfsync
        tcpdump: listening on fxp0, link-type EN10MB
        13:50:19.353189 10.15.112.230 > 224.0.0.240: UPD ST COMP:
           (DF) [tos 0x10] (ttl 255, id 61586, len 216)
        13:50:19.355544 10.15.112.230 > 224.0.0.240: INS ST:
            (DF) [tos 0x10] (ttl 255, id 25774, len 280)
        13:50:19.456062 10.15.112.230 > 224.0.0.240: UPD ST COMP:
            (DF) [tos 0x10] (ttl 255, id 29395, len 392)
        13:50:19.942582 10.15.112.230 > 224.0.0.240: UPD ST COMP:
            (DF) [tos 0x10] (ttl 255, id 52619, len 568)
        13:50:20.348517 10.15.112.230 > 224.0.0.240: UPD ST COMP:
            (DF) [tos 0x10] (ttl 255, id 52876, len 480)
        13:50:20.496719 10.15.112.230 > 224.0.0.240: UPD ST COMP:
            (DF) [tos 0x10] (ttl 255, id 63700, len 480)
^C
17 packets received by filter
0 packets dropped by kernel
#
```

Figura 2.4: Tráfego PFsync capturado via tcpdump

2.5 FWBuilder

O FwBuilder é uma *Graphical User Interface* (GUI) para gerenciamento de *firewall*. É possível gerenciar mais de um *firewall* em qualquer estação da rede, podendo compilar complexas regras sem precisar saber como fazê-las em modo texto.

O FwBuilder pode ser utilizado para criar regras em vários sistemas, podendo gerar regras para Netfilter/Iptables, FreeBSD's IPFilter, OpenBSD's PF e Cisco PIX. As regras criadas são salvas em um arquivo XML, podendo ser compilada para os outros sistemas utilizando o mesmo arquivo, sendo isto muito útil na migração de algum *firewall*. Com ele é possível executar todos os comandos que normalmente são utilizados em um *firewall* utilizando apenas o *mouse* para criar as regras (GODOY, 2006).

Algumas funcionalidades do Fwbuilder:

- Permite comentários nas regras;
- Permite criar grupos de regras;

- Suporte a módulos;
- Multiplataforma.

A figura 2.5 exemplifica a criação de regras no FwBuilder, onde foi criada duas regras, sendo a primeira permissão da rede interna pra qualquer lugar na porta ftp, e uma regra para negar todo o resto.

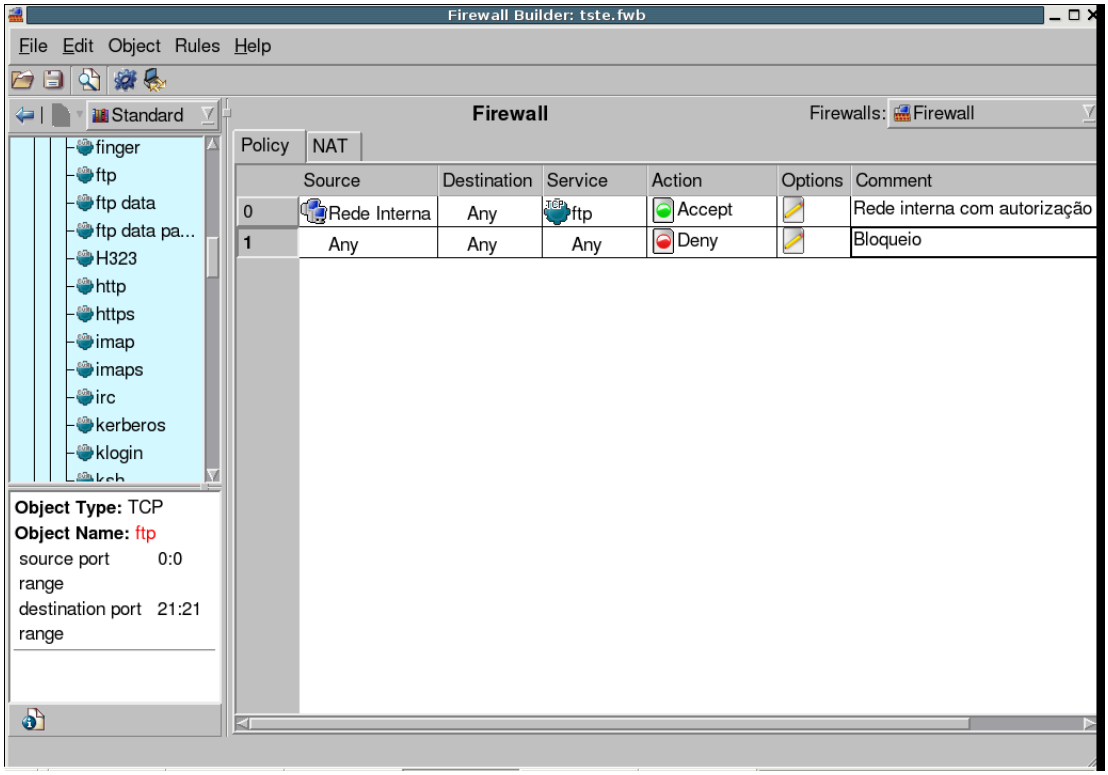

Figura 2.5: Firewall Builder

O apêndice A e B demonstram o arquivo gerado pelo FwBuilder na criação das regras.

Capítulo 3

Estudo de caso

Um *firewall* é um sistema ou grupo de sistemas que força uma política de acesso entre duas redes. Os *firewalls* tendem a ser visto como uma proteção entre a *Internet* e uma rede privada. Mas de modo geral, um *firewall* deve ser considerado como um meio para dividir o mundo em duas ou mais redes: uma ou mais redes seguras e uma ou mais redes não seguras. Um *firewall* pode ser um PC, um roteador, um *midrange*[1], um *mainframe*, uma estação UNIX ou uma combinação destes, determinando quais informações ou serviços podem ser acessados de fora (RIBEIRO, 2004).

Geralmente, um *firewall* está instalado no ponto onde a rede interna é segura e a rede externa é insegura, este ponto é conhecido como ponto de sufoco ou *choke point* (RIBEIRO, 2004).

3.1 Cenário

A figura 3.1 mostra a estrutura inicial da solução de *firewall* adotada para proteger os servidores da Intranet do Paraná. Esta solução, à princípio, atendia a demanda de acessos, contando com tráfego, número de sessões e pacotes por segundo.

A solução contava com dois *firewalls* utilizando o sistema operacional debian, com o *firewall* iptables, e placas de rede 10/100Mbps. Em cada lado do *firewall*,

[1]Dispositivo que realiza uma ou mais conexão entre um ou mais dispositivos. Em informática um midrange é um servidor para uma rede cliente servidor; trata-se, geralmente, de um computador de médio porte (RIBEIRO, 2004).

existia um switch de balanceamento, o qual fazia o balanceamento de carga que passavam pelos *firewalls*.

Devido ao grande número de regras, e ao alto número de acessos requisitados diariamente, foi necessário utilizar uma ferramenta gráfica para facilitar a sua criação. Para isso foi utilizado o fwbuilder, facilitando na gerência.

O Fwbuilder foi instalado separado, tendo uma máquina própria para seu uso. Após a criação da regra era feita uma cópia do arquivo até os *firewalls*, sendo compiladas as novas regras no *firewall*. Essas regras eram criadas muitas vezes ao dia; sendo assim, o *firewall* necessitava ser recompilado várias vezes. A figura 3.2 demonstra como a rede se comportava com um firewall parado.

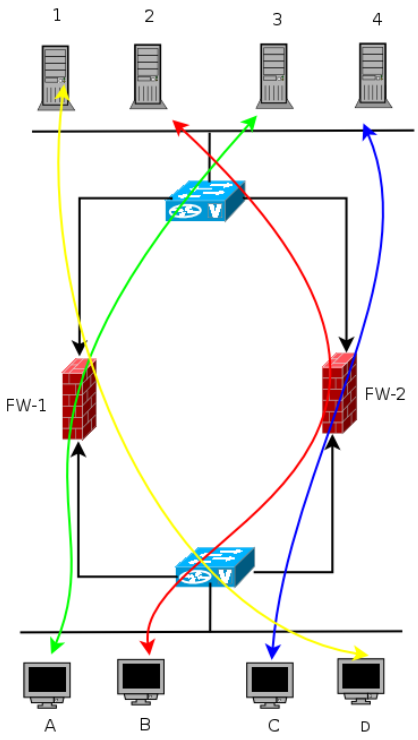

Figura 3.1: Rede utilizando *firewalls* Debian com iptables

3.2 Motivo da mudança

Vários motivos proporcionaram essa mudança:

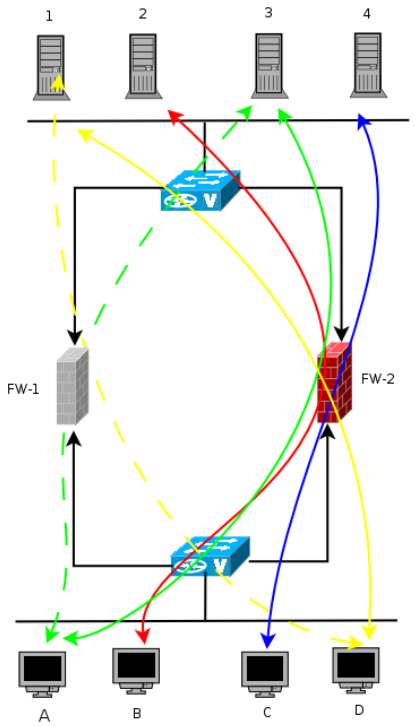

Figura 3.2: Comportamento da rede com a falha em um dos *firewalls*

