Captura e Análise de Tráfego da Camada de Aplicação por Software com Alto Desempenho

Tiago Macambira* Dorgival Guedes Wagner Meira Jr.
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Hozionte - Minas Gerais - Brasil

E-mail: {tmacam,dorgival,meira}@dcc.ufmg.br

Resumo

O aumento da utilização comercial de redes de computadores tem aumentado a necessidade de se desenvolver sistemas que permitam aos administradores de redes monitorar e analisar o tráfego das mesmas em detalhes. As soluções disponíveis usualmente dependem de hardware especializado, de alto custo. Neste trabalho discutimos os desafios para se desenvolver uma ferramenta de monitoração e análise de tráfego da camada de aplicação em tempo real baseada em software, utilizando hardware convencional. Apresentamos uma arquitetura de coleta que permite a análise do estado das aplicações utilizadas na rede, realizando a recuperação do conteúdo trocado entre os usuários de uma rede P2P, por exemplo. Os resultados mostram que a arquitetura proposta apresenta uma taxa de perda de pacotes de apenas 3,3 % ao monitorar um tráfego a 500 Mbps, com um desempenho 99 % superior ao que é obtido utilizando uma abordagem tradicional. Mesmo a essa taxa de transmissão bastante elevada, o sistema ainda é capaz de recuperar 71 % dos dados trocados entre os usuários de uma aplicação P2P, mais que 100 vezes mais que a solução convencional.

1. Introdução

O crescimento da utilização de redes de computadores pelos variados setores da economia cria novas demandas para os administradores de redes em geral. Ser capaz de observar e analisar em tempo real o tráfego na rede é uma necessidade hoje para provedores de acesso em todos os níveis. Com o aumento da complexidade das aplicações, não basta identificar os fluxos em função dos endereços e portos envolvidos: é preciso também entender o que é transmitido por cada usuário. Esse entendimento é importante para planejamento estratégico em função do comportamento dos usuários, para detecção de anomalias de comportamento que podem indicar um ataque ou uma intrusão, ou apenas para criar soluções que tentem melhorar a experiência coletiva em função do uso de cada usuário, como no caso de *caches*.

Para ser capaz de analisar o tráfego tal como ele é entregue às aplicações envolvidas e recuperar as informações trocadas é necessário que o sistema de captura tenha um desempenho elevado. Taxas comuns em canais de saída de provedores de serviço banda larga, por exemplo, estão hoje na faixa de centenas de megabits por segundo ou mais. Um sistema de análise deve ser capaz de capturar e tratar os pacotes a essa taxa, com todo o procesamento necessário para recompor o comportamento dos usuários das aplicações de interesse. Com as taxas atuais, a arquitetura dos computadores de uso geral tradicionais (usualmente identificados pela sigla COTS, *commodity, off-the-shelf hardware*) está perto do seu limite de capacidade [7, 16]. Algumas soluções utilizando hardware especializado existem para o problema de captura de pacotes em enlaces de alta velocidade [16, 13, 23], mas com um custo elevado e limitadas à captura.

É importante salientar que o fato de existirem soluções que possibilitam a captura de pacotes em enlaces de alta velocidade com hardware comum não basta para que se possa construir sistemas que realizem recuperação de estado em tempo real nesse mesmo hardware. Há de se considerar que existe um custo não desprezível para processar o tráfego capturado para realizar a recuperação de estado. Além disso, tendo em mente que a tarefa de monitoração de tráfego é uma atividade onde não existe controle direto sobre a velocidade dos dados processados, se esse custo for muito alto, o sistema pode não ser capaz de realizar a sua função de maneira satisfatória.

Uma área que pode se beneficiar de recursos de

* Trabalho desenvolvido com o auxílio do CNPq.

monitoração de tráfego da camada de aplicação é, por exemplo, a de aplicações de troca de conteúdo baseadas em arquiteturas *peer-to-peer* (P2P). O tráfego desse tipo de aplicação tem crescido nos *backbones* e redes de acesso, preocupando operadores. Com um sistema de coleta e processamento de tráfego da camada de aplicação é possível desenvolver sistemas que possam, por exemplo, implementar *caches* passivas de conteúdo, limitar tráfego com conteúdo questionável ou aplicar políticas de qualidade de serviço diferenciadas por tipo de utilização.

Nesse contexto, neste trabalho apresentamos uma arquitetura de coleta capaz de realizar a recuperação de estado de aplicações por software com bom desempenho. Para isso, avaliamos o desempenho de diversas otimizações que podem ser aplicadas a um sistema operacional de uso geral utilizado para esse fim. Comparamos o desempenho da arquitetura resultante a uma solução equivalente sem as otimizações aplicadas e confirmamos ganhos elevados com a nova versão.

