Controlo de Tráfego num *Edge Device* Multi-Serviço Linux

J. Urbano, E. Monteiro
{stmaria, edmundo}@dei.uc.pt

Laboratório de Comunicações e Telemática
CISUC / DEI
Universidade de Coimbra
Pólo II, Pinhal de Marrocos, 3030-290
Coimbra
http://lct.dei.uc.pt

A. Alves, A. Raposo
{aalves, raposo}@criticalsoftware.com

Critical Software, S.A.
EN1/IC2 Km 185,6
Banhos Secos - Santa Clara
3040-032 Coimbra – Portugal
http://www.criticalsoftware.com

Resumo

A vulgarização das redes de comunicação de pequena e média dimensão em ambientes *Small Office/Home Office* (SOHO) tem impulsionado a procura e o desenvolvimento de pequenos dispositivos de fronteira multi-serviço para o acesso à Internet. Uma das componentes de primordial importância destes dispositivos é o subsistema de controlo de tráfego, que permite optimizar a utilização e a distribuição da largura de banda no ponto mais crítico da rede, o acesso à Internet. O trabalho descrito neste artigo insere-se num projecto em desenvolvimento de cooperação entre o Laboratório de Comunicações e Telemática da Universidade de Coimbra e a empresa Critical Software, e que tem como objectivo a construção de um dispositivo de fronteira multi-serviço sobre Linux. Mais concretamente, este artigo aborda os principais problemas encontrados na integração do subsistema de controlo de tráfego com o subsistema de *firewall* e com *gateways* de aplicação específicos num mesmo dispositivo Linux – Edge Device. Sendo uma área esparsamente documentada na comunidade Linux, este artigo apresenta algumas soluções para os problemas referidos.

Palavras Chave: Controlo de Tráfego; Dispositivos de Fronteira; Linux.

1. Introdução

A utilização de redes informáticas de pequena e média dimensão com acesso à Internet tem vindo a vulgarizar-se em ambientes domésticos e em empresas e instituições de pequena e média dimensão, normalmente abrangidas pela designação *Small Office/Home Office* (SOHO). Inerente ao crescimento deste tipo de redes está a procura de pequenos dispositivos de acesso que permitam a interligação à Internet de um modo extremamente simples e intuitivo, do ponto de vista do utilizador, mas com complexidade crescente a nível dos serviços oferecidos, tais como os serviços básicos IP (por exemplo, DNS, DHCP, SMTP e NAT) e os serviços de segurança e de controlo de tráfego.

Tradicionalmente, estes serviços são oferecidos em dispositivos dedicados e de complexa integração, utilizando tecnologias proprietárias e muitas vezes patenteadas, e com custos de aquisição, integração e manutenção extremamente elevados. Mais recentemente, têm vindo a surgir alguns dispositivos multi-serviço, tal como o 6WINDGATE da 6WIND [1[1], o Access Point da Lucent Technologies [2] e o ERX Edge Router da Juniper Networks [3] que, embora integrando num único dispositivo os diversos serviços IP, apresentam ainda alguma complexidade a nível de gestão e custos algo elevados.

Uma das componentes que tem vindo a ganhar relevo neste tipo de dispositivos de fronteira é o subsistema de controlo de tráfego, responsável pela gestão de largura de banda no acesso à Internet. A importância deste subsistema é inequívoca, tendo em conta que o acesso ao ISP (*Internet Service Provider*) é geralmente o ponto de estrangulamento de uma rede de comunicações, e que o custo da largura de banda alugada a um ISP é ainda bastante elevado. Atendendo a que alguns ISPs nacionais e internacionais começam já a disponibilizar serviços *DiffServ* (*Differentiated Services* [4]) e MPLS (*Multiprotocol Label Switching* [5]), os subsistemas de controlo de tráfego tenderão a incorporar funcionalidades de nó de fronteira de redes *DiffServ* e/ou MPLS.

O presente artigo descreve o subsistema de controlo de tráfego no acesso à Internet de um dispositivo de fronteira multi-serviço Linux, uma das componentes de um projecto de grande dimensão a ser desenvolvido em conjunto pelo Laboratório de Comunicações e Telemática da Universidade de Coimbra e pela empresa Critical Software. Mais concretamente, serão abordadas as questões (até à data apenas vestigialmente documentadas na comunidade Linux) associadas à complexidade da integração, num mesmo dispositivo, do subsistema de controlo de tráfego com as funcionalidades de encaminhamento, NAT e *firewall*, e com *gateways* de aplicação específicos (por exemplo, o *proxy* Squid), quer para tráfego de saída (*upstream*), quer para tráfego de ingresso (*downstream*). Será ainda abordada, neste artigo, a utilização das funcionalidades que o Linux oferece para a marcação *DiffServ* do tráfego de saída.

A estrutura do artigo é descrita a seguir. Na Secção 2, é feita uma breve apresentação do subsistema de controlo de tráfego e da arquitectura *netfilter* do Linux.

Na secção 3, é feita uma caracterização do dispositivo de fronteira em desenvolvimento, com especial destaque para as questões relativas ao controlo de tráfego no acesso à Internet.

A secção 4 apresenta algumas das questões a ter em conta na integração do subsistema de controlo de tráfego com o subsistema de *firewall*, e são apresentadas algumas das soluções encontradas, tanto a nível de tráfego *upstream*, como de tráfego *downstream*. Ainda nesta secção, é apresentada uma avaliação preliminar aos testes de interoperabilidade e integração das soluções apresentadas.

Na secção 5 são discutidas e avaliadas as soluções de controlo de tráfego utilizadas.

Por fim, na secção 6 são apresentadas as conclusões deste artigo e referido o trabalho futuro.

2. Controlo de Tráfego em Linux

Nesta secção vai ser analisado controlo de tráfego no Sistema operativo Linux e a sua utilização no suporte de *DiffServ*.

2.1 Subsistema de Controlo de Tráfego

O Linux possui, a partir da versão 2.2 do seu *kernel*, um sistema de rede completamente redesenhado de raiz [8], sendo o novo código de encaminhamento, filtragem e classificação tão ou mais poderoso do que aquele fornecido por muitos *routers* dedicados e produtos de *firewall* e de modelação de tráfego.