- O script gerado pelo fwbuilder, por padrão, coloca *POLICY DROP* nas regras do iptables e, devido ao grande número de regras, isso passou a ser um problema durante a compilação do *firewall*. O problema ocorrido era devido a demora na instalação da regra e, como a *POLICY* estava como *DROP*, as conexões ativas eram derrubadas até que a nova compilação terminasse;

- As placas de rede utilizadas nos dois *firewalls* eram de 10/100Mbps, as quais estavam trabalhando no seu limite e, devido ao aumento exponencial de tráfego, esse limite logo seria excedido;

- Caso um dos *firewalls* parasse por algum motivo, o outro *firewall* não teria condição de assumir todo tráfego sozinho, isto é, não existia redundância na solução utilizada;

- Era necessário retirar os switches de balanceamento para utilização em outras soluções dentro da rede.

Além dos motivos descritos anteriormente, também foi buscado alguma alternativa que disponibilizasse mais segurança e um controle do que estava trafegando pelo *firewall*.

Foi adotado o sistema operacional OpenBSD juntamente com o *firewall* PF+CARP+PFSYNC para resolver esses problemas. Com essa solução, os problemas citados acima poderiam ser resolvidos e, ao mesmo tempo contando com um sistema operacional bem conceituado na área de segurança e com um *firewall* nativo chamado PF.

3.3 Migração

As expectativas para a reconfiguração da rede é ela poder suportar o seguinte:

- Ter uma melhor disponibilidade sem deixar a rede fora;
- Caso ocorra algum problema em alguma máquina, manter as conexões sem interrupção nos serviços.

Antes de iniciar a migração do sistema, foram utilizadas duas máquinas simples para compreender melhor os conceitos de carp e pfsync e poder se familiarizar com a nova solução.

Durante essa instalação, foram feitas as traduções das regras de IPTABLES para PF. Essas regras foram simples de traduzir, ocasionando apenas alguns ajustes em algumas configurações do PF juntamente com o Fwbuilder.

Os scripts de instalação das regras estavam prontos, isto é, faltava apenas colocar as máquinas para funcionar. A princípio achava-se que as duas máquinas menores iriam suportar e a idéia era apenas deixar alguns dias utilizando essas máquinas para testar a solução do *firewall* e fazer os devidos ajustes. Com isso, caso algum problema ocorresse, a solução seria voltar rapidamente para os *firewalls* debian.

Chegou o ponto em que a solução utilizando debian não suportou mais o tráfego da rede, ocorrendo a migração imediata para a nova solução. Como a rede é grande, o PF deveria ter sido configurado com valores especiais, isto é, logo que foi mudado, o número de sessões não foi suportado pois era o valor padrão que estava setado.

Logo, foi utilizado um *hardware* mais robusto para colocar o OpenBSD com o PF. A nova estrutura da rede é mostrada na figura 3.3, a qual demonstra que

as conexões ativas passam por um firewall e ao mesmo tempo o *firewall backup* contém as entradas das conexões existentes.

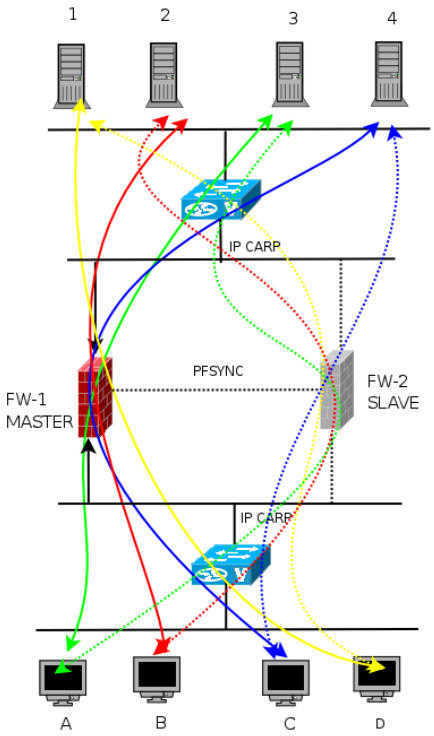

Figura 3.3: Utilização da nova estrutura de *firewall*

Com a migração, os problemas que ocorriam passaram a não ocorrer mais. Caso uma máquina caísse, a outra máquina assumiria sem a perda de conexão, como mostrado na figura 3.4

Ocorridos após a migração:

- A criação de regras padrão do Fwbuilder para o PF passou a ser com *POLICY ACCEPT*, o qual acabou deixando a rede aberta durante a instalação das regras, mas isso foi resolvido, pois apenas um *firewall* começou a ser utilizado pela rede, enquanto o outro passou a ficar como *SLAVE*. Assim, a regra era instalado no SLAVE primeiro e os *firewalls* passaram a ser alterados entre *MASTER* e *SLAVE* durante a instalação;
- Foi colocado uma placa de rede 10/100/1000Mbps, a qual suporta o tráfego da rede em somente uma máquina;

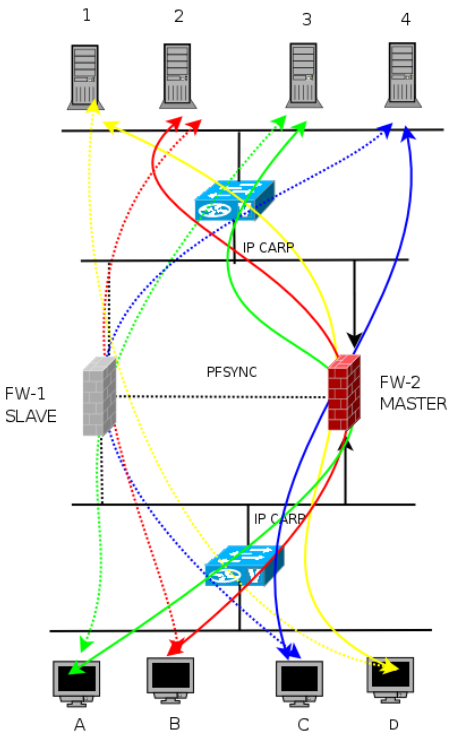

Figura 3.4: Alta disponibilidade com o openbsd

- Passou a ter redundância, isto é, caso um *firewall* caísse, o outro teria condições de assumir toda a rede sozinho;
- Os *switches* de balanceamento foram retirados, não sendo mais necessário fazer balanceamento, pois as duas máquinas passaram a compartilhar o mesmo ip virtual, e a queda de uma máquina passou a ficar de maneira transparente a rede, não derrubando as conexões ativas.

Capítulo 4

Implementação

Este capítulo tem por objetivo mostrar como foi feita a implementação do sistema apresentado no capítulo 3, detalhando a sua configuração.

4.1 Configuração

4.1.1 Configuração das interfaces

Para o funcionamento correto do *firewall*, deve-se configurar as interfaces e o gateway.

Para configurar o *gateway* é necessário adicionar ou editar o arquivo mygate, localizado no /etc. A figura 4.1 mostra o arquivo de configuração.