Com esse objetivo, o restante deste artigo está organizado da seguinte forma: a próxima seção discute trabalhos relacionados; em seguida, a seção 3 detalha os passos envolvidos na coleta de pacotes e recuperação de informação ao nível das aplicações, enquanto a seção 4 apresenta as soluções da arquitetura para esses passos; em seguida, as seções 5 e 6 apresentam a metodologia utilizada para realização dos testes de desempenho e os resultados obtidos; finalmente, a seção 7 apresenta algumas conclusões e possíveis trabalhos futuros.

2. Trabalhos Relacionados

Sistemas que utilizam captura e processamento de tráfego incluem sistemas de identificação de tráfego [23, 18], sistemas de detecção de intrusão à redes [12, 6, 19] e até mesmo sistemas de roteamento de pacotes [27].

Na literatura podemos encontrar diversos trabalhos que abordam o problema de monitoração passiva de tráfego utilizando sistemas operacionais de propósito geral [21]. Alguns propuseram novas arquiteturas ou arquiteturas aperfeiçoadas para captura [21, 20, 26]. Outros buscaram otimizar o processo de filtragem [30, 2, 14, 3], enquanto que alguns concentraram-se em diminuir o custo de recuperar o pacote da placa de rede para a área de memória do *kernel* e o custo da cópia de pacotes dessa área para a área de memória das aplicações [22, 11], transferir parte do processamento feito pelas aplicações para dentro do *kernel* [4, 17] e até paralelizar o processo de captura para aumentar o seu desempenho [28].

Paralelamente aos esforços para tornar a captura eficiente em sistemas comuns através de *software*, existem soluções que buscam melhorar o desempenho da coleta de pacotes otimizando por hardware alguns ou vários dos passos da captura [7, 13, 8, 9, 6]. No entanto, nenhum desses trabalhos apresenta uma solução definitiva. O uso de cada uma delas deve ser analisado frente às realidades de custo, desempenho e flexibilidade de cada projeto.

Em algumas circunstâncias, a captura e processamento em uma única máquina pode se mostrar inviável ou fazê-lo pode ser muito caro computacionalmente. Alguns trabalhos abordam o problema de realizar recuperação de estado nessas circunstâncias e buscam contorná-lo através da divisão do custo da monitoração por várias máquinas [19].

O argumento de que a arquitetura do PC está perto de atingir o limite do que ela pode oferecer à sistemas de captura de pacotes em redes com enlaces de alta-velocidade tem sido utilizado em favor de soluções de hardware [7, 16]. Entretanto, muitos trabalhos argumentam que ainda é possível capturar tráfego em velocidades de até 10 Gbps usando hardware comum (COTS) [9, 10].

Nosso trabalho é similar em espírito aos que buscam quantificar e melhorar a capacidade de captura de pacotes em sistemas operacionais de código aberto [10, 11]. Todavia, ao invés de abordar apenas a captura, nosso foco é analisar a viabilidade de se construir um sistema capaz de realizar monitoração passiva de tráfego da camada de aplicação, com recuperação de estado e utilizando hardware comum.

3. O processo de captura de pacotes

Para entender os desafios em se implementar um sistema como proposto, é preciso identificar precisamente os passos envolvidos em sua operação. Inicialmente, um sistema de captura de pacotes é inserido na rede de forma a receber uma cópia de todo o tráfego de interesse. A maior parte das soluções baseadas em hardware oferecem o recurso de se interpor diretamente o sistema no canal de comunicação, enquanto que a maioria das soluções baseadas em software, como no caso presente, assumem que o tráfego seja replicado por um elemento de rede para um segundo canal onde o coletor é instalado.

Cada quadro da camada de rede que deve ser capturado chega à interface de rede do sistema coletor, onde é transferido para a memória da placa de rede. De lá o pacote deve ser transferido para a memória principal da máquina, onde deverá ser processado. Essa transferência é feita pelo acionador de dispositivo (*device driver*), usualmente ao ser acionado por uma interrupção gerada pela placa. Em sistemas de coleta mais simples, duas coisas podem acontecer: se o sistema tem por objetivo coletar estatísticas de tráfego por fluxo, o cabeçalho é inspecionado para identificar o fluxo adequado, onde o pacote é contabilizado; se o sistema é configurado para coleta apenas (para processamento *off-line*), o pacote termina armazenado em memória secundária para acesso posterior.

No caso de um sistema de análise de tráfego da camada de aplicação, é preciso recuperar dos pacotes o conteúdo semântico da comunicação. Um primeiro passo é realizar o processamento da pilha TCP/IP que ocorreria nas extremidades da conexão, a fim de identificar os dados da aplicação. Em uma máquina usual recebendo apenas o tráfego que lhe é direcionado isso é feito no *kernel* do sistema operacional; no caso da captura, entretanto, os pacotes vistos se referem à comunicação entre diversas máquinas distintas e devem ser recompostos de forma especial.