Uma das componentes do sistema é o subsistema de controlo de tráfego (TC), escrito por Alexey N. Kuznetsov. O seu funcionamento baseia-se na combinação de três elementos básicos – disciplinadores, classes e filtros de tráfego – que, em conjunto, permitem a

classificação, a prioritização, a partilha de largura de banda e a limitação de tráfego em interfaces de entrada e de saída.

As características mais relevantes deste subsistema enumeram-se de seguida:

- Implementação estável (a partir do *kernel* 2.4) de vários disciplinadores: FIFO, TBF, SFQ, PRIO, CBQ, CSZ, TEQL, WRR, HTB, RED, GRED e DS_MARK.
- Capacidade para a configuração hierárquica de classes de tráfego.
- Oferta de um conjunto rico e diversificado de filtros, dentro de cada disciplinador, incluindo filtros *u32*, *fw*, *route*, rsvp, *rsvp6*, *tcindex*, *protocol*, *parent*, *prio* e *handle*.
- Implementação de um dispositivo virtual, IMQ (*Intermediate Queueing Device*) [9], que estende até à interface de entrada (*ingress*) as capacidades de gestão de tráfego existentes na interface de saída (*egress*) do dispositivo e permite o estabelecimento de limites globais de largura de banda, tratando cada interface como uma classe.
- Suporte dos modelos *IntServ* (*Integrated Services* [10]) e *DiffServ* (nomeadamente, os PHBs EF [11] e AF [12]), da IETF.

A configuração de disciplinadores, classes e filtros é feita através da aplicação *tc*, que integra o utilitário *iproute2*. Este utilitário conta ainda com as aplicações *ip*, que permite a configuração estática ou baseada em políticas de endereçamento e encaminhamento, e *rtmon*, usada para a captura de estatísticas sobre o tráfego que atravessa o dispositivo de rede num determinado instante.

2.2 Suporte *DiffServ*

O Linux suporta o modelo *DiffServ* da IETF através da implementação das RFCs 2474, 2475, 2597 e 2598. O código para a implementação do *DiffServ* no Linux foi escrito por Werner Almesberger [13] e é suportado por um disciplinador específico, *sch_dsmark*, que extrai e marca o campo DSCP (*differentiated services code point*) dos pacotes, pelo classificador *cls_tcindex*, que utiliza a informação do campo DSCP para colocar o tráfego em classes de tráfego e pelo o disciplinador *sch_gred*, que suporta múltiplas prioridades de eliminação de pacotes e partilha de espaço de *buffer*.

Werner Almesberger mantém uma página sobre a implementação do *DiffServ* no Linux em [14].

2.3 *Netfilter*

Para além do subsistema de controlo de tráfego, o Linux apresenta ainda o *Netfilter* [15], uma versão particularmente interessante da pilha protocolar do Linux (IPv4, IPv6 e DECnet) que apresenta ganchos (*hooks*), ou pontos de interrupção no percurso dos pacotes de tráfego, onde podem ser acrescentados módulos do *kernel* para o processamento de pacotes de rede. Por exemplo, o *Netfilter* permite que, em certos pontos do caminho de um pacote pela pilha protocolar, exista um módulo que encaminhe o pacote para a interface de saída, elimine ou altere esse pacote ou o coloque numa fila para ser tratado em espaço de utilizador. Um destes módulos é o *iptables* [16], parte integrante do subsistema de *firewall* do Linux que permite a filtragem de pacotes (*stateless* e *stateful*), a tradução de endereços de rede (NAT, ou *Network Address Translation*) e a execução de funcionalidades genéricas de manipulação de pacotes, tal como, por exemplo, a alteração do campo DSCP dos pacotes IP.

O diagrama da Figura 1 mostra o percurso de um pacote pelo *kernel* do Linux, desde que chega à interface de entrada até que é entregue a um processo de aplicação ou novamente enviado para a rede, na interface de saída.

De acordo com o diagrama, os pacotes que atravessam o IPv4 podem ser interceptados em cinco ganchos diferentes: quando entram no dispositivo de rede, após a execução de testes de validação do pacote, e antes do código de encaminhamento (NF_IP_PRE_ROUTING); quando são destinados à máquina local e antes de serem passados a qualquer processo local (NF_IP_LOCAL_IN); quando são encaminhados para outro dispositivo de rede (NF_IP_FORWARD); depois do código de encaminhamento e imediatamente antes de serem novamente enviados para a rede, no caso de serem destinados a outra máquina (NF_IP_POST_ROUTING); e no caminho dos pacotes que são criados localmente (NF_IP_LOCAL_OUT).

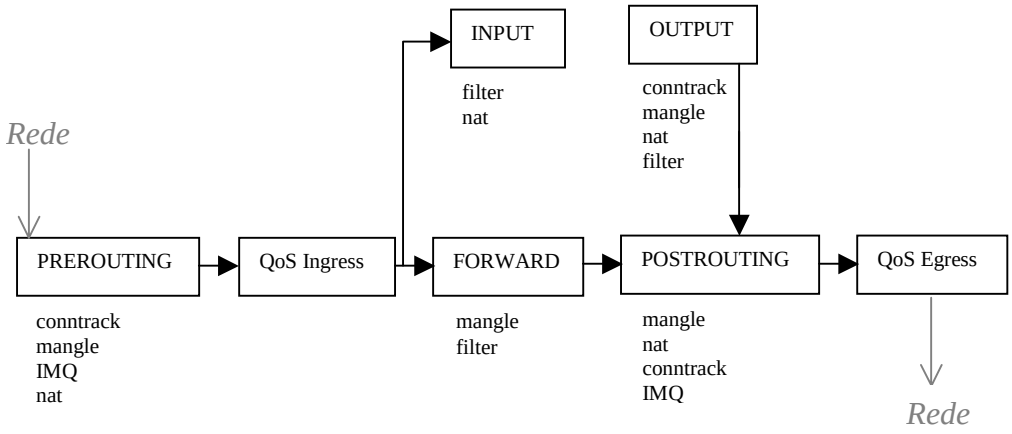

Figura 1 Percurso de um pacote pelo *kernel* do Linux

Como se pode ver pelo diagrama, existem dois momentos onde pode ser feita a diferenciação do tráfego e consequente gestão de largura de banda, o *QoS Ingress* e o *QoS Egress*. O *QoS Ingress* consiste na utilização de um disciplinador de ingresso, a *ingress qdisc*. Este disciplinador é relativamente limitado, não sendo utilizado para enviar pacotes para a rede mas antes para fazer o policiamento do tráfego que chega à interface. O policiamento é feito aplicando filtros *token bucket* aos pacotes que ingressam na interface, normalmente conjugados com os estimadores providenciados pelo *kernel*.