```
# vi /etc/mygate
    10.15.55.1
```

Figura 4.1: Configurar o gateway

Também é necessário configurar as interfaces físicas do *firewall*. Para isso, é necessário adicioná-las para que sejam inicializadas de forma automática ao iniciar o *firewall*. A figura 4.2 mostra os arquivos que devem ser configurados, a terceira interface é utilizada para fazer a sincronia de sessões entre os *firewalls*; assim, caso um dos *firewalls* deixe de funcionar, o outro assume a rede de forma transparente

sendo uma interface ponto-a-ponto. Lembrar que INTERFACE1, INTERFACE2 e INTERFACE3 devem ser substituidos pelo nome das interfaces da máquina.

```
# vi /etc/hostname.INTERFACE1
        inet 10.15.55.110 255.255.252.0 NONE
# vi /etc/hostname.INTERFACE2
        inet 10.15.60.100 255.255.255.0 NONE
# vi /etc/hostname.INTERFACE3
        inet 10.15.112.13 255.255.255.252 NONE
```

Figura 4.2: Configurar as interfaces utilazadas pelo tráfego da rede

Após as alterações é preciso reiniciar a máquina ou simplesmente reiniciar as interfaces de rede da máquina.

sh /etc/netstart

Para ativar o roteamento na máquina, é preciso alterar o arquivo /etc/sysctl.conf e descomentar a linha mostrada na figura 4.3.

```
net.inet.ip.forwarding=1
```

Figura 4.3: Ativar o roteamento da máquina

Essa alteração de roteamento passa a ser válida a partir do momento em que a máquina for reiniciada. Para que isso não ocorra, pode-se usar o comando apresentado na figura 4.4 para uma mudança imediata do roteamento. Lembrar que, se usar somente esse comando, o roteamento só funcionará até a máquina ser reiniciada.

```
# sysctl net.inet.ip.forwarding=1
   net.inet.ip.forwarding: 0 -> 1
```

Figura 4.4: Utilizar o comando sysctl para ativar o roteamento sem reiniciar a máquina

4.1.2 Ativando CARP

As funções básicas são gerenciadas por variáveis sysctl. A principal delas é net.inet.carp.allow. Ela é mostrada na figura 4.5, a qual demonstra que o sistema está preparado para o funcionamendo do carp.

```
# sysctl net.inet.carp.allow
net.inet.carp.allow=1
```

Figura 4.5: Permitir a utilização do carp

Para chegar se o sistema está configurado corretamente, pode-se usar o sysctl para ver as variáveis carp relacionadas. Após a configuração das interfaces físicas, deve-se ativar o CARP entre as máquinas. Para isso, deve-se alterar o arquivo /etc/sysctl.conf e adicionar ou editar as seguintes linhas mencionadas na figura 4.6.

```
net.inet.carp.allow=1
net.inet.carp.preempt=1
net.inet.carp.arpbalance=0
net.inet.carp.log=1
```

Figura 4.6: Configurações para o funcionamendo do carp

Para funcionar, pode-se reiniciar a máquina ou utilizar o comando sysctl para cada entrada.

As variáveis mais importantes são as duas primeiras, enquanto normalmente não é necessário alterar as duas últimas. Para manter o *failover* entre as máquinas, é necessário alterar o net.inet.carp.preempt, como mostrado na figura 4.7.

```
# sysctl net.inet.carp.preempt=1
```

Figura 4.7: Alterar a variável preempt

Observando que:

net.inet.allow: Aceita entrada de pacotes CARP ou não. Padrão é 1 (sim);

net.inet.arpbalance: Faz balanceamento por mac de origem através do arp;

net.inet.log: Registra pacotes CARP. O padrão é 0 (desabilitado);

net.inet.preempt: Permite *hosts* dentro de um grupo de redundância ter um melhor advbase e advskew para tomar o lugar (preempt) do *master*. Adicionalmente, esta opção habilita *failing over* nas interfaces caso uma interface deixe de funcionar (*down*). Se uma interface física CARP deixar de funcionar, o CARP trocará o advskew para 240 em todas as outras interfaces CARP, em essência o *failing* termina por si mesmo. Esta opção é 0 (desabilitada) por padrão.

As interfaces virtuais com seus ips virtuais também devem ser criadas, seguindo o mesmo modelo de criação das interfaces físicas.

Cada grupo de redundância é representado por uma interface de rede virtual, podendo ser configurada através do comando ifconfig ou da alteração do *hostname* do carp. A figura 4.8 mostra como configura uma interface carp utilizando o ifconfig e a figura 4.9 mostra como definir as interfaces carp no arquivo *hostname*.

```
ifconfig carpN create

   ifconfig carpN vhid vhid [pass senha] [carpdev carpdev] \
      [advbase advbase] [advskew advskew] [state estado] endereço-ip \
      netmask máscara
```

Figura 4.8: Configurar uma inteface carp

As configurações utilizadas para configurar a interface são:

carpN: nome da interface virtual carp(4) onde N é um inteiro que representa o número da interface (ex., carp10);

vhid: Este é um número único usado para identificar o grupo de redundância de outros nós na rede. Valores aceitos são 1 à 255;

senha: senha de autenticação para usar quando estiver conversando com outros hosts CARP neste grupo de redundância. Deve ser a mesma para os membros do grupo;

carpdev: Este parâmetro opcional especifica a interface de rede física que pertence a este grupo de redundância. Por padrão, o CARP tentará determinar qual interface usar procurando pela interface física que está na mesma subrede da combinação endereço-ip e máscara dada a interface carp(4);

advbase: Este parâmetro opcional especifica com que frequência, em segundos, anunciar quem são membros do grupo de redundância. O padrão é 1 segundo. Valores aceitos são 1 à 255;

advskew: Este parâmetro opcional especifica quanto desviar o advbase quando estiver enviando anúncios CARP. Manipulando o advskew, o host CARP master pode ser escolhido. Quanto mais alto o número, menos preferido será o host quando estiver escolhendo um master. O padrão é 0. Valores aceitos são 0 à 254;

estado: Força uma interface carp(4) em um certo estado. Estados válidos são init, *backup e master*;

endereço-ip: Este é o endereço IP atribuído ao grupo de redundância. Este endereço não tem que estar na mesma subrede que o endereço IP na interface física (se presente). Este endereço, contudo, precisa ser o mesmo nos hosts no grupo;

máscara: A máscara de subrede do IP compartilhado.