Uma vez recuperadas as conexões TCP com o seu conteúdo, o próximo passo é interpretar o protocolo de comunicação das aplicações envolvidas. Isso exige que seja replicado todo o processamento da aplicação, a fim de identificar requisições e suas respostas, por exemplo. Finalmente, uma vez que o protocolo da aplicação é reconhecido, é possível extrair da comunicação os dados trocados pelos usuários das aplicações. Em um sistema de troca de arquivos P2P, por exemplo, pode-se recuperar os arquivos tranferidos entre os usuários.

Todo esse processo precisa ser implementado em um sistema de captura com recuperação do estado das aplicações e cada etapa acrescenta um processamento particular que precisa ser considerado. A transferência dos pacotes da placa de rede para a memória principal possui dois aspectos importantes: a utilização da banda do barramento interno da máquina e o tratamento das interrupções. Devido à impossibilidade de se utilizar a pilha TCP existente no *kernel* padrão, o processamento desta pilha também exige atenção. O processamento do protocolo da aplicação obviamente é específico para cada caso e deve ser desenvolvido também de forma especial, pois precisa considerar que em um mesmo sistema será replicado o comportamento de diversos usuários. Finalmente, é preciso incluir também a lógica que fará uso da informação extraída — por exemplo, um mecanismo de *cache* passiva que disponibilize para todos os usuários locais o conteúdo dos arquivos observados, ou um mecanismo de controle de conteúdo, que pode optar por bloquear transferências de arquivos que tenham sido identificados como nocivos segundo alguma política do provedor (distribuição não autorizada de material protegido pelas leis de direito autoral, por exemplo)

4. O Sistema

Considerando-se o processo de captura e análise de pacotes discutido, devemos então decidir como implementar a arquitetura de coleta para esse fim. Ao considerarmos uma solução que concentra seus esforços quase que em sua totalidade em *software* ou, mais especificamente, na melhoria de um sistema operacional de propósito geral, nos defrontamos com a possibilidade de realizar cada parte do trabalho de análise de tráfego em modo protegido (*kernel-mode*) ou em modo de usuário (*user-level mode*).

Sistemas construídos em modo protegido podem obter ganhos significativos de desempenho por não estarem sujeitos ao custo de troca de contexto e de cópia de dados entre o modo protegido e o modo usuário que sistemas desenvolvidos como aplicações simples têm que pagar. Além disso, esses sistemas têm acesso direto aos dispositivos de rede, o que permite acessar recursos não disponíveis através das interfaces do sistema operacional para aplicativos que operam em modo de usuário.

Todavia, a criação de sistemas que operem em modo protegido é mais laboriosa, tanto devido às limitações impostas pelos próprios sistemas operacionais à construção de código que execute em modo protegido quanto à dificuldade de depurar tal código. Soluções que buscam eliminar parte dessas complicações ainda são incipientes ou não possuem a flexibilidade necessária para a construção de um sistema como o planejado.

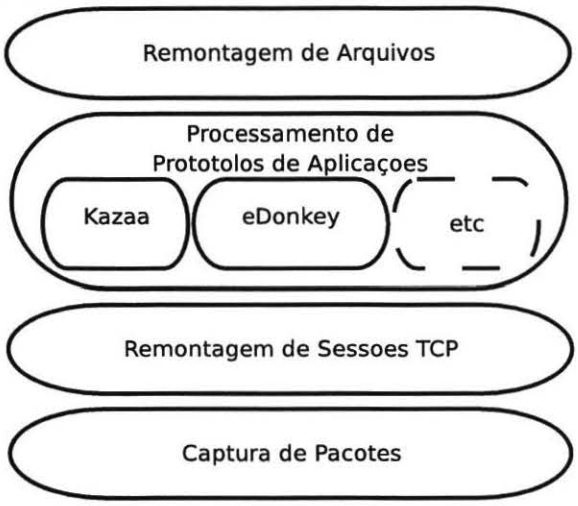

Figura 1. Organizacão em Camadas do Sistema

Optamos então pelo desenvolvimento da arquitetura em sua maior parte no modo usuário. Isso levou à criação de um sistema dividido em camadas que assemelha-se bastante a soluções de servidores de rede em nível de usuário. Em síntese, realizamos a construção de uma pilha de protocolos que será executada em modo de usuário [5, 25], como ilustrada na Figura 1.

Consideramos quatro camadas:

1. **Captura de pacotes:** para reduzir o custo da transferência dos pacotes da placa de rede para a memória principal, utilizamos um *patch* para o kernel do Li-

nux que implementa um *buffer* circular mapeado em memória do nível da aplicação para reduzir o custo das transferências entre o modo *kernel* e o modo usuário, denominado PF_RING [11].

2. **Remontagem de sessões TCP:** como discutido anteriormente, não é possível utilizar diretamente a pilha de protocolos do sistema operacional para fazer a análise dos pacotes. É necessário então um sistema que replique as máquinas de estados dos protocolos de forma eficiente a fim de garantir um desempenho suficiente para a operação em tempo real. Optamos então pela utilização da libNIDS [29], uma biblioteca de código aberto que se mostrou bastante eficiente nos nossos testes e que é capaz de realizar tanto a validação e desfragmentação dos pacotes IPs quanto a remontagem dos fluxos TCP em tempo real.

3. **Processamento dos protocolos das aplicações:** responsável pela análise dos protocolos das aplicações que se deseje monitorar. Para cada aplicação de interesse é preciso implementar as operações que seriam executadas na aplicação ao receber as mensagens da rede. A partir do uso da libNIDS foi definida uma interface através da qual *plugins* específicos para cada aplicação poderiam ser implementados e facilmente inseridos na arquitetura. Foram criados alguns desses *plugins*, para recuperar os dados das redes P2P eDonkey [15] e KaZaa [1].

4. **Remontagem de arquivos:** uma última camada da arquitetura foi concebida com o intuito de recuperar os arquivos transferidos pelos diversos protocolos de aplicação P2P monitorados, como base para uma *cache* passiva para os mesmos.

Em suma, exceto pela aplicação do *patch* do sistema operacional para reduzir o custo da comunicação com a placa de rede, o sistema de coleta opera completamente em modo usuário. O *patch* contribui também para reduzir o custo da troca de contexto entre os modos *kernel* e usuário, utilizando o *buffer* circular mapeado para a memória da aplicação.

5. Experimento

Para verificar o desempenho do sistema de coleta, executamos dois experimentos onde a sua capacidade foi verificada sob condições de carga variável. Para isso, utilizamos um *trace* de tráfego coletado de um provedor de acesso à Internet banda larga. Além de observar o desempenho da arquitetura, pretendemos também caracterizar mais precisamente as limitações encontradas em um sistema de código aberto e hardware usual. Além disso, pretendemos observar o que pode ser feito para melhorar o desempenho desses sistemas e como tais melhorias afetam o desempenho do sistema original.

Característica	No *trace*	Recebido
Número de Pacotes	7.928.526	15.857.052
Número de Conexões TCP	84.194	168.388
Tamanho em *Mbytes*	2.921	5.842
Tempo de Coleta	816,49s	*N/A*

Tabela 1. Características do *trace* e do tráfego observado pela máquina *A*

No primeiro experimento, variamos a taxa de envio dos pacotes a serem coletados para avaliar o comportamento do sistema sem otimizações. Com isso, pretendíamos avaliar o desempenho do sistema para entender as limitações da arquitetura original. Para tanto medimos a taxa de perda de pacotes. Essa taxa representa a incapacidade do sistema de lidar com o velocidade de envio de dados pela rede. Ela é conseqüência direta das ineficiências do sistema e, quanto maior ela for, menos dados o sistema terá para processar e portanto pior será a qualidade dos resultados que poderão ser obtidos pelo uso do sistema. No segundo experimento verificamos como a adição de cada nível da arquitetura alterava o desempenho da coleta, novamente observando a taxa de perda de pacotes do sistema.

Os experimentos utilizaram três microcomputadores baseados na arquitetura IBM-PC iguais, os quais chamaremos de computadores *A*, *B*, e *C*. Todos possuíam as mesmas especificações: processador Intel Pentium4 de 2.80GHz, 1 gigabyte de memória RAM DDR-400, placa-mãe Intel D865PERL, disco rígido SATA Seagate modelo ST3120026AS e placa de rede PCI Intel PRO/1000. Todas as máquinas foram interligadas por um *switch* Intel Gigabit Ethernet. A maquina *A* executava o *kernel* do Linux na sua versão 2.6.11.12. As versões usadas do *patch* PF_RING e da libNIDS são, respectivamente, a 3.0 e a 1.20.

As características do tráfego coletado no provedor banda larga e usado como carga nos experimentos é apresentado na Tabela 1. As máquinas *B* e *C* foram utilizadas para gerar a carga entregue à máquina *A*. Como cada uma continha uma cópia de todo o *trace*, o tráfego recebido pela máquina de coleta foi o dobro do conteúdo original coletado no provedor de banda larga.

Para que o tráfego pelas duas máquinas não fosse tratado como uma simples repetição dos mesmos pacotes utilizamos para o processo de envio o aplicativo tcpreplay [24], capaz de modificar deterministicamente os cabeçalhos IP presentes no tráfego a ser enviado. Dessa forma, para o objetivo da coleta e análise, o tráfego proveniente das duas máquinas geradoras de carga pode ser tratado como sendo indepen-

dente pela máquina de coleta. A taxa de envio do tráfego gerado conjuntamente pelas duas máquinas e que era entregue à máquina *A* foi ajustada com velocidades que variaram de 100 Mbps a 500 Mbps no primeiro experimento e que foram mantidas em 500 Mbps no segundo.

Na máquina *A*, onde o sistema foi instalado, monitoramos a quantidade de pacotes capturados com sucesso, de onde podemos determinar o número de pacotes perdidos, e como esse valor reagiria tanto à variação da velocidade do tráfego como à adição de cada nível da arquitetura:

1. apenas com a captura de pacotes,
2. com o processo de remontagem de conexões TCP através da libNIDS,
3. com o processo de análise de protocolos,
4. com o processo de remontar totalmente arquivos.

Obviamente, cada novo nível adicionado pressupõe a manutenção dos níveis inferiores a ele.

Três dimensões de otimização do sistema foram consideradas:

- o uso ou não do *patch* PF_RING
- a adição ou não de suporte a *polling* da interface no *driver* da placa de rede Intel Pro/1000, recurso ao qual nos referiremos no restante do texto por NAPI,
- o uso ou não da capacidade de HyperThreading (HT) do processador Pentim 4.

O recurso de *polling* é uma opção do *kernel* Linux que reduz a carga de interrupções geradas pela chegada de pacotes na interface. O recurso de HyperThreading da CPU Pentium 4 também é uma característica configurável no *kernel* Linux, cujo impacto também foi avaliado.

Foram consideradas as seguintes combinações das otimizações acima ou, como nos referiremos no restante do texto, configurações:

1. com PF_RING, NAPI e HT
2. com NAPI e HT
3. com PF_RING apenas
4. sem nenhuma das otimizações

Após os experimentos, verificamos que o uso de NAPI não causou nenhuma variação significativa nos resultados, mesmo com mudanças na taxa de envio da carga. Por esse motivo, retiramos dos gráficos e das estatísticas os dados referentes a configurações que diferiam apenas pelo uso ou não dessa otimização, para aumentar a clareza dos gráficos.

Para verificar a etapa de captura utilizamos o programa *pcount*, disponível com o pacote do *PF_RING*. Para verificar o desempenho com as demais camadas utilizamos versões modificadas do nosso sistema, limitadas a operar até a camada selecionada para cada teste. Para cada combinação de etapa e configuração foram feitas três execuções do experimento e a média dessas foi utilizada.

6. Resultados

Como discutido anteriormente, avaliamos o impacto do aumento da taxa de envio dos pacotes e das mudanças de configuração do sistema.

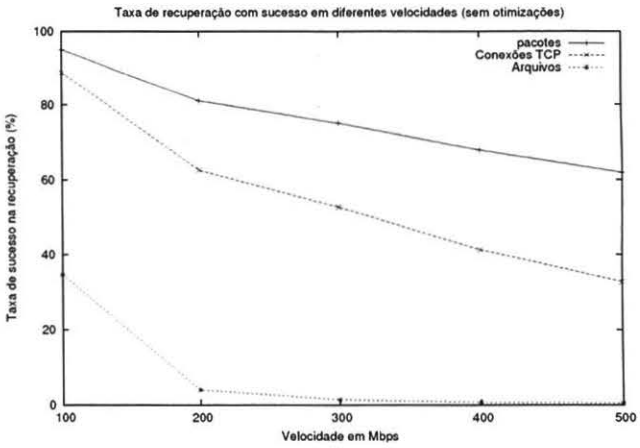

Figura 2. Impacto da taxa de chegada dos pacotes sobre a recuperação de pacotes, conexões TCP e arquivos

Inicialmente observamos o comportamento do sistema sem nenhuma otimização, observando como a velocidade de transmissão dos pacotes a serem monitorados influencia o desempenho do processo de coleta. Para cada taxa de transferência, verificamos quantos pacotes foram realmente entregues à aplicação, quantos estabelecimentos de conexões TCP a libNIDS foi capaz de identificar a partir dos dados e quantos bytes de dados foram retirados da aplicação pela camada de remontagem de arquivos.

Os resultados desse experimento podem ser observados na Figura 2, onde mostra-se o desempenho relativo de cada operação em relação ao total de pacotes, conexões e bytes de arquivos existentes no *trace* original. Como se pode ver, o desempenho da coleta de pacotes cai quase linearmente à medida que a velocidade aumenta. Mesmo com taxas de apenas 100 Mbps, um sistema sem otimizações já mostra perdas da ordem de 5 % na sua capacidade de capturar pacotes, atingindo 38 % a 500 Mbps.

Ainda na Figura 2, pode-se observar o impacto que a perda de pacotes acarreta na capacidade do sistema de reportar o estabelecimento de conexões TCP. É interessante

observar que a degradação na capacidade de identificar conexões TCP é muito mais pronunciada do que a que observamos na capacidade de captura de pacotes. Isso se deve ao fato de que um único pacote perdido durante o processo de estabelecimento da conexão implica na perda da conexão como um todo, logo uma perda tem impacto maior sobre a detecção da conexão, onde três pacotes TCP estão envolvidos (*three-way handshake*).

O efeito cumulativo das perdas se torna ainda mais notável no processo de remontagem de arquivos. Isso se deve ao fato de que o processo de recuperação de todos os dados de uma conexão TCP exige que todos os pacotes da conexão sejam processados. Uma vez que uma perda ocorre no fluxo de dados, a libNIDS não é capaz de continuar o processo de remontagem, mesmo que outros pacotes sejam recebidos com sucesso para aquela conexão. Com isso, mesmo para velocidades acima de 200 Mbps o sistema já é incapaz de recuperar mais do que 1,35 % do volume de dados disponíveis em condições ideais.

A Figura 3 mostra o total de pacotes capturados com sucesso para as quatro configurações testadas e para cada um dos quatro estágios. Nesse caso, como mencionado anteriormente, a carga foi gerada a 500 Mbps. Na legenda, *HT* se refere ao uso de *HyperThreading* e *NAPI* se refere ao uso ou não de *polling* de pacotes no *driver* da placa de rede. A numeração das etapas segue aquela usada na seção 5: (1) só captura de pacotes, (2) remontagem das conexões TCP, (3) processamento do protocolo da aplicação (eDonkey) e (4) remontagem dos arquivos trocados entre usuários.

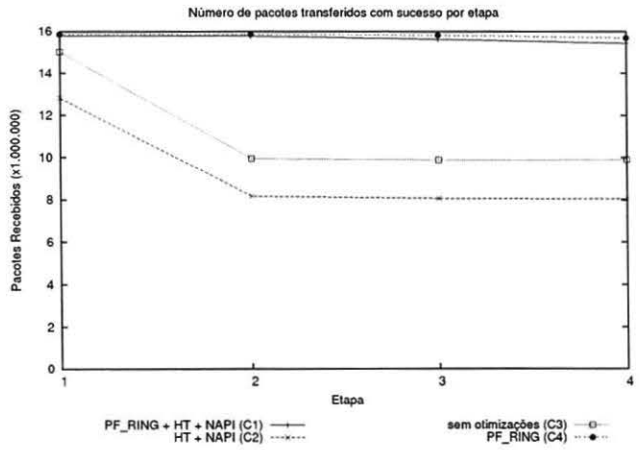

Figura 3. Quantidade de pacotes processados com sucesso

Primeiramente, é interessante notar o impacto negativo do uso do recurso de *HyperThreading* do Pentium 4: o desempenho de configurações com esse recurso habilitado são sempre inferiores àquelas sem o recurso. Acreditamos que o fato de não haver um uso intensivo nem de CPU, nem de *multithreading* e nem de paralelismo seja o motivo pelo qual o uso de HT não foi benéfico nesse caso.

Configuração	Conexões	Arquivos
PF_RING + HT + NAPI (C1)	97 %	57 %
HT + NAPI (C2)	21 %	0,4 %
sem otimizações (C3)	33 %	0,5 %
PF_RING (C4)	99 %	71 %

Tabela 2. Desempenho das várias con gurações a 500 Mbps

Outro elemento interessante é que o processo de remontagem das conexões TCP pode ser apontado como a tarefa mais cara no sistema. Isso é visível pela queda acentuada ao se incluir o estágio da libNIDS, enquanto as curvas praticamente permanecem estáveis após esse ponto. Apesar de a tarefa de remontar as conexões e processar as máquinas de estado TCP ser realmente importante, ela não é em si a responsável pela queda de desempenho. O problema nesse caso deve ser atribuído ao baixo desempenho da interface do sistema operacional sem o *patch* PF_RING para a transmissão dos pacotes. Esse impacto se torna mais visível quando a libNIDS tenta processar o conteúdo de cada pacote, o que não ocorre no caso do programa de testes que apenas verifica se o pacote foi recebido.

Como pode ser observado, é nítida a diferença de desempenho entre as configurações com e sem o *patch* PF_RING. Na primeira etapa, onde é feita apenas a captura do pacote, sem processamento posterior, a pior configuração, sem o *patch* e com HT e NAPI (C2), chega a perder 19 % dos pacotes que chegam à interface. Mesmo a configuração sem nenhuma otimização, ou seja, a configuração sem HT e sem o *patch* (C3), também já inicia com perdas significativas: 5 %. Na última etapa, a pior configuração sem o *patch* (C2) chega a perder 49 % dos pacotes, enquanto a pior configuração com o *patch* perde apenas 3 %.

Finalmente, na Tabela 2, pode-se observar que as configurações com PF_RING são aquelas que conseguem os melhores valores para a capacidade de recuperar conexões e arquivos do tráfego monitorado. O melhor resultado na recuperação de arquivos com o PF_RING, 71 %, é mais de 100 vezes superior ao que é obtido pelo sistema sem otimizações.

Como pode ser visto, a arquitetura resultante atende aos objetivos especificados, tendo demonstrado a capacidade de recuperar o estado das aplicações P2P consideradas. O sistema vem sendo usado para trabalhos de análise e caracterização de aplicações TCP na rede de um grande

provedor banda larga de Belo Horizonte, onde tem operado sobre um tráfego com banda média de 150 Mbps com perda mínima (por conseqüência, recuperando praticamente todo o conteúdo dos arquivos). Os resultados mostram que o uso de otimizções que minimizem o custo de cópia de pacotes do *kernel* do sistema operacional para o modo usuário podem oferecer grandes ganhos a sistemas de captura, mesmo que nestes ocorra processamento dos dados capturados, como ocorre no nosso sistema de 4 camadas.

7. Conclusões e Trabalhos Futuros

Este trabalho apresenta a arquitetura de uma ferramenta de coleta e análise de tráfego em redes de alta velocidade que permite até a recuperaçao do conteúdo trocado entre usuários da rede. Esse tipo de aplicação vem ganhando importância por auxiliar os gerentes das redes de acesso, provedores e *backbones* no entendimento do perfil de seus usuários e na definição e implementação de políticas de utilização das redes.

Como vimos, as ferramentas utilizadas para a construção dessa ferramenta se mostraram bastante satisfatórias. Com a sua utilização, conseguimos construir um sistema capaz de, além da monitoração, realizar a remontagem dos arquivos trocados no tráfego monitorado. Mesmo à velocidade de 500 Mbps, considerada alta para os objetivos propostos, o sistema trabalha com uma taxa de perda de pacote da ordem de 3,3 % e um desempenho 99 % superior ao que poderia ser obtido utilizando abordagens tradicionais. Mesmo operando nessa condição de carga extrema, o sistema ainda é capaz de recuperar 70 % dos dados observados no *trace*. Apesar desse valor não ser o ideal, ele é mais de 100 vezes superior ao que pode ser obtido utilizando-se um sistema não otimizado.

Outra contribuição do trabalho foi a identificação dos principais pontos de contenção no processo de coleta. Verificamos que grande parte das perdas em um sistema de coleta desse tipo são devidas ao *overhead* da passagem dos dados através da fronteira do sistema operacional. Com a utilização de uma atualização do *kernel* Linux que reduz o custo da transferência de dados para o modo usuário o sistema passou a apresentar um desempenho bastante superior ao original.

Como trabalhos futuros identificamos duas linhas principais de ação, uma focada no aprofundamento da análise do sistema e outra na aplicação da ferramenta desenvolvida. Na primeira, pretendemos instrumentar o *kernel* para identificar as principais causas de *overhead*, comparar o desempenho desse tipo de tarefa entre sistemas operacionais diferentes e avaliar outras técnicas de implementação, como o uso de uma máquina de estados TCP simplificada e o aproveitamento de recursos de hardware disponíveis nas placas de rede modernas (TCP *offloading*, por exemplo). Na segunda linha, o sistema de coleta desenvolvido já está sendo aplicado hoje em diversos estudos sobre o comportamento de redes P2P, avaliando tanto o processo de busca por recursos nessas redes, quanto o conteúdo dos recursos realmente transferidos pelas mesmas. Esses estudos permitirão uma caracterização melhor das aplicações P2P e a avaliação do impacto de técnicas como *caches* passivas e sistemas de policiamento e adequação de tráfego (*traffic shaping*).

Referências

[1] KaZaa home page, 2004. http://www.kazaa.com.

[2] M. L. Bailey, B. Gopal, M. A. Pagels, L. L. Peterson, and P. Sarkar. PathFinder: A pattern-based packet classifier. In *Proc. of the 1st Symposium on Operating System Design and Implementation*, 1994. USENIX Association.

[3] A. Begel, S. McCanne, and S. L. Graham. BPF+: exploiting global data-flow optimization in a generalized packet filter architecture. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, New York, NY, USA, 1999. ACM Press.

[4] H. Bos, W. de Bruijn, M. Cristea, T. Nguyen, and G. Portokalidis. FFPF: Fairly fast packet filters. In *Proceedings of 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, 2004.

