Flow Manager

Luís Manuel Pinto da Silva

Orientador FEUP: Prof. José Ruela
Orientador INESC: Jorge Mamede

Universidade do Porto
Faculdade de Engenharia
FEUP

INESCPORTO
INSTITUTO DE ENGENHARIA DE SISTEMAS
E COMPUTADORES DO PORTO

Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Electrotécnica e de Computadores

Julho de 2006

Flow Manager

Luís Manuel Pinto da Silva

Orientador FEUP: Prof. José Ruela
Orientador INESC: Jorge Mamede

Trabalho realizado no âmbito do projecto de fim de curso, da
Licenciatura em Eng. Electrotécnica e de Computação da
Faculdade de Engenharia da Universidade do Porto.

Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Electrotécnica e de Computadores

Julho de 2006

Ciência.Inovação 2010 Programa Operacional Ciência e Inovação 2010
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR

Resumo

Um router é, hoje em dia, algo comum nas nossas casas. Com o aumento da largura de banda das ligações, com a massificação do wireless nas nossas casas, torna-se comum a partilha da ligação entre multiplos utilizadores. Mas o que acontece quando algum utilizador começa a fazer o download de um grande ficheiro, daquele servidor rapidíssimo? Invariavelