OTIMIZADOR DE CONJUNTO DE REGRAS DE FILTROS DE PACOTES BASEADOS EM CADEIA

Thiago M. N. Oliveira
Instituto de Computação
Universidade Estadual de Campinas
13083-970 Campinas - SP
thiago.oliveira@ic.unicamp.br

João Porto de Albuquerque
Instituto de Computação
Universidade Estadual de Campinas
13083-970 Campinas - SP
jporto@ic.unicamp.br

Paulo Lício de Geus
Instituto de Computação
Universidade Estadual de Campinas
13083-970 Campinas - SP
+55 19 3788-5865 *paulo@ic.unicamp.br*

1 PALAVRAS-CHAVE

Firewalls, Escrutínio, Performance, Otimização, Automatização

2 PROBLEMA ALVO

Nos últimos anos, o uso de *firewalls* tem se tornado cada vez mais difundido nas arquiteturas de segurança de redes locais. Apesar de ser muitas vezes entendido erroneamente como uma simples aplicação para restrição do tráfego de pacotes através de um roteador, um *firewall* é composto por diferentes módulos[1]. A porção mais tradicional dos sistemas de *firewall* é o filtro de pacotes.

Filtros de pacotes são aplicações de baixo custo e que oferecem grande apoio à funcionalidade dos *firewalls*, através do escrutínio dos pacotes da rede[2]. Os filtros têm seu comportamento definido por regras definidas pelo administrador, de modo a restringir o tráfego através de um roteador.

A maioria dos filtros implementa suas regras em lista, sendo o tempo de teste para cada pacote passante da ordem de $O(n)$, sendo n o número de regras. Esse consumo de tempo não é desprezível, sobretudo em roteadores com muitas interfaces e com regras de controle de fluxo muito complexas, como em *backbones*, redes restritivos, *intranets* de grandes instituições, provedores de acesso à Internet, *sites* compartilhados por vários grupos distintos e outros casos semelhantes.

O problema abordado é garantir um bom tempo de acesso a qualquer regra de um conjunto de entrada, através de uma reestruturação deste, de modo a minimizar o número máximo de testes necessários para se alcançar uma regra. Além desse problema, considerou-se que uma verdadeira otimização deva garantir que as regras mais freqüentes tenham os menores tempos de acesso.

3 ESCOPO E RELEVÂNCIA

Os filtros de pacotes variam desde os mais simples, em que se pode apenas definir o tráfego permitido com base em algumas poucas características dos cabeçalhos dos pacotes IP, até aplicações que permitem uma distinção dos pacotes baseada no estado de conexões (ou pseudo-conexões UDP), uma vasta gama de características do pacote (incluindo dados de protocolos de diferentes níveis), extração de estatísticas e até limitação de volume de tráfego. Sua configuração se dá através da definição de regras, utilizando uma sintaxe específica para a aplicação utilizada. A tarefa de definir corretamente as regras para a filtragem, e geralmente dos demais componentes de um *firewall*, costuma ser atribuída ao administrador da rede e é bastante delicada, pois erros podem resultar em verdadeiros convites a ataques[1].

Muitos filtros apresentam suas regras de entrada estruturadas em cadeias distintas (os filtros *baseados em cadeia*) semelhantes a árvores, de modo a permitir que o administrador organize suas regras. Exemplos desses filtros são os do projeto *netfilter* (*iptables*, *ipchains* e *ipfwadm*)[3], para Linux, e o *ipfw*[4]. No *ipf* do *FreeBSD* há uma abstração equivalente que é a de grupos, além de haver, como no *ipfw*, a possibilidade de se saltar regras se o pacote avaliado satisfizer condições previamente definidas. Apenas essa funcionalidade já permitiria a construção do conjunto de regras de modo a se obter o mesmo efeito do uso de cadeias[5].