O *QoS Egress* acontece imediatamente antes de os pacotes serem enviados para a rede, na interface de saída, e permite configurações bastante mais complexas do que o disciplinador de ingresso, através da utilização de possíveis diferentes combinações de disciplinadores (por exemplo, CBQ, HTB e *dsmark*) e classes de tráfego. Ao contrário da *ingress qdisc*, que apenas permite o policiamento de tráfego, os disciplinadores utilizados no *QoS Egress* permitem a modelação de todo o tráfego de saída.

A conjugação do subsistema de controlo de tráfego com a arquitectura *netfilter* permite ainda uma situação bastante interessante a nível de modelação de tráfego, e que consiste na utilização do dispositivo virtual IMQ para a modelação de tráfego em dois pontos adicionais do percurso dos pacotes: no PREROUTING e no POSTROUTING (ver diagrama acima).

3. Caracterização do Edge Device

O dispositivo de fronteira descrito neste artigo está a ser desenvolvido sobre o *kernel* 2.4.18 do Linux. Este dispositivo interage com a LAN através da interface LAN_IFACE, sendo o acesso à Internet feito através da interface INET_IFACE.

As máquinas da LAN utilizam endereços privados (192.168.0.0/24), sendo feita SNAT (*source* NAT) para um endereço público no POSTROUTING. O tráfego Web é redireccionado de forma transparente para a aplicação Squid [17], no PREROUTING. O subsistema de *firewall* é realizado nos cinco *chains* pré-definidos (*hooks*) do Linux.

Em termos de tráfego que é importante caracterizar, distingue-se o tráfego TCP e UDP extremo-a-extremo, que apenas sofre o processo de NAT, o tráfego TCP quebrado, destinado a *gateways* de aplicação residentes no dispositivo de fronteira, com particular destaque para o tráfego HTTP, e ainda tráfego tradicionalmente difícil de classificar, como aquele gerado por aplicações que utilizam diversos portos estáticos e dinâmicos. O caso que analisamos neste artigo é o tráfego *streaming* que segue a norma H.323 da ITU.

Como foi já dito anteriormente, pretende-se fazer a gestão de largura de banda no acesso à Internet para tráfego de saída (*upstream*) e tráfego de ingresso (*downstream*). A diferenciação do tráfego deverá ser feita por utilizador, máquina, endereço IP e subrede. Para além da diferenciação e subsequente modelação do tráfego, pretende-se que o subsistema de controlo de tráfego faça a marcação do campo DSCP dos pacotes de saída, segundo os PHBs AF e EF.

A secção que se segue faz a análise das possíveis soluções para o cenário de controlo de tráfego aqui descrito. Pretende-se, tanto quanto possível, a obtenção de uma solução genérica e independente dos *gateways* de aplicação a utilizar.

4. Solução de Controlo de Tráfego

Esta secção apresenta um conjunto de questões que surgiram quando se faz controlo de largura de banda no acesso à Internet com diferenciação por endereço IP origem num dispositivo de fronteira que também assume funcionalidades de *firewall* e de *gateway* de aplicações. Para cada questão colocada, são exploradas as possíveis soluções oferecidas pelo Linux.

4.1 Tráfego *Upstream*

Uma das questões a ter em conta na análise da gestão do tráfego *upstream* é a utilização de *proxy* transparente para tráfego HTTP. O *kernel* 2.2 do Linux trazia suporte específico para *proxy* transparente. Basicamente, este suporte consistia em permitir o *bind* a *sockets* não locais, quando a opção CONFIG_IP_TRANSPROXY estava activa. O endereço destino original (endereço IP e porto) de uma destas ligações era conseguido através de uma chamada ao sistema *getsockname()*. No entanto, este suporte foi retirado a partir da versão 2.4 do *kernel*, havendo agora uma clara dissociação das funcionalidades de NAT relativamente ao código nuclear do TCP/IP. Tal facto resulta numa dificuldade evidente em diferenciar o tráfego HTTP que passa pelo Squid por endereço IP origem, no *QoS Egress*, uma vez que o endereço que vai ser visto pelo subsistema de controlo de tráfego vai ser o do próprio dispositivo de fronteira.[1]

Outra questão igualmente importante na diferenciação de tráfego *upstream* diz respeito à realização de SNAT no POSTROUTING. Ambas as questões são consideradas na análise que

[1] Balázs Scheidler, da empresa Balabit [18], desenvolveu um *patch*, o T-Proxy [19], que permite a implementação de uma solução integral de *transparent proxy*, baseando-se nas funcionalidades do *kernel* 2.2 referidas acima. Este *patch* permite abrir ligações locais com endereços externos e escutar ligações num endereço não local, através da introdução de alterações significativas ao *netfilter*, tais como a criação de uma tabela específica (*tproxy*) e a adição de um novo *target* (*TPROXY*). No entanto, actualmente apenas o *proxy* Zorp, também da Balabit, está preparado para utilizar este *patch*.

se segue, que explora as opções de modelação de tráfego de saída na interface INET e na interface LAN.

Modelação na Interface INET

Num dispositivo Linux dedicado ao controlo de tráfego, o local mais adequado à modelação do tráfego *upstream* é a interface de saída para o exterior (INET), no *QoS Egress*. No entanto, a utilização de NAT no POSTROUTING e de uma *cache* de conteúdos HTTP, assim como a necessidade de se fazer marcação *diffserv* nos pacotes de saída, fazem com que a modelação no *QoS Egress* não seja trivial.