```
# vi /etc/hostname.carp1
        inet 10.15.55.225 255.255.255.0 10.15.55.255 \
        vhid 1 carpdev pcn0
# vi /etc/hostname.carp2
        inet 10.15.74.225 255.255.255.0 10.15.74.255 \
        vhid 2  carpdev pcn2
```

Figura 4.9: Configurar carp no arquivo hostname

Para fazer o teste, deve-se reinicar as novas configurações e verificar se as interfaces CARP estão assumindo como MASTER e no outro firewall como BACKUP.

4.2 Manter os estados sincronizados com pfsync

Para manter a redundância dos *firewalls* de forma transparente ao usuário, é necessário configurar o pfsync para existir uma sincronização da tabelas de estados utilizadas. Isto é, caso ocorra algum problema com um dos *firewalls*, o outro *firewall* irá assumir o tráfego da rede e não permitirá a queda das conexões que estavam ativas, pois através do pfsync, ela conhece as conexões executadas; sendo assim, o usuário nem percebe que houve um problema na rede. Para o funcionamento, é necessário interfaces configuradas com o pfsync.

O pfsync pode ser configurado em qualquer interface, mas é melhor configurá-lo em uma rede separada para a sincronização. Pode-se utilizar uma rede ponto a ponto para compartilhamento da tabela de estados dos *firewalls* e, caso exista mais que duas máquinas, pode-se utilizar um switch, hub ou vlan para a conexão.

4.3 Otimização das regras

O funcionamento de um filtro de pacotes não deve afetar o legítimo tráfego da rede. Na realidade, vários fatores limitam o quão bem um filtro de pacotes pode atingir

esse objetivo. Pacotes têm de passar através do dispositivo, adicionando uma quantidade de latência entre o momento em que um pacote é recebido e o momento em que é transmitido. Qualquer dispositivo só pode processar uma quantidade finita de pacotes por segundo. Quando os pacotes chegam a uma taxa mais elevada do que o dispositivo pode encaminhá-los eles são descartados(HARTMEIER, 2006).

A maioria dos protocolos, como TCP, lida bem com a adição latência. É possível alcançar altas taxas de transferência TCP nos *links* que têm várias centenas de milissegundos de latência. Por outro lado, uma rede de jogos interativos, mesmo com algumas dezenas de milisegundos se acaba percebendo a latência (HARTMEIER, 2006).

Uma unidade comumente utilizada para comparar o desempenho de rede é o *throughput* em *bytes* por segundo. Esta unidade é totalmente inadequada para medir o desempenho pf. Mas o verdadeiro fator limitante não é o *throughput* mas, a taxa de pacotes, que é o número de pacotes por segundo que o *host* pode processar. Um *host* com *link* de 10Mbps com pacotes de 40 bytes pode prejudicar mais o *firewall* do que um *host* com um *link* de 100Mbps com pacotes de 1500 bytes. Pois o *host* com *link* de 100 Mbps vai gerar 8 mil pacotes por segundo, enquanto o outro irá gerar 32 mil pacotes por segundo (HARTMEIER, 2006).

Para entender esse funcionamento se faz necessário entender como um pacote passa por um *host*. Os pacotes são recebidos do cabeamento através da placa de rede (NIC) e lido em um pequeno *buffer* de memória da placa de rede. Quando esse *buffer* está cheio, a placa aciona uma interrupção de *hardware*; assim, faz com que a placa copie os pacotes na memória do *kernel*. Nessas operações da memória, é somente inspecionado o cabeçalho dos pacotes. Isto também ocorre com o pf, que passando um pacote, faz a decisão em bloqueá-lo ou aceitá-lo (HARTMEIER, 2006).

Muitas das operações tem um alto custo por pacote, mas baixo custo pelo tamanho do pacote. Processando um grande pacote, é apenas um pouco mais caro do que processar um pequeno pacote. Muitos dos limites são baseados em *hardware e software* fora do pf e a escolha do *hardware* pode impor alguns limites em que otimizar do pf vai ajudar (HARTMEIER, 2006).

4.4 Configurando PFSTAT

Existe apenas um único arquivo, no qual é configurado o que será coletado e a forma que será mostrado na tela.

Um exemplo do arquivo pfstat.conf é mostrado na figura 4.10

```
        collect 1 = interface "sk0" pass bytes in ipv4 diff
        collect 2 = interface "sk0" pass bytes out ipv4 diff
        collect 3 = interface "sk0" block bytes in ipv4 diff
        collect 4 = interface "sk0" block bytes out ipv4 diff
        collect 99 = global states entries

; grafico

image "/var/www/htdocs/pfstat/bandwidth\_sk0\_hora.jpg" {
        from 1 hours to now
        width 980 height 300
        left
                graph 1 bps "in_pass" "bits/s" color 217 217 25,
                graph 2 bps "out_pass" "bits/s" color 0 255 0,
                graph 3 bps "in_block" "bits/s" color 255 0 255,
                graph 4 bps "out_block" "bits/s" color 255 0 0
        right
                graph 99 "states" "entries" color 0 0 0

}
```

Figura 4.10: Exemplo do arquivo de configuração do pfstat

Após a configuração do pfstat, é necessário fazer uma entrada no crontab para que o gráfico seja atualizado. A figura 4.11 mostra a configuração do crontab.

```
* * * * * /usr/local/bin/pfstat -q >/dev/null
*/5 * * * * /usr/local/bin/pfstat -p -c /etc/pfstat.conf >/dev/null
```

Figura 4.11: Configuração do crontab para atulização dos gráficos do pfstat

Capítulo 5

Ferramentas para plano de capacidade

"Com quanta largura de banda o PF pode trabalhar?"

"Que máquina deve-se utilizar para trabalhar com minha conexão Internet?"

Não existe respostas fáceis para essas questões. Para algumas aplicações, uma máquina simples com um par de placas de redes ISA pode filtrar e fazer NAT por perto de 5Mbps, mas para outras aplicações uma máquina mais rápida com placas de redes muito melhores podem não suportar esse tráfego de 5Mbps, pois a real questão não é o número de bits por segundo, mas a taxa de pacotes enviados e a complexidade das regras criadas.

A performace do PF é determinada por diversas variáveis (OPENBSD, 2006b):

- Número de pacotes por segundo. Quase o mesmo processamento necessário para ser feito com um pacotes de 1500 *bytes* é feito com um pacote de apenas um byte. O número de pacotes por segundo determina o número de vezes que a tabela de estados e, no caso de não ser encontrada, as regras de filtragens devem ser avaliadas a cada segundo, determinando a efetiva demanda no sistema;

- Performance no barramento do sistema. O barramento ISA tem uma largura máxima de banda de 8MB/seg, e quando o processador é acessado, ele tem que diminuir sua velocidade efetiva, não importando a velocidade do pro-

cessador. O barramento PCI tem maior efetividade com a largura de banda e menos impacto no processador;

- Eficiência da placa de rede. Algumas placas de rede são mais eficiente do que outras. Placas Realtek 8139 tendem a ter relativamente pior performace do que placas Intel 21143. Para máxima performace, é importante considerar o uso de placas gigabit, as quais tem um buffer mais avançado;

- Complexidade das regras. Quanto mais complexas forem as regras, mais lentas serão. Quanto mais pacotes são filtrados mantendo o estado e com regras rápidas, melhor é a performance. Tendo mais linhas para serem avaliadas por cada pacote, menor é o desempenho;

- Como o PF é baseado em processos de *kernel*, ele não usará espaço de swap. Então, se tiver RAM suficiente, ele roda. Não é necessário quantidade enorme de RAM, sendo que 32MB deve suportar em torno de 30,000 estados, o qual é bastante estados para um pequeno escritório ou para aplicações domésticas. Para a maioria dos usuários será suficiente um computador simples com placas de redes boas e uma boa base de regras.

5.1 PFTOP

O PFTOP indica os estados ativos do *packetfilter*, as regras e as filas e, periodicamente, essas informações são atualizadas (ACAR, 2002).

A figura 5.1 mostra algumas informações retiradas do comando pftop.

Algumas outras opções podem ser utilizadas juntamente com o pftop para obter mais informações sobre a utilização das regras. Algumas dessas opções são mostradas na figura 5.2.

5.2 PFSTAT

O PFSTAT é um pequeno utilitário que coleta estatísticas do filtro de pacotes (PFSTAT, 2007) sendo possível produzir vários gráficos. Ele monitora o *link* e o tráfego no *firewall*, pacotes que entram, que saem, bloqueados e que são liberados.

A figura 5.3 mostra o tráfego que passou pela interface na última hora.

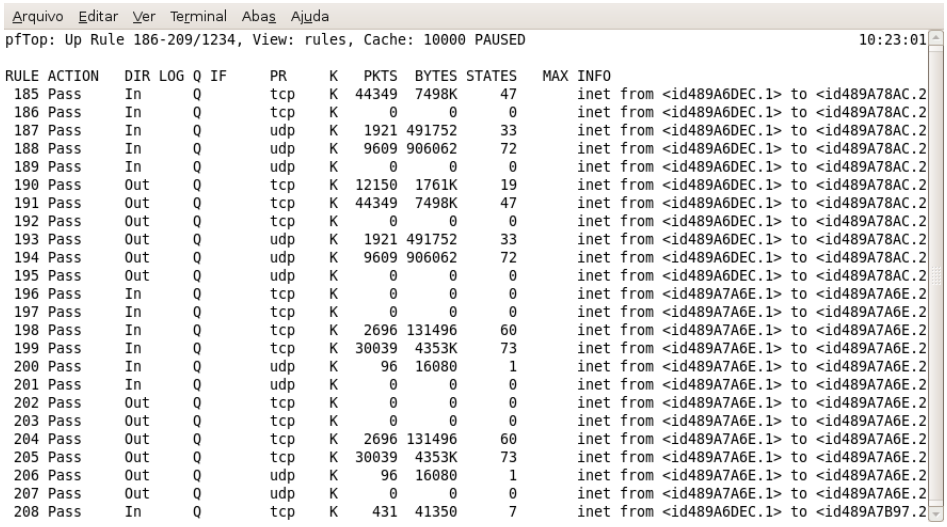

Figura 5.1: Exemplo de saída mostrada pelo comando pftop

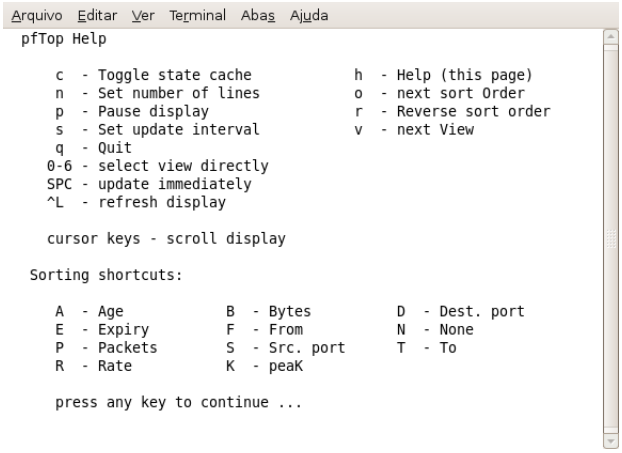

Figura 5.2: Opções que podem ser utilizadas juntamente com o pftop

Também pode ser visto o número de inserções e remoções feitas no *firewall*, como mostrado na figura 5.4.

Dentre as configurações, é possível escolher a cor utilizada, podendo ser preenchida ou não e facilitando assim a busca de alguma informação sobre o *firewall*.

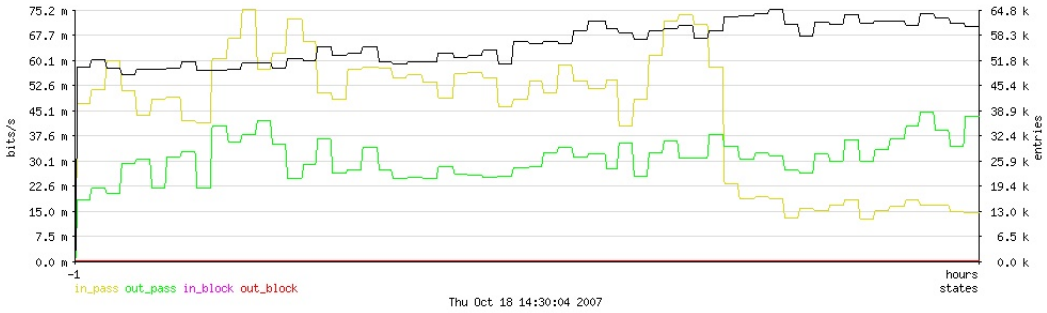

Figura 5.3: Tráfego que passou pela interface e o número de estados utilizados

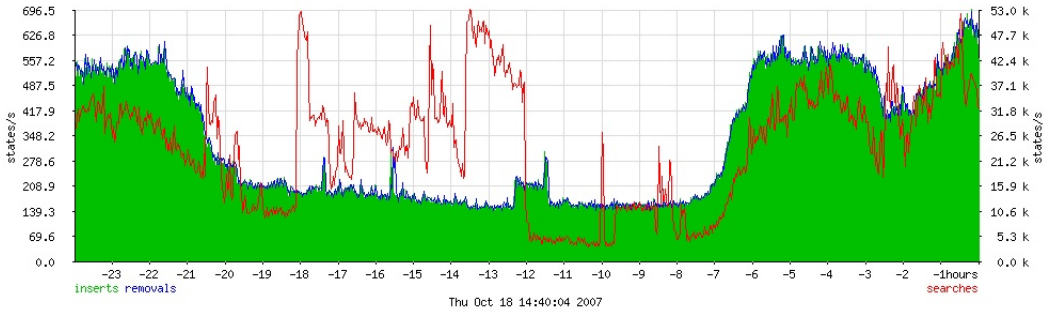

Figura 5.4: Número de inserções e remoções feitas no firewall

5.3 VMSTAT

Em sistemas Unix, atividades relacionadas a memória ocorre quando a memória real é utilizada e o *kernel* tem que utilizar a memória virtual. O vmstat lista uma série de estatísticas da memória virtual. A sintaxe do comando é (VMSTAT, 2008):

vmstat [-options [disks] [interval [count]]

O *interval* e o argumento *count* deve ser usado para especificar o intervalo de tempo em segundos e o número de intervalos para ser mostrado, permitindo monitoramento interativo do uso da memória (VMSTAT, 2008)

A figura 5.5 exemplifica o uso do vmstat.

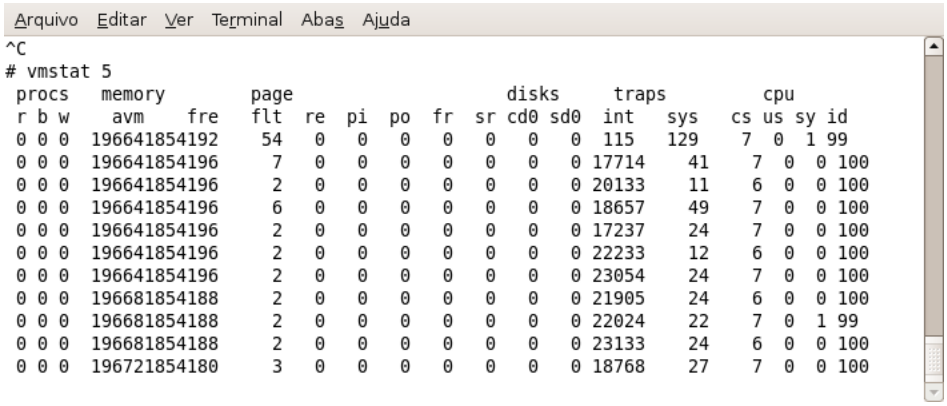

Figura 5.5: Saída do comando vmstat

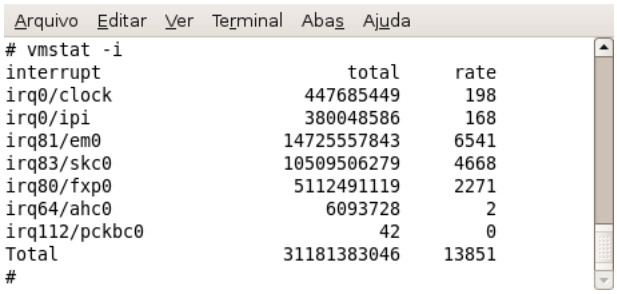

Figura 5.6: Número de interrupções geradas

5.4 SYSTAT

O systat mostra várias estatísticas do sistema. Juntamente com o systat pode ser usado estatísticas de entrada e saída (iostat), estatísticas de memória virtual (vmstat), rede, utilização de mbuf, estatísticas TCP/IP e conexões de rede (netstat) (SYSTAT, 2008)

Exemplos do comando systat são mostrados nas figuras 5.7 e 5.8