Muitas vezes essa estrutura em cadeias (ou uma análoga) é utilizada para melhorar a performance do filtro, como se propõe neste trabalho, porém manualmente. Não é incomum administradores precisarem dividir alguns milhares de regras em dezenas ou centenas de cadeias distintas, por seus filtros estarem prejudicando em demasia a performance do roteador. Como pode-se esperar, essa reestruturação raramente é simples, devido à dificuldade de se garantir que o conjunto reestruturado seja equivalente

ao original. Além disso, a melhoria de performance obtida pode ser muito aquém do ótimo.

Uma funcionalidade presente na quase totalidade dos filtros atuais e que poderia ajudar muito na definição de uma nova estrutura mais eficiente para as regras é a manutenção de *contadores de acertos* para cada regra definida. Tais contadores, geralmente mantidos em estruturas do *kernel* do sistema operacional e inicializados em zero, são incrementados cada vez que as propriedades definidas numa regra são satisfeitas pelas características de um dado pacote, aplicando-se a ele a ação definida na mesma regra. Contudo, apesar de ser possível detectar as regras com os mais elevados contadores de acertos e privilegiá-las tendo em vista uma melhor performance, levar os dados desses contadores em consideração ao se reestruturar um conjunto de regras manualmente é uma tarefa muito complexa.

Alguns poucos esforços para a automatização da otimização da estrutura do conjunto de regras dos filtros de pacotes já foram feitos. Uma iniciativa importante está presente no *pf* do *OpenBSD* e consiste em testar apenas a primeira de um determinado conjunto de regras adjacentes quando elas possuem um mesmo parâmetro e o teste deste falha na primeira regra[6]. Contudo, sua abordagem não garante sempre uma boa otimização, já que depende do posicionamento das regras no arquivo.

A ferramenta apresentada neste trabalho implementa um algoritmo para a reestruturação automática de um conjunto de regras que garante uma nova estrutura ótima ou boa, conseguindo redução efetiva do número máximo de testes necessários por pacote sempre que possível. O algoritmo também deve garante que o conjunto de regras construído sempre tem o mesmo efeito do conjunto original. A ferramenta apresentada vai além de utilizar apenas as características das regras na reestruturação, pois também leva em consideração aspectos dinâmicos do tráfego. Assim, garante-se que as regras cujos parâmetros são satisfeitos por mais pacotes, referidas neste documento como *regras mais freqüentes*, são as mais beneficiadas na estrutura criada pela ferramenta.

4 DESCRIÇÃO DA ARQUITETURA DE SOFTWARE

A ferramenta desenvolvida recebe um conjunto de regras estruturado e devolve um outro conjunto, equivalente ao original, mas reestruturado para garantir os menores tempos de cruzamento de um pacote com as regras.

A linguagem utilizada no desenvolvimento foi C, sendo definidas estruturas de dados genéricas para cadeias, regras e parâmetros. A definição de tais estruturas visa uma maior portabilidade da ferramenta para vários filtros distintos.

O projeto foi dividido em quatro módulos: *a)* um tradutor do arquivo de entrada para as estruturas genéricas, *b)* o ordenador das regras com base em seus contadores de acertos, *c)* o gerador da nova estrutura - uma aproximação de uma árvore binária e *d)* um tradutor das estruturas genéricas para a linguagem específica do filtro.

Os tradutores, *a)* e *d)*, são os módulos responsáveis pela interface com a sintaxe específica de cada filtro de pacote, enquanto o ordenador e o construtor da nova estrutura, *b)* e *c)*, são os responsáveis pela implementação do algoritmo otimizador, descrito posteriormente.

5 EMBASAMENTO TEÓRICO E FUNCIONAMENTO

5.1 Algoritmo Proposto

O algoritmo desenvolvido consiste em se retirar a regra mais freqüente de um conjunto de regras e então dividir tal conjunto em dois, com aproximadamente o mesmo número de regras. Essa divisão é feita de modo que a árvore resultante não seja degenerada, os dois conjuntos não sejam muito diferentes quanto à freqüência de suas regras e que haja um conjunto de parâmetros com os mesmos valores presente em todas as regras de um grupo e ausentes em todas as do outro grupo. Inserem-se, então, duas novas regras ao conjunto, sendo uma delas uma regra genérica no inicio, que possui como parâmetros apenas aqueles utilizados na divisão e que desvie a avaliação das regras para o grupo com tais parâmetros, caso um pacote os satisfaça. A outra regra criada é totalmente geral (todos os seus parâmetros são genéricos) e é inserida no final do conjunto, sendo sua ação a mesma da política padrão do conjunto de regras original.

O procedimento descrito é repetido para cada um dos dois grupos recém-definidos e assim sucessivamente, até que o número de regras num conjunto seja menor ou igual a quatro, quando a otimização é impossível.

5.1.1 Restrições à divisão e critérios de escolha

O processo de divisão de um dado conjunto satisfazendo as restrições não é trivial e se utiliza de alguns critérios de decisão para a eleição da melhor maneira dentre as possíveis, sendo necessário um esclarecimento mais detalhado.

Todas as regras de um conjunto são avaliadas, criando-se registros para cada valor de cada parâmetro presente em ao menos uma regra daquele grupo. Outros dados são mantidos durante essa avaliação, como o total de regras em cada registro e a soma de seus respectivos contadores.

É importante ressaltar que a divisão é realizada de forma complementar, ou seja, um grupo possui todas as regras com um determinado conjunto de parâmetros com certos valores, enquanto o outro possui todas as regras que não os possuem. Para manter essa propriedade, se existirem regras com

valores indefinidos em um desses parâmetros complementares, elas devem ser duplicadas e estar presentes nos dois conjuntos resultantes.

Afim de se evitar a degeneração da árvore, um limite superior para a diferença do número de regras dos dois conjuntos é utilizado, avaliando-se a diferença entre o número total de regras e o dobro do número de regras num dado registro. Tal limite é deixado a cargo do usuário. Dentre as possíveis divisões restantes escolhe-se aquela em que os dois grupos terão a menor diferença entre seus somatórios de contadores de acertos das regras, sendo essa considerada a melhor escolha.

Na prática será criada apenas uma nova cadeia de regras (grupo), com os parâmetros. O grupo em que estes estão ausentes será a própria cadeia original, considerando-se apenas depois das regras mais freqüentes e genéricas.

5.2 Prova do Algoritmo

Pré-Condição: Seja um conjunto de regras A, com uma política padrão e n regras independentes entre si em relação à posição. Cada regra possui um contador de pacotes acertados.

Pós-Condição: Uma estrutura semelhante a uma árvore binária em que a raiz é o início do conjunto original. O número de níveis da árvore é $L = O(\log n)$.

Hipótese de Indução: Em qualquer nível l todas as cadeias do mesmo sub-ramo e níveis superiores já estão estruturadas de modo que, juntas, sejam equivalentes ao conjunto original e as regras com maiores contadores estão nas primeiras posições. Conhece-se o número de regras em cada cadeia do nível l e o somatório de seus contadores.

Caso Base: Se A é folha ($n < 4$), então nada é feito.

Passo de Indução: Percorre-se as n regras de A colhendo-se os dados necessários para os critérios de decisão da melhor divisão, mantendo-se em separado a regra de maior contador. As possíveis divisões de A que não respeitam o limite de desbalanceamento a cada nível são desconsideradas.

Escolhe-se a melhor divisão. Um novo grupo B é criado e as regras escolhidas são movidas de A para B. Aplica-se o algoritmo a A' (remanescente de A após passo anterior) e a B. Uma regra para política padrão é criada e inserida no final das regras de B, fazendo valer a política padrão de A. Uma regra genérica em relação a todos os parâmetros, exceto aqueles que definiram a divisão, e com ação que desvia a avaliação das regras para B é criada e inserida no início de A'. Então a regra de maior contador é inserida no início de A'.

6 APLICAÇÃO PRÁTICA E RESULTADOS EXPERIMENTAIS

6.1 Dificuldades de Implementação

Alguns casos especiais precisaram ser considerados durante a implementação, pois não haviam sido pensados antes. Um deles foi a necessidade de se utilizar alguns parâmetros em conjunto. Isso ocorre porque há casos em que um parâmetro não pode ser utilizado sem o outro, como o par porta/protocolo.

Contudo, a questão mais complexa foi certamente a da generalidade, tanto com regras generalistas para um parâmetro quanto com a relação de endereços IP e seus endereços de rede.

Considera-se uma regra generalista para um parâmetro não definido, pois, nesse caso, qualquer valor satisfará tal parâmetro numa comparação com os dados de um pacote. Por aceitarem qualquer valor, quando um parâmetro escolhido para uma divisão não for definido para uma determinada regra, essa regra deve estar presente nos dois conjuntos resultantes.

Já quanto aos endereços IP e de rede, a questão é mais sutil. Naturalmente, se o parâmetro escolhido para a divisão for o endereço de origem ou de destino do pacote e seu valor for um endereço de rede, todas as regras com endereços IP pertencentes a essa rede no valor do mesmo parâmetro devem ser tratadas como se os valores fossem idênticos. Mas, ainda que o valor do parâmetro escolhido seja apenas um endereço IP, se houver alguma regra com o endereço daquela rede na seqüência da cadeia, pacotes daquele endereço nunca alcançarão regras especificadas para a rede em geral. Para contornar esse problema, as regras desse caso devem ser tratadas como no problema de regras generalistas para o parâmetro escolhido.

6.2 Limitações da Implementação

Devido à ferramenta ser ainda um protótipo, há algumas restrições e simplificações em sua implementação. A mais relevante delas é a não verificação da independência da posição das regras dentro de cada cadeia, uma das pré-condições do algoritmo desenvolvido.

Uma simplificação que deve ser mencionada é a não consideração de uma combinação de parâmetros que satisfaça as restrições ao se colher os dados para encontrar as possíveis divisões, mas sim o uso de um único parâmetro. Tal restrição não interfere na equivalência entre a nova estrutura de regras e a original, mas pode impedir a ferramenta de encontrar a estrutura ótima.

Por último, vale ressaltar que não pode haver variáveis no arquivo de entrada. Se o conjunto de regras original utilizar variáveis (o que é uma prática relativamente comum), elas devem ser substituídas pelos seus respectivos valores para que o protótipo seja capaz de reconhecer dependências en-

tre endereços IP e endereços de rede. Pode-se usar, para tanto, um *shell* como o *bash*.

O protótipo foi desenvolvido para trabalhar com o filtro *iptables*, do projeto *netfilter*, devido à rede do laboratório em que os testes foram feitos estar sobre plataforma linux e o *iptables* ser o filtro mais difundido nessa plataforma.

6.3 Resultados Obtidos

Os testes foram realizados no Laboratório de Mecânica Computacional (LabMeC), na Faculdade de Engenharia Civil da Universidade Estadual de Campinas - UNICAMP.

Tomou-se o cuidado de utilizar um roteador relativamente lento e *hosts* e rede rápidos, para que o gargalo seja nitidamente a filtragem. Para o roteador foi utilizado um Intel Pentium 166 MHz com 32 MB de RAM e os *hosts* foram um dual AMD MP 2,0 GHz e um dual Intel Xeon 2,0 GHz, ambos com 1 GB de RAM DDR. O teste foi feito numa rede ethernet 100 Mb/s isolada e consistiu em transferências pelo comando scp de arquivos de 50, 100, 500 e 1024 MB de um *host* para o outro.

Foram realizados dois conjuntos de testes: um utilizando um arquivo de entrada com 70 regras na cadeia de FORWARD do filtro e outro utilizando um arquivo com 800 regras na mesma cadeia. Em ambos os casos os resultados foram satisfatórios, apresentando as mesmas proporções no aumento de performance, o que deveria-se esperar de uma estrutura que se aproxima de uma árvore binária.

Utilizando as regras que permitiam o tráfego do teste tendo os contadores mais altos (melhor caso) o resultado foi exatamente o ótimo, já que elas realmente estavam nas primeiras posições do arquivo.

Já no pior caso, com as regras que dariam acerto como folhas da árvore, em ambos os testes observou-se que o tempo de transferência foi 50% do tempo no pior caso do conjunto de regras original. Ou seja, no pior caso, a filtragem de um pacote com a aplicação da estrutura de regras otimizada apresenta um *overhead* igual ao *overhead* médio obtido com a aplicação da estrutura de regras original.

É importante ressaltar que as regras mais privilegiadas são também as mais freqüentes. Assim sendo, apenas os serviços menos requisitados terão suas regras próximas do pior caso, apresentando um *overhead* de filtragem igual ao tempo médio com o uso do arquivo não otimizado.

7 BENEFÍCIOS ADVINDOS DO USO DA FERRAMENTA

A ferramenta apresentada implementa o algoritmo proposto, obtendo-se como resultado uma redução efetiva do número máximo de testes por pacote e conseqüente ganho real de tempo de filtragem. Além disso, o comportamento do conjunto otimizado de regras manteve-se equivalente ao do conjunto original.

Mesmo com a utilização de uma implementação limitada em relação à escolha das possibilidades de construção da estrutura em árvore, os resultados foram satisfatórios. O ganho de performance alcançado foi de pelo menos 50%.

Assim sendo, o uso da ferramenta apresentada mostrou-se indicado para administradores de redes cuja performance do roteador é uma questão importante, automatizando uma tarefa crítica e garantindo desempenho bom ou ótimo.

8 TRABALHOS FUTUROS

Como previamente explicado, o protótipo desenvolvido possui limitações em suas possibilidades de construção da árvore. Isso causou a construção de árvores com menos níveis do que o ótimo. Seria interessante implementar todas as possibilidades, incluindo regras com NAT (*Network Address Translation*), para que o programa fosse capaz de reconhecer características mais delicadas e ser capaz de construir árvores melhores. Os resultados certamente seriam ainda bem mais satisfatórios do que os que foram obtidos.

Um aspecto interessante é que a implementação apresentada deixa o limitante do desbalanceamento em cada nível a cargo do usuário, sem ajudá-lo em sua decisão. Poderia-se estudar alguma maneira de se encontrar o melhor (ou um bom) limitante para o desbalanceamento, de modo que o usuário não seja obrigado a fazer vários testes ou, simplesmente, utilizar um valor aleatório.

Não se estudou a freqüência em que uma reestruturação das regras é benéfica. Isso, com certeza, seria um campo muito interessante, pois poderia-se desenvolver um *daemon* que mantivesse as regras constantemente numa estrutura boa para o tráfego daquele momento.

Referências

[1] Zwicky, E. D., Cooper, S., Chapman, D. B. (2000). *Building Internet Firewalls*. 2nd Edition, O'Reilly and Associates.

[2] Cheswick, W. R., Bellovin, S. M., Rubin, A. V. (2003) *Firewalls and Internet Security: Repelling the Wily Hacker*. 2nd Edition, Addison Wesley.

[3] Netfilter Project. Disponível: *http://www.netfilter.org* . Acessado em 20/07/2003.

[4] IPFW Documentation. Disponível: http://www.rt.com/man/ipfw.4.html. Acessado em 20/07/2003.

[5] IP Filter Documentation. Disponível: http://www.openbsd.org/cgi-bin/man.cgi. Acessado em 20/07/2003.

[6] OpenBSD Packet Filter Documentation. Disponível: www.openbsd.org. Acessado em 20/07/2003.