A diferenciação e posterior modelação de tráfego HTTP por endereço IP origem na interface de saída é possível no cenário de operação actual, em que o tráfego HTTP *upstream* passa pelo Squid. Tal é conseguido da seguinte forma:

- O tráfego HTTP é redireccionado para o porto do Squid (por exemplo, 3128) no PREROUTING (1).
- O Squid abre uma ligação TCP para o servidor da página requerida, fazendo a marcação DSCP dos pacotes de saída segundo o endereço IP origem (utilizando o comando `tcp_outgoing_dscp` no ficheiro de configuração do squid, squid.conf).
- No *QoS Egress*, o tráfego é colocado em classes de tráfego específicas, de acordo com a marcação DSCP dos pacotes. O tráfego poderá ainda ser remarcado, através do disciplinador *dsmark* (2).

(1) `iptables -t nat -A PREROUTING –i $LAN_IFACE -p tcp --dport 80 –j REDIRECT --to-port 3128`
(2) `tc class change dev $INET_IFACE parent 1:0 classid 1:1 \ dsmark mask 0x3 value 0xb8` (exemplo de marcação do campo DSCP com o código EF).

Figura 2 Código para a modelação de tráfego HTTP *upstream* na interface INET

Embora possível, esta solução não é genérica. Outros *proxies* aplicacionais *open source*, tal como, por exemplo, a aplicação Frox para tráfego FTP puro[2], não possuem mecanismos de marcação do tráfego por DSCP ou ToS semelhantes ao do Squid, pelo que a marcação de tráfego FTP na *QoS Egress*, deste modo, não poderá ser feita por endereço IP origem.

Para o tráfego que apenas sofre a operação de NAT, a diferenciação no *QoS Egress* por endereço IP origem pode ser feita da seguinte forma:

- No *chain* FORWARD, os pacotes são marcados na tabela *mangle*[3] segundo o endereço IP origem (3).
- No *chain* POSTROUTING, é feito SNAT para o novo endereço. No entanto, a marca feita no ponto anterior mantém-se (4).
- No *QoS Egress*, o tráfego é classificado segundo a marca e colocado na classe *diffserv* correspondente (5).

[2] Aqui distingue-se tráfego FTP puro das implementações de FTP sobre HTTP, tal como é feito, por exemplo, no Squid.
[3] Cada *chain* do *netfilter* pode ter uma ou mais das seguintes tabelas pré-definidas: *filter*, *mangle* e *nat*.

```
(3) iptables -A FORWARD -t mangle -o $INET_IFACE -p tcp -s
   $IP1 –j  MARK --set-mark $IP1_MARK
(4) iptables –t nat –A POSTROUTING –s $IP1 –j SNAT --to-source
   $IP_NAT
(5) tc filter add dev $INET_IFACE protocol ip parent 1:0 \
   prio 1 handle $IP1 fw flowid 1:1
```

Figura 3 Código para a modelação de tráfego não HTTP *upstream* na interface INET

As características específicas das aplicações *streaming* fazem com que necessitem de ser analisadas à parte. Normalmente, estas aplicações requerem a utilização de *gateways* específicos, tal como o SOCKS [20], para manter o *streaming* das aplicações e actuar ao nível da autenticação e estabelecimento das ligações, necessário para aplicações cadenciadas.

Uma outra questão a considerar relaciona-se com a coexistência de aplicações *streaming* e das funcionalidades de tradução de endereços (NAT). Olhando para o exemplo específico de aplicações que utilizam a norma H.323, como é o caso do NetMeeting da Microsoft, verifica-se que estas aplicações utilizam os portos TCP fixos 389, 522, 1503, 1720 e 1731 e ainda portos atribuídos dinamicamente (TCP e UDP). De modo a suportar as operações de NAT e de *connection tracking* sobre os fluxos de dados requeridos nos portos dinâmicos, é necessário acrescentar código específico ao *kernel* do Linux. Este código existe actualmente sobre a forma de *patch*, o h323-conntrack-nat (ver [15]).[4]

A utilização deste *patch* permite que a manutenção dos fluxos de dados possa ser feita sem recorrer ao auxílio de um *gateway* de aplicação, pelo que a diferenciação deste tipo de tráfego por endereço IP pode ser reduzido ao caso mais simples da diferenciação de tráfego que apenas sofre a operação de NAT, já descrito em cima.

Modelação na Interface LAN

Uma alternativa à modelação na *QoS Egress* da interface INET consiste no redireccionamento do tráfego para um dispositivo IMQ, no PREROUTING. Desta forma, a modelação do tráfego é feita antes do tráfego ser redireccionado para os *proxies*. A modelação/classificação de tráfego é então feita da seguinte forma:

- O tráfego que ingressa na interface LAN é redireccionado para o IMQ, na tabela *mangle* do PREROUTING (1).
- No IMQ, o tráfego é modelado através da utilização de disciplinadores de egresso e é marcado o campo DSCP dos pacotes (2).
- Na tabela *nat* do PREROUTING, o tráfego HTTP é redireccionado para o Squid, onde o tráfego é novamente marcado com um código DSCP (3).
- O tráfego que não passa por qualquer *gateway* de aplicação mantém a marcação DSCP feita no ponto 1.

[4] Para versões anteriores ao *kernel* 2.4.20, é necessário instalar ainda o *patch newnat.* Este *patch* pode ser encontrado no patch-o-matic-20020825 [15].

```
(1) iptables -t mangle -A PREROUTING -i $LAN_IFACE -j IMQ
(2) tc class change dev $IMQ parent 1:0 classid 1:1 \
    dsmark mask 0x3 value 0x28 (exemplo de marcação do campo DSCP
    com o código AF 11)
(3) iptables -t nat -A PREROUTING –i $LAN_IFACE -p tcp --dport
    80 –j  REDIRECT --to-port 3128
```

Figura 4 Código para a modelação de tráfego *upstream* na interface LAN

Como conclusão, o controlo de largura de banda pode ser feito de uma forma genérica – antes de qualquer redireccionamento de tráfego para *proxies*, ou da execução de qualquer acção de NAT – com o redireccionamento do tráfego de entrada para o IMQ, no PREROUTING.

Com o presente estudo, ainda não foi encontrada uma solução genérica para a marcação DSCP dos pacotes que saem do dispositivo de fronteira. Por um lado, a marcação de tráfego HTTP é feita no Squid, implicando a adição de linhas específicas no ficheiro de configuração do Squid. Por outro lado, a marcação do tráfego FTP sobre o qual é feito *proxy* transparente não pode ser feita tendo em atenção o endereço IP origem. Finalmente, o restante tráfego é marcado de forma genérica, quer no *QoS Egress*, quer no IMQ.

4.2 Tráfego *Downstream*

Quando se faz o controlo de tráfego de ingresso, não é possível ter-se o controlo sobre o que é enviado da Internet para o nosso dispositivo de fronteira. Por exemplo, em caso de *flooding*, não adianta fazer o controlo de tráfego no ingresso, pois o tráfego já chegou ao *router* antes de se poder filtrá-lo.

No entanto, o controlo do tráfego proveniente da Internet pode permitir, por um lado, que o nosso dispositivo de fronteira seja o ponto de estrangulamento da rede, e não o *router* do ISP, e, por outro lado, que possamos ter algum mecanismo de controlo de tráfego TCP sobre o tráfego *downstream*.[5]

Existem três maneiras possíveis de se fazer modelação à entrada: na interface INET, através do disciplinador de ingresso, *ingress qdisc* (na verdade, está-se a fazer policiamento e não modelação); na interface INET ou na interface LAN, através do dispositivo virtual IMQ; e na fila de saída da interface LAN, no ponto *QoS Egress*.

Na Interface INET

Como vimos já anteriormente, a *ingress qdisc* coincide com o *QoS Ingress*, colocando-se depois do PREROUTING (i.e., depois do NAT ser resolvido) e imediatamente antes da decisão de encaminhamento. Isto faz com que o tráfego destinado a *proxies* e *gateways* de aplicação do dispositivo de fronteira não possa ser policiado tendo em conta o endereço IP destino real.

A *ingress qdisc* apresenta, contudo, outras limitações. Por um lado, não permite a diferenciação entre sub-fluxos, não garantindo, consequentemente, a partilha justa de largura de banda entre sub-fluxos.[6] Por outro lado, as estatísticas do *qdisc ingress* não funcionam na versão do *kernel* escolhida para este projecto, a 2.4.18.[7]

---

[5] A este propósito, refira-se que Patrick McHardy está actualmente a desenvolver um novo disciplinador capaz de implementar algoritmos de controlo de tráfego TCP (*TCP rate control*) [21][22].

[6] Ao contrário, a utilização de disciplinadores de egresso, tal como o SFQ (*Stochastic Fair Queuing*), permite a diferenciação de sub-fluxos e a posterior justa distribuição de largura de banda por esses sub-fluxos.

[7] Depois de alguma pesquisa, foi encontrado o *patch ing-stats* [23], que resolverá este problema. No entanto, este *pacth* não é directamente aplicável à distribuição utilizada neste projecto.

Uma alternativa à utilização do disciplinador de ingresso é o IMQ. No percurso de um pacote no *kernel*, o IMQ pode ocorrer no PREROUTING ou no POSTROUTING. No PREROUTING, o IMQ ocorre antes do NAT (DNAT), pelo que redireccionar os pacotes que entram na interface INET para o IMQ no PREROUTING se torna incompatível com a tradução de endereços. Ou seja, não é possível fazer a diferenciação de tráfego neste ponto por endereço IP destino. Outra solução passa pelo redireccionamento do tráfego para o IMQ no *chain* POSTROUTING. No entanto, o IMQ no POSTROUTING precede imediatamente o *QoS Egress*, pelo que, neste caso, a modelação do tráfego no IMQ em detrimento da modelação na *QoS Egress* acarreta o *overhead* da introdução de mais uma fila.

Patrick McHardy, o autor da versão actual do disciplinador IMQ, desenvolveu um *patch* que permite registar o IMQ depois do NAT, no PREROUTING. Este *patch*, *imqnat.diff* [24], soluciona a diferenciação de tráfego TCP e UDP extremo-a-entremo por endereço IP origem, para tráfego *downstream*.

Em termos comparativos, a utilização do IMQ (com o *imqnat.diff*) cobre todas as funcionalidades do disciplinador de ingresso desejadas para o nosso dispositivo de fronteira, para além de permitir uma verdadeira modelação através da associação de disciplinadores de saída (*egress qdisc*). No entanto, a utilização do dispositivo IMQ introduz um *overhead* associado à introdução de mais uma fila no caminho dos pacotes pelo *kernel*.

Relativamente ao caso específico da diferenciação do tráfego HTTP, podem ser consideradas as seguintes opções:

a. O tráfego HTTP não é redireccionado para a IMQ, sendo modelado no Squid através das *delay pools*.[8]

b. O tráfego HTTP é modelado à entrada (através da *ingress qdisc* ou do IMQ), sem ter em conta o endereço IP origem. Tal opção pode levar a que tráfego HTTP de utilizadores prioritários não receba a largura de banda de que necessita.

c. O tráfego HTTP é modelado na interface LAN, no *QoS Egress*. No entanto, esta opção tem a desvantagem de implicar a modelação de todo o tráfego HTTP, incluindo aquele que já se encontra em *cache*.

d. O tráfego HTTP de utilizadores prioritários não passa pelo Squid e, consequentemente, pode ser modelado à entrada, por endereço IP. Por outro lado, o restante tráfego HTTP cai na situação descrita no ponto b., acima. No entanto, esta opção conduz a que se percam os benefícios da *cache* de conteúdos para os utilizadores prioritários.

Na Interface LAN

A modelação na interface LAN é feita de modo semelhante ao tráfego *upstream*, na *QoS Egress*. A solução é perfeitamente compatível com o NAT e com o tráfego TCP quebrado, pois o *QoS Egress* é a última etapa antes do pacote ser enviado para a rede. No entanto, esta opção tem a desvantagem de modelar o tráfego HTTP que já se encontra em *cache*.