[5] J. C. Brustoloni and P. Steenkiste. User-level protocol servers with kernel-level performance. In *Proceedings of the INFOCOM'98*, 1998.

[6] Y. H. Cho, S. Navab, and W. H. Mangione-Smith. Specialized hardware for deep network packet filtering. In *FPL '02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*, London, UK, 2002. Springer-Verlag.

[7] J. Cleary, S. Donnelly, I. Graham, A. McGregor, and M. Pearson. Design principles for accurate passive measurement. In *Passive and Active Measurement Workshop*, 2000.

[8] L. Degioanni, M. Baldi, F. Risso, and G. Varenni. Profiling and optimization of software-based network-analysis applications. In *SBAC-PAD '03: Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, Washington, DC, USA, 2003. IEEE Computer Society.

[9] L. Degioanni and G. Varenni. Introducing scalability in network measurement: toward 10 gbps with commodity hardware. In *IMC04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, New York, NY, USA, 2004. ACM Press.

[10] L. Deri. Passively monitoring networks at gigabit speeds using commodity hardware and open source software. In *In Passive and Active Measurement Workshop 2003*, 2003.

[11] L. Deri. Improving passive packet capture: Beyond device polling. In *4th International System Administration and Network Engineering Conference*, 2004.

[12] N. Desai. Increasing performance in high speed NIDS, 2002. http://www.linuxsecurety.com, last accessed on April 2005.

[13] Endance Measurement Systems. The DAG project, 2005. http://dag.cs.waikato.ac.nz, last access on April 2005.

[14] D. R. Engler and M. F. Kaashoek. DPF: fast, flexible message demultiplexing using dynamic code generation. In *SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 1996. ACM Press.

[15] T. Hoßfeld, K. Leibnitz, R. Pries, K. Tutschku, P. Tran-Gia, and K. Pawlikowski. Information diffusion in eDonkey filesharing networks. In *Australian Telecommunication Networks and Applications Conference (ATNAC 2004)*, Sydney, Australia, 2004.

[16] G. Iannaccone, C. Diot, I. Graham, and N. McKeown. Monitoring very high speed links. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement 2001*. ACM, 2001.

[17] S. Ioannidis, K. Anagnostakis, J. Ioannidis, and A. Keromytis. xPF: packet filtering for lowcost network monitoring. In *IEEE Workshop on High-Performance Switching and Routing (HPSR)*, 2002.

[18] T. Karagiannis, M. Faloutso, A. Broido, N. Brownlee, and K. C. Claffy. Transport layer identification of P2P traffic. In *Internet Measurement Conference 2004*, 2004.

[19] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2002. IEEE Computer Society.

[20] S. McCanne and V. Jacobson. The BSD packet filter: A new architecture for usert-level packet capture. In *Proc. of the Winter 1993 USENIX Conference*, pages 259–270, San Diego, California, 1993.

[21] J. Mogul, R. Rashid, and M. Accetta. The packer filter: an efficient mechanism for user-level network code. In *SOSP '87: Proceedings of the eleventh ACM Symposium on Operating systems principles*, New York, NY, USA, 1987. ACM Press.

[22] L. Rizzo. Device polling support for FreeBSD. In *BSDConEurope Conference*, 2001.

[23] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of P2P traffic using application signatures. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*. ACM Press, 2004.

[24] tcpreplay. tcpreplay's homepage, 2005. http://tcpreplay.sourceforge.org, last access on June 2005.

[25] C. A. Thekkath, T. D. Nguyen, E. Moy, and E. D. Lazowska. Implementing network protocols at user level. *IEEE/ACM Trans. Netw.*, 1(5), 1993.

[26] J. van der Merwe, R. Cáceres, Y. hua Chu, and C. Sreenan. mmdump: a tool for monitoring internet multimedia traffic. *SIGCOMM Comput. Commun. Rev.*, 30(5), 2000.

[27] G. van Rooij. Real stateful TCP packet filtering in IP Filter, 2001. Unpublished invited talk, Tenth USENIX Security Symposium, http://www.usenix.org/events/sec01/invitedtalks/rooij.pdf, last access on June 2005.

[28] G. Varenni, M. Baldi, L. Degioanni, and F. Risso. Optimizing packet capture on symmetric multiprocessing machines. In *SBAC-PAD '03: Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, Washington, DC, USA, 2003. IEEE Computer Society.

[29] R. Wojtczuk. libNIDS homepage, 2005. http://libnids.sourceforge.net, last access on June 2005.

[30] M. Yuhara, B. N. Bershad, C. Maeda, and J. E. B. Moss. Efficient packet demultiplexing for multiple endpoints and large messages. In *USENIX Winter*, 1994.