```
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
    1 users    Load  0.09  0.16  0.11                  Wed Aug  6 10:16:41 2008

Iface    State     Ibytes    Ipkts  Ierrs       Obytes    Opkts  Oerrs    Colls
em0      up:U     4082121     7308      0      1326684     6554      0        0
sk0      up:U     1239714     6559      0      4173164     7291      0        0
fxp0     up:U        4775       26      0      2296794     4185      0        0
vr0      dn:D           0        0      0            0        0      0        0
pflog0   up             0        0      0         3522       48      0        0
enc0     dn             0        0      0            0        0      0        0
lo0      up             0        0      0            0        0      0        0
pfsync0  up             0        0      0            0        0      0        0
carp1    up:U     1239714     6555      0          140        1      0        0
carp2    up:U     4082121     7292      0          155        1      0        0
Totals           10648445    27740      0      7800459    18080      0        0
```

Figura 5.7: Estatísticas e utilização das interfaces

```
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
    1 users    Load  0.30  0.20  0.12                  Wed Aug  6 10:15:28 2008

           memory totals (in KB)           PAGING   SWAPPING     Interrupts
          real   virtual     free          in  out   in  out    20922 total
Active    19992     19992  1854064   ops                          200 clock
All      207460    207460  2902340   pages                        101 ipi
                                                                 9491 em0
Proc:r  d  s  w    Csw   Trp   Sys   Int   Sof  Flt       forks    6349 skc0
          7         7     4    42 20620     4    7                          1   16
  0.0
   0.0%Int   0.2%Sys   0.0%Usr   0.0%Nic  99.8%Idle       pwait        pckbc0
|    |    |    |    |    |    |    |    |    |    |      relck
                                                         rlkok
                                                         noram
Namei         Sys-cache    Proc-cache    No-cache        ndcpy
    Calls     hits    %    hits     %    miss   %                        128
```

Figura 5.8: Estatísticas de interrupções e memória

Capítulo 6

Resultados Obtidos

Neste capítulo são apresentados os resultados obtidos com a migração, mostrando alguns problemas e limitações encontrados e o *tunning* feito para melhor desempenho do *firewall*.

6.1 Problemas e limitações

Durante a migração, foram encontrados alguns problemas e limitações na implementação. Dentre esses problemas e limitações podem ser citados:

- Fwbuilder: O fwbuilder até sua versão atual não suporta adicionar em sua interface mais de 100 mil sessões por segundo.Isto acabou sendo uma limitação, pois a utilização atual da rede necessita mais do que isso. Para solucionar essa limitação, foi feita a modificação no código do fwbuilder, adicionando um valor mais alto para ser instalado. O estudo de HOFLER CHRISTIAN BURKERT (2008) utilizando o OpenBSD 3.5 demonstra que o limite de sessões no firewall é de 100 mil conexões; isso é mostrado na figura 6.1. Na versão utilizada, OpenBSD 4.2, não existe essa limitação em torno do *sofware*, somente chegando ao limite de *hardware*;
- Como no início estava sendo trabalhado muito perto do limite de 100 mil sessões, qualquer utilização que precisasse um valor mais alto que isso, acabava derrubando as conexões que estavam ativas para entrar as novas conexões. Para solucionar foi alterado o valor para o limite de 200 mil sessões por segundo;

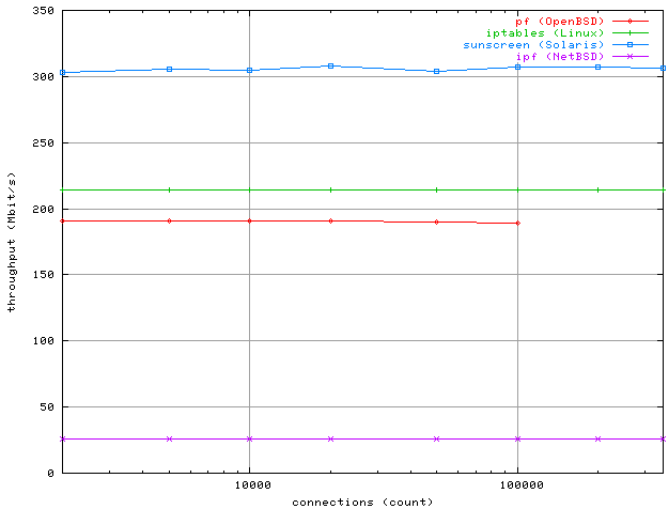

Figura 6.1: Limitação do número de sessões utilizada no firewall (HOFLER CHRISTIAN BURKERT, 2008)

- Na realidade, vários fatores limitam em como o PF pode atingir seu objetivo. Os pacotes são enviados através do dispositivo, adicionando latência desde a chegada do pacote até o seu encaminhamento. Qualquer dispositivo pode processar um número finito de pacotes por segundo. Quando a taxa de pacotes enviados é muito maior do que o dispositivo pode encaminhar, então os pacotes são perdidos (HARTMEIER, 2008).

  Uma unidade comum para comparar o desempenho da rede é o *throughput* em *bytes* por segundo. Mas, essa unidade de medida é inadequada para avaliar o desempenho do PF. O fator limitante não é o *throughput*, mas o número de pacotes por segundo que o *host* pode processar. O mesmo *host* que trabalha bem com 100Mbps de pacotes com 1.500 bytes pode não trabalhar bem com 10Mbps de tráfego com pacotes de 40 bytes. O primeiro exemplo forma 8.000 pacotes por segundo, mas o segundo exemplo trabalha com 32.000 pacotes por segundo, o que causa quatro vezes mais trabalho para o *host* (HARTMEIER, 2008).

  Os pacotes são recebidos pelo cabo pela interface de rede (NIC) e são lidos em um pequeno *buffer* de memória na placa de rede. Quando esse *buffer* está cheio, a placa de rede gera uma interrupção de *hardware*, ocasionando a cópia dos pacotes da placa de rede no *buffer* de memória do kernel. Alguns limites são baseados em *hardware e software* não relacionados com o PF. Um exemplo é máquinas i386 não trabalharem com muito mais de 10.000

interrupções por segundo, não importando a velocidade do CPU. Algumas placas de rede geram uma interrupção pora cada pacote recebido. O host irá perder pacotes quando a taxa passar de 10.000 mil pacotes por segundo. Outras placas de rede, como as placas gibabit mais caras conseguem alocar vários pacotes em apenas uma interrupção. Assim a escolha do *hardware* pode impor alguns limites que a otimização do PF não irá resolver (HARTMEIER, 2008).

As placas de rede utilizadas não estavam gerando muitas interrupções para o hardware da máquina, a qual é i386, visto que estava sendo gerado uma interrupção por pacote enviado À medida que esse número aumentava, o *hardware* não suportava mais e a rede passou a ficar mais lenta. Foi descoberto que a placa de rede, mesmo sendo gigabit, não suportava *interrupt coalescence*, a qual tem por objetivo adicionar mais de um pacote por interrupção.

A chegada e a saída de pacotes de uma placa de rede são dois eventos em que geram interrupções. No entanto uma interrupção pode gerar um estado de *livelock*, onde a CPU é totalmente consumida com o processamento das interrupções da rede ao invés do processamento dos dados contidos nos pacotes recebidos. Em uma placa Gibabit, pacotes de 1500 bytes pode chegar em um host a cada 12 mili-segundos. Se o processo de interrupção for maior que isso, acontece um *livelock* quando uma interrupção é gerada a cada pacote que entra (PRASAD; DOVROLIS, 2008).

Para reduzir o *overhead* da CPU e evitar um *livelock*, maioria das placas de rede com alta largura de banda usa *Interrupt Coalescence* (IC). IC é uma técnica em que as placas de rede conseguem agrupar múltiplos pacotes, recebidos em um curto periodo de tempo, em uma única interrupção. Assim a CPU consegue executar esses vários pacotes em apenas uma interrupção (PRASAD; DOVROLIS, 2008).

O objetivo básico da IC é reduzir o número de interrupções geradas em uma placa de rede em um pequeno período de tempo.

6.2 Tuning

Alguns limites devem ser alterados. A opção *limit* altera o tamanho do espaço da memória que o PF utiliza para as tabelas de estados e para as tabelas de endereço. Esses são limites de *hardware*, então deve-se alterar esses limites por várias razões. Caso a rede seja muito grande e congestionada, esses valores são muito

importantes, precisando alterar valores como o número de estados (HANSTEEN, 2008).

É importante lembrar que o total de memória disponível é retirada do espaço de memória do *kernel*. O *kernel* aloca uma quantidade fixa de memória para seu próprio uso, entretando, se essa memória não for trocada, o total de memória alocada para uso do *kernel* não pode exceder a memória física do sistema. Caso isso aconteça não existirá, espaço para os programas do usuário (HANSTEEN, 2008).

Para ver os limites utilizados, pode-se usar o comando pfctl -sm. A figura 6.2 mostra os limites utilizados para suportar a utilização do *firewall*.

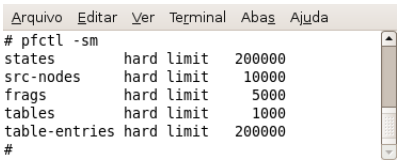

Figura 6.2: Limites utilizados no firewall

Para alterar esses valores, deve-se editar o pf.conf para incluir uma ou mais linhas com os novos valores estabelecidos.

A figura 6.3 mostra os limites alterados no firewall.