[8] As *delay pools* são um mecanismo que o Squid fornece para a limitação de largura de banda de certos pedidos baseada numa lista de critérios [25].

5. Avaliação da Solução

Para avaliar as funcionalidades e a interoperabilidade do subsistema de controlo de tráfego do nosso dispositivo de fronteira, foi criado um ambiente de testes ilustrado na Figura 5, e que se descreve de seguida.

O *testbed* é constituído por um dispositivo de fronteira (Edge Device) multi-serviço, com Linux 2.4.18, incluindo todas as funcionalidades de encaminhamento, *gateway* de aplicações, *firewall* e controlo de tráfego. Para a geração de tráfego de teste e recolha de resultados são utilizados quatro *endpoints*.

De modo a permitir as funcionalidades de controlo de tráfego, NAT e *firewall* foi activado o suporte do *netfilter* e do QoS no *kernel* do dispositivo de fronteira. De igual forma, foram instaladas as *packages* iproute2 e iptables-1.2.7a, e aplicados os *patches* para o IMQ, HTB, h323-conntrack-nat, assim como o já referido imqnat. Relativamente à *cache* de conteúdos *web*, foi instalado o Squid v.2.

Para a execução dos testes e análise dos resultados foram utilizadas as aplicações Chariot [26], Ethereal [27] e tcpdump, e ainda as estatísticas fornecidas pela aplicação *tc* do iproute2. Os *endpoints* Chariot EP1, EP2 e EP3 foram usados para simular máquinas na rede local. O *endpoint* EP4 foi usado para simular uma máquina na Internet (ver Figura 5).

Relativamente aos testes propriamente ditos, foram feitos testes de validação das seguintes situações:

- Integração das funcionalidades de NAT, IMQ e Squid na diferenciação de tráfego por endereço IP para o *upstream* e para o *downstream*.
- Marcação DCSP dos pacotes de saída tendo em conta o endereço IP.
- Redireccionamento selectivo de tráfego HTTP para o Squid por IP prioritário.

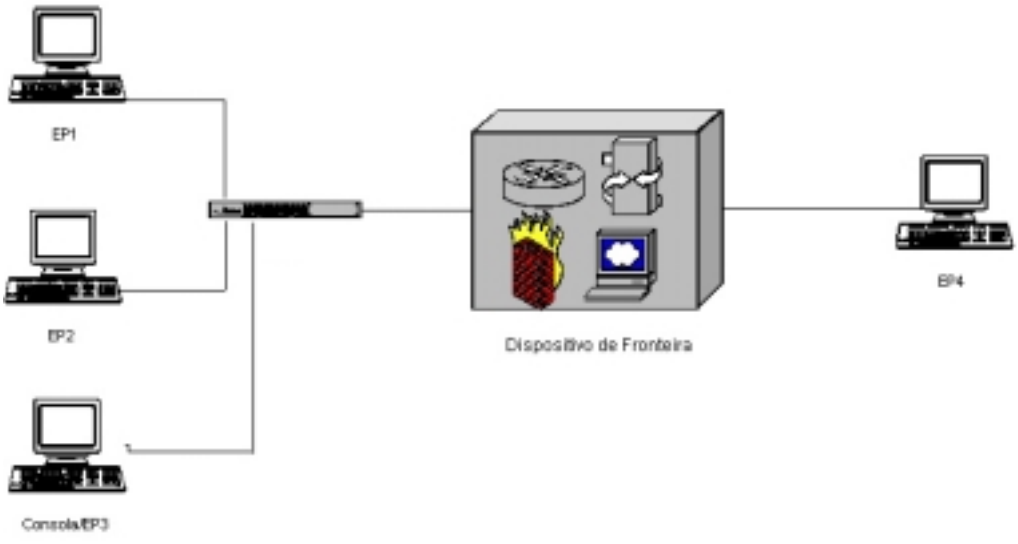

Figura 5 Ligação do dispositivo de fronteira e dos *endpoints*

A utilização das ferramentas tcpdump, Ethereal e das próprias estatísticas do *tc* permitiram mostrar que estamos na presença de uma solução equilibrada e funcional, que resolve os problemas de integração mencionados ao longo deste artigo.

Para além dos testes de validação, foi iniciada uma fase de realização de testes de eficácia às soluções descritas neste documento. Neste artigo, é apresentado um desses testes, assim como os primeiros resultados obtidos.

A solução que se pretendeu testar faz o controlo de tráfego *upstream* na interface INET – utilizando a hierarquia de disciplinadores dsmark, cbq e sfq – e o controlo do tráfego *downstream* na interface INET, através da utilização da interface virtual IMQ no PREROUTING, do *patch imqnat* e de uma hierarquia de disciplinadores cbq e sfq. Ainda segundo esta solução, o tráfego *upstream* é classificado em três possíveis classes de tráfego (U1, U2 e U3) e o tráfego *downstream* em duas (D1 e D2). Finalmente, o teste pretendeu simular o acesso à Internet numa linha assimétrica, disponibilizando 256kbps para o tráfego *upstream* e 1500kbps para o tráfego *downstream*. Esta simulação foi feita usando uma classe cbq em cada sentido, com o parâmetro *rate* correspondente aos valores indicados acima.

A tabela que se segue apresenta os fluxos de teste utilizados. Todos os fluxos gerados são TCP.

Par	***Endpoint 1***	***Endpoint 2***	**Direcção**	**Débito do Fluxo**
Pair 1	EP4	EP1	*Downstream*	256Kbps
Pair 2	EP4	EP3	*Downstream*	256Kbps
Pair 3	EP4	EP2	*Downstream*	256Kbps
Pair 4	EP2	EP4	*Upstream*	128Kbps
Pair 5	EP1	EP4	*Upstream*	128Kbps
Pair 6	EP3	EP4	*Upstream*	128Kbps

Tabela 1 Fluxos de teste

Numa primeira fase, os testes foram executados sobre um dispositivo de fronteira cujo único controlo de tráfego consistiu na limitação do débito de entrada e de saída nos 1500kbps e 256kbps já referidos. Os resultados dos testes feitos com a aplicação Chariot podem ser visualizados na Figura 6.

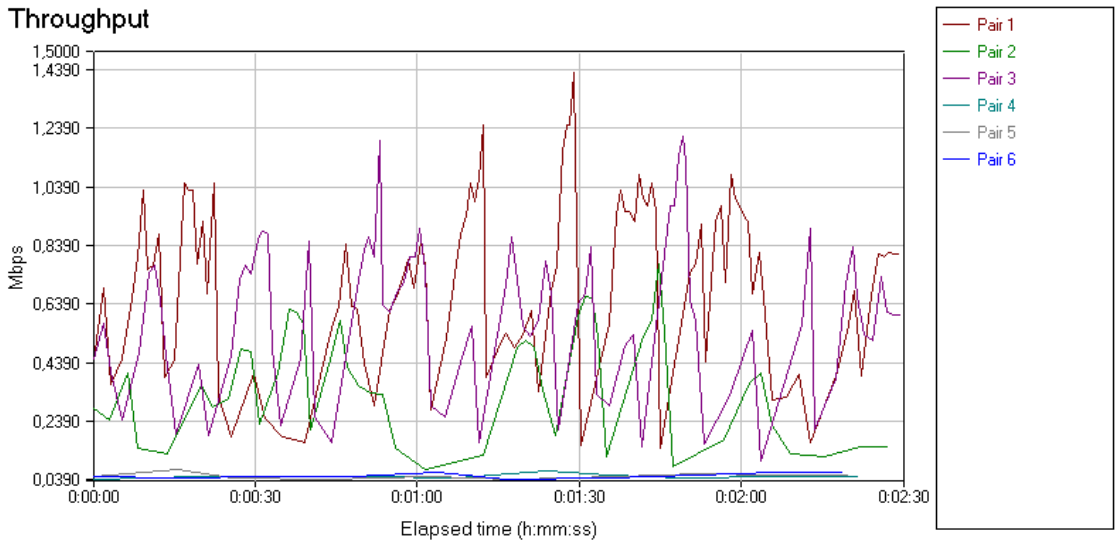

Figura 6 Resultados obtidos sobre o Edge Device sem controlo de tráfego

As oscilações que podemos ver no traçado de cada par são devidas ao mecanismo de controlo de janela do protocolo TCP.

Numa segunda fase, repetiram-se os testes descritos acima sobre um dispositivo de fronteira com controlo de tráfego, i.e., capaz de diferenciar o tráfego *upstream* e *downstream* em diferentes classes e de marcar o campo DSCP dos pacotes *upstream*. A definição das diferentes classes em termos de largura de banda e prioridade e a correspondência entre estas classes e os fluxos de teste podem ser observadas na Tabela 2.

Classe	**Par**	**Direcção**	**Descrição (Largura de banda, Prioridade)**
D1	*Pair* 1, *Pair* 2	*Downstream*	600Kbps, Prio1
D2	*Pair* 3	*Downstream*	800Kbps, Prio1
U1	*Pair* 4	*Upstream*	100Kbps, Prio1
U2	*Pair* 5	*Upstream*	100Kbps, Prio2
U3	*Pair* 6	*Upstream*	56Kbps, Prio3

Tabela 2 Correspondência dos fluxos de teste com as classes de tráfego

Com esta configuração, pretendeu-se que tráfego *downstream* não prioritário caísse numa classe genérica (*Pair* 1 e *Pair* 2 na classe D1), protegendo o tráfego proveniente de endereços IP prioritários (neste caso, *Pair* 3 e classe D2), o mesmo se passando relativamente ao tráfego *upstream* (com o tráfego crítico *upstream* a corresponder ao *Pair* 4).

Os resultados dos testes da segunda fase são apresentados na Figura 7.

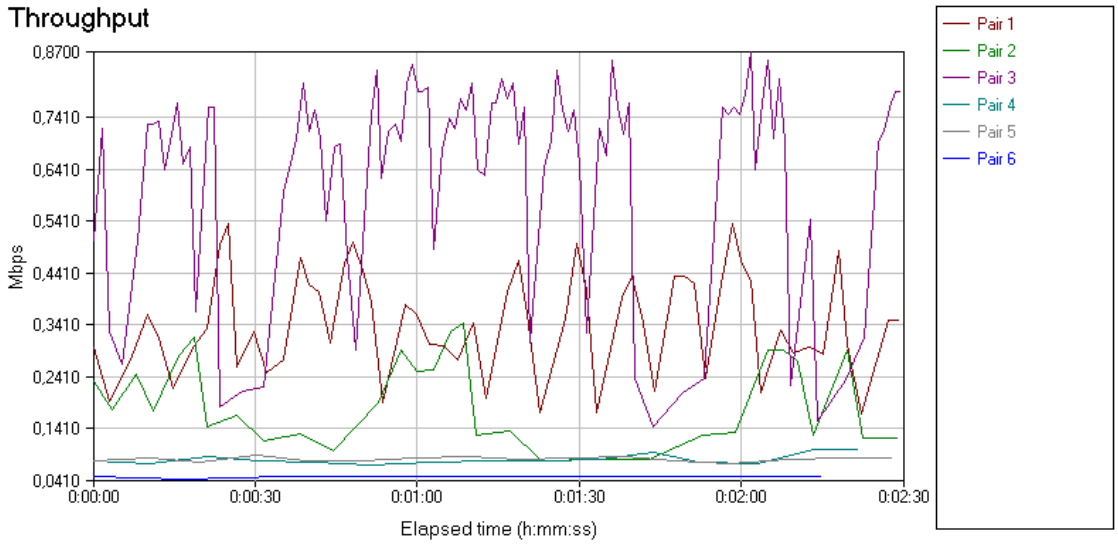

Figura 7 Resultados obtidos sobre o Edge Device com controlo de tráfego

Comparando as figuras 6 e 7 acima, verifica-se a eficácia da solução adoptada. Na direcção *downstream*, o tráfego do fluxo *Pair 3* consegue um débito médio de 0,450 Mbps na situação de ausência de diferenciação, atingindo os 0,532 Mbps quando colocado numa classe prioritária.

Relativamente ao tráfego *upstream*, verifica-se uma melhoria do fluxo mais prioritário (*Pair* 4) quando se passa da situação de ausência de controlo de tráfego para a situação de diferenciação no dispositivo de fronteira (0,051Mbps e 0,079Mbps, respectivamente), e uma degradação no débito do fluxo menos prioritário (*Pair* 6, classe U3). A diferenciação do tráfego *upstream* em três classes com diferentes prioridades é igualmente evidenciada pelas próprias estatísticas do *tc*, em que as classes mais prioritárias são capazes de enviar mais pacotes do que as classes menos prioritárias (Figura 8).

```
class cbq 2:100 parent 2:1 leaf 100: rate 100Kbit prio 1
 Sent 1908953 bytes 5378 pkts (dropped 0, overlimits 18942)
  borrowed 1304 overactions 2231 avgidle 624 undertime 0
class cbq 2:200 parent 2:1 leaf 200: rate 100Kbit prio 2
 Sent 1956796 bytes 3662 pkts (dropped 0, overlimits 17263)
  borrowed 182 overactions 2074 avgidle 624 undertime 0
class cbq 2:300 parent 2:1 leaf 300: rate 56Kbit prio 3
 Sent 1119997 bytes 2026 pkts (dropped 0, overlimits 16647)
  borrowed 34 overactions 1412 avgidle 624 undertime 0
```

Figura 8 Diferenciação do tráfego *upstream* (estatísticas do *tc*). As classes 2:100, 2:200 e 2:300 correspondem às classes U1, U2 e U3, respectivamente.

6. Conclusão e Trabalho Futuro

A integração de controlo de tráfego, NAT, *firewall* e *gateways* de aplicação num mesmo dispositivo Linux não é trivial. Este artigo focou alguns dos principais problemas associados a esta integração e apresentou soluções para a modelação de tráfego de saída (*upstream*) e de entrada (*downstream*) num dispositivo multi-serviço. Testes de integração e interoperabilidade mostraram que as soluções apresentadas são viáveis e funcionais.

Neste artigo, foram ainda apresentados alguns testes de eficácia efectuados a uma solução específica de controlo de tráfego a ser implementada no dispositivo de fronteira, tendo sido utilizados nesses testes fluxos do tipo TCP. Resultados preliminares mostraram a eficácia da solução integrada.

A próxima fase deste projecto é dedicada à realização de testes exaustivos à eficácia e ao desempenho do subsistema de controlo de tráfego, com vista à afinação dos *script* de controlo de tráfego e de *firewall* do dispositivo de fronteira. Nessa fase, ir-se-ão realizar outros tipos de testes, onde se incluem a utilização de fluxos UDP (que permitem uma melhor afinação do *script* de controlo de tráfego, uma vez que os resultados obtidos com fluxos TCP são mascarados pelo efeito da janela de congestão do protocolo), a experimentação de outros disciplinadores (nomeadamente, a substituição do disciplinador cbq pelo disciplinador htb) e a reconfiguração do ambiente de testes em termos de *endpoints*.

Por fim, após a fase de testes de eficácia e desempenho, proceder-se-á à integração do subsistema de controlo de tráfego com os restantes serviços do dispositivo de fronteira e à posterior realização dos testes de integração.

Agradecimentos

Este trabalho foi parcialmente financiado pela Agência de Inovação, no âmbito do projecto EDGEDEVICE.

Referências

[1] 6WIND, http://www.6wind.com.

[2] Lucent Technologies, http://www.lucent.com.

[3] Juniper Networks, http://www.juniper.net.

[4] S. Blake et al., “An Architecture for Differentiated Services Framework”, RFC 2475, http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2475.html.

[5] E. Rosen, A. Viswanathan, R. Callon, RFC 3031, “Multiprotocol Label Switching Architecture“.

[6] Laboratório de Comunicações e Telemática da Universidade de Coimbra, http://lct.dei.uc.pt.

[7] Critical Software, http://www.criticalsoftware.com.

[8] Linux Advanced Routing & Traffic Control Homepage, http://www.lartc.org/.

[9] The Intermediate Queuing Device, http://luxik.cdi.cz/~patrick/imq/.

[10] R. Braden , D. Clark, S. Shenker , “Integrated Services in the Internet Architecture: an Overview”, RFC 1633, http://www.faqs.org/rfcs/rfc1633.html.

[11] V. Jacobson et al., “An Expedited Forwarding PHB”, RFC 2598, http://www.faqs.org/rfcs/rfc2598.html.

[12] J. Heinanen, “Assured Forwarding PHB”, RFC 2597, http://www.faqs.org/rfcs/rfc2597.html.

[13] W. Almesberger, J. H. Salim, A. Kuznetsov, “Differentiated Services on Linux” <draft-almesberger-wajhak-diffserv-linux-01.txt>, http://www.globecom.net/ietf/draft/draft-almesberger-wajhak-diffserv-linux-01.html.

[14] Differentiated Services on Linux, http://diffserv.sourceforge.net/.

[15] Netfilter, http://www.netfilter.org.

[16] O. Andreasson, “Iptables Tutorial 1.1.18”, http://iptables-tutorial.frozentux.net/iptables-tutorial.html.

[17] Squid, http://squid-cache.org.

[18] Balabit, http://www.balabit.hu.

[19] TPROXY, http://www.balabit.hu/en/downloads/tproxy.

[20] SOCKS, http://www.socks.permeo.com.

[21] Arquivo da *mailing list lartc*, http://www.spinics.net/lists/lartc/msg04082.html.

[22] Arquivo da *mailing list lartc*, http://mailman.ds9a.nl/pipermail/lartc/2003q1/007785.html.

[23] *Patch ing-stats*, http://www.cyberus.ca/~hadi/patches/ing-stats.patch.

[24] *Patch imqnat.diff*, http://mailman.ds9a.nl/pipermail/lartc/2002q3/004725.html.

[25] Delay Pools, http://www.squid-cache.org/Doc/FAQ/FAQ-19.html.

[26] Chariot, http://www.netiq.com/products/chr/default.asp.

[27] Ethereal, http://www.ethereal.com.