```
set limit { states 200000, src-nodes 10000, table-entries 200000 }
```

Figura 6.3: Alterar os limites

Pode ser alterada a otimização do *firewall*, os possíveis valores são *normal, high-latency, satellite, aggressive e conservative.* A recomendação é manter a opção padrão, a não ser que se tenha necessidades específicas. Os valores de *high-latency e satellite* são sinônimos, onde os estados expiram mais lentamente para compensar a alta latência. A *aggressive* expira antes os estados para economizar memória e a conservative preserva os estados e as conexões em modo idle, utilizando assim mais memória.

Outra alteração feita no *kernel* foi na variável net.inet.ip.ifq.maxlen, a qual define quantos pacotes podem ser enfileirados na fila de entrada antes dos pacotes serem derrubados. Pacotes vindo da placa de rede são primeiro colocados nessa fila. Placas Gigabit com *interrupt coalescence* podem colocar mais de um pacote por interrupção, melhorando o acesso. Caso a placa não suporte, uma grande quan-

tidade de pacotes pode estar sobrecarregando essa fila (BRAUER, 2008) (JEKER, 2008).

Assim que a fila aumenta, o número de pacotes derrubados aumenta; por isso, a variável net.inet.ip.ifq.drops acaba aumentando exponencialmente. Para melhorar essa fila, foi alterado o net.inet.ip.ifq.maxlen para um número maior, diminuindo o número de pacotes derrubados (BRAUER, 2008) (JEKER, 2008).

Capítulo 7

Conclusão

O OpenBSD e suas ferramentas, disponíveis gratuitamente sob a licença BSD, provêem uma solução robusta e acessível para roteamento e firewall redundante.

O PF se mostrou uma boa solução para firewall, tendo ótimas ferramentas para controle e redundância. Essa alternativa de firewall pode substituir boa parte dos firewalls que hoje em dia são proprietários, por exemplo: ASA e PIX da Cisco, Checkpoint e Junniper.

As soluções proprietárias contam com muitas ferramentas de controle e são mais interessantes para utilizar em uma rede grande, com alta complexidade. Logo, para uma rede de menor complexidade, o OpenBSD com PF, CARP e pfsync, atendem bem.

Proposta de trabalhos futuros

Para trabalhos futuros, pode ser implementado a utilização do ALTQ na solução de firewall, possibilitando controle de banda e limitando, por exemplo, o tráfego de backup que passa pelo firewall.

Outra opção de trabalho, é o CARP BALANCE, o qual pode ser usado para fazer um balanceamento de carga entre os firewalls, possibilitando a utilização das duas máquinas para passagem do tráfego.

Uma análise comparativa entre firewalls é uma proposta que pode ser desenvolvida, podendo envolver firewalls como PF, iptables, checkpoint.

Referências Bibliográficas

ACAR, C. E. *OpenBSD System Manager's Manual*. [S.l.], 2002. Disponível em: <http://www.eee.metu.edu.tr/%7Ecanacar/pftop/pftop.8.html>.

ATTEBURY, G.; RAMAMURTHY, B. *Router and Firewall Redundancy with OpenBSD and CARP*: Department of computer science and engineering university of nebraska-lincoln. [S.l.], 2008. 6 p.

BOTELHO, M. A. F. *Alta disponibilidade em firewall utilizando pfsync e carp sobre freebsd*: Monografia (pós-graduação em administração em redes linux) faculdade de ciências da computação, universidade federal de lavras, lavras. [S.l.], 2006. 53 p.

BRAUER, H. *Maxlen*. [S.l.], 2008. Disponível em: <http://www.webservertalk-.com/archive249-2006-10-1716005.html>.

FORSTER, A. P. S. *IPFW no Linux*. [S.l.], 1998. Disponível em: <http://www-.rnp.br/newsgen/9811/ipfw.html>.

FUGBR, R. *pfSense 1.2 está chegando*. [S.l.], 2002. Disponível em: <http://www.fug.com.br/content/view/334/54/>.

GODOY, L. *FWBuilder - Firewall para Iptables,Ipfw,Ipfilter,PIX*. [S.l.], 2006. Disponível em: <http://www.linuxnarede.com.br/artigos/fullnews.php?id=87>.

HANSTEEN, P. N. M. *The Book of PF*. San Francisco: William Pollock, 2008.

HARTMEIER, D. *Firewall Ruleset Optimization*. [S.l.], 2006. Disponível em: <http://undeadly.org/cgi?action=article&sid=20060927091645>.

HARTMEIER, D. *Spherix*. [S.l.], 2008. Disponível em: <http://spherix.jeka.ru-/index.php?page=2&prnt=1026>.

HOFLER CHRISTIAN BURKERT, M. T. T. *Comparative Firewall Study*. [S.l.], 2008. Disponível em: <http://archiv.tu-chemnitz.de/pub/2004/0148/data-/attachment.pdf>.

HUMPHREY, N. C. P. *Securing Financially Sensitive Environments with OpenBSD*: Department of mathematics royal holloway, university of london. [S.l.], 2008. 95 p.

JEKER, C. *Speed Problem*. [S.l.], 2008. Disponível em: <http://kerneltrap.org-/mailarchive/openbsd-misc/2007/9/26/323239>.

KASPER, M. *M0n0wall*. [S.l.], 2007. Disponível em: <http://m0n0.ch/wall/>.

OLIVEIRA, D. B. de. *Alta disponibilidade com CARP, Ifstated e pfsync*. [S.l.], 2006. Disponível em: <http://www.fug.com.br/content/view/124/77/>.

OPENBSD. *OpenBSD*. [S.l.], 2006. Disponível em: <http://www.openbsd.org-/pt/>.

OPENBSD. *OpenBSD*. [S.l.], 2006. Disponível em: <http://www.openbsd.org-/faq/pf/pt/>.

PFSTAT. *FWBuilder - Firewall para Iptables,Ipfw,Ipfilter,PIX*. [S.l.], 2007. Disponível em: <http://www.benzedrine.cx/pfstat.html>.

PRASAD, M. J. R.; DOVROLIS, C. *Effects of Interrupt Coalescence on Network Measurements*. [S.l.], 2008. Disponível em: <http://www.pam2004.org/papers-/265.pdf>.

PROJECT, F. D. *FreeBSD Handbook*. [S.l.], 2007. Disponível em: <http://www-.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-ipf.html>.

RIBEIRO, S. A. *Firewall em Linux*: Monografia (pós-graduação em administração em redes linux) faculdade de ciências da computação, universidade federal de lavras, lavras. [S.l.], 2004. 70 p.

SILVA, G. M. da. *Guia Foca GNU/Linux*. [S.l.], 2006. Disponível em: <http://focalinux.cipsga.org.br/guia/avancado/ch-fw-iptables.htm>.

SYSTAT. *FreeBSD General Commands Manual*. [S.l.], 2008. Disponível em: <http://fuse4bsd.creo.hu/localcgi/man-cgi.cgi?systat+1>.

THOMAZ, Y. *Dominando o iptables*. [S.l.], 2005. Disponível em: <http://www-.vivaolinux.com.br/artigos/verArtigo.php?codigo=2315>.

VINGAARD, M. *For network security, build a m0n0wall*. [S.l.], 2005. Disponível em: <http://www.linux.com/articles/113972>.

VMSTAT. *Overview of vmstat*. [S.l.], 2008. Disponível em: <http://users.ictp.it-/~radionet/nuc1996% -/ref/usail/tuning/vmstat.html>.

Apêndice A

Parte do arquivo de configuração criado pelo fwbuilder - Fw-Intra-BSD.conf

```
set limit  states 200000
    set optimization Normal
    scrub in all no-df
    scrub out all random-id
    table <id48A6BE98.1>  10.15.55.101 , 10.15.60.103 , 10.15.55.102 ,
    10.15.60.104 , 10.15.55.110 , 10.15.60.100 ,10.15.60.101 , 10.15.55.111
    table <id48A6BF4B.1>  10.15.112.13 , 10.15.112.14
    table <id48A6C03D.1>  sk0 , em0 , fxp0
    table <id48A6AAE7.2>  10.15.60.0/22 , 10.15.112.14 , 10.15.112.13
    , 10.15.40.0/24
    table <id48A6AAFE.1>  10.15.20.51 , 10.15.20.17 , 10.15.20.161 ,
    10.15.20.102 , 10.15.20.50 , 10.15.21.218 ,
    10.15.17.0 , 10.15.17.54 , 10.15.17.51 , 10.15.17.71 , 10.15.112.14 ,
```

10.15.112.13

table <id48A6C145.2> 10.0.0.0/8 , 200.189.113.0/24 , 200.201.0.0/17 ,

192.168.220.71 , 192.168.232.4 ,

192.168.220.2 , 200.189.112.0/24 , 200.214.44.177

table <id48A6C1B9.1> 10.15.60.146 , 10.15.60.147

Rule 0 (sk0)

CARP

#

pass in quick on sk0 inet from <id48A6BE98.1> to 224.0.0.18 keep state label "RULE 0 – ACCEPT "

pass out quick on sk0 inet from <id48A6BE98.1> to 224.0.0.18 keep state label "RULE 0 – ACCEPT "

#

Rule 0 (em0)

CARP

#

pass in quick on em0 inet from <id48A6BE98.1> to 224.0.0.18 keep state label "RULE 0 – ACCEPT "

pass out quick on em0 inet from <id48A6BE98.1> to 224.0.0.18 keep state label "RULE 0 – ACCEPT "

#

Rule 0 (fxp0)

PFSYNC

#

pass in quick on fxp0 inet from <id48A6BF4B.1> to any keep state label "RULE 0 – ACCEPT "

pass out quick on fxp0 inet from <id48A6BF4B.1> to any keep state label "RULE 0 – ACCEPT "

```
#
# Rule 0 (global)
#
#
pass in quick inet proto icmp from any to any label "RULE 0 – ACCEPT "
pass in quick inet proto tcp from any to any port 7 label "RULE 0 – ACCEPT
"
pass out quick inet proto icmp from any to any label "RULE 0 – ACCEPT "
pass out quick inet proto tcp from any to any port 7 label "RULE 0 – ACCEPT
"
#
# Rule 1 (global)
#
#
pass in quick inet from any to 127.0.0.1 keep state label "RULE 1 – ACCEPT
"
pass out quick inet from any to 127.0.0.1 keep state label "RULE 1 – ACCEPT
"
#
# Rule 2 (global)
#
#
pass in quick inet from 127.0.0.1 to any keep state label "RULE 2 – ACCEPT
"
pass out quick inet from 127.0.0.1 to any keep state label "RULE 2 – ACCEPT
"
#
# Rule 3 (global)
```

.

#

pass in quick inet from <id48A6C03D.1> to <id48A6C03D.1> keep state label "RULE 3 – ACCEPT "

pass out quick inet from <id48A6C03D.1> to <id48A6C03D.1> keep state label "RULE 3 – ACCEPT "

pass out quick inet from <id48A6C03D.1> to <id48A6AAE7.2> keep state label "RULE 3 – ACCEPT "

#

Rule 4 (global)

#

#

pass in quick inet from <id48A6AAFE.1> to <id48A6C03D.1> keep state label "RULE 4 – ACCEPT "

#

Rule 5 (global)

Regra para banir máquinas não cadastradas

#

block in log quick inet from any to 10.15.61.108 label "RULE 5 – DROP "

block out log quick inet from any to 10.15.61.108 label "RULE 5 – DROP "

Apêndice B

Script gerado para execução do arquivo de configuração - Fw-Intra-BSD.fw

```
#!/bin/sh
    #
    # This is automatically generated file. DO NOT MODIFY !
    #
    # Firewall Builder fwb_pf v2.0.9-1
    #
    # Generated Fri Aug 15 18:10:51 2008 BRT by adm-diser
    #
    # files: * Fw-Intra-BSD.fw
    # files: Fw-Intra-BSD.conf
    FWDIR=‘dirname $0‘
    log()
    test -x "$ LOGGER"&& $LOGGER -p info "$1"
```

```
add_addr()
addr=$1
nm=$2
dev=$3
( ifconfig $dev | egrep -q "inet +$addr ") ||
echo "$dev: $addr/$nm"
ifconfig $dev inet $addr netmask $nm alias
PFCTL="/sbin/pfctl"
SYSCTL="/sbin/sysctl"
LOGGER="/usr/bin/logger"
$SYSCTL -w net.inet.ip.forwarding=1
log 'Activating firewall script generated Fri Aug 15 18:10:51 2008 by admdiser'
$PFCTL -d
$PFCTL -F nat
$PFCTL -F rules
$PFCTL -F Sources
$PFCTL -F Tables
#
# Prolog script
#
#
# End of prolog script
#
$PFCTL -f $FWDIR/Fw-Intra-BSD.conf
$PFCTL -e
```

```
#
# Epilog script
#
# End of epilog script
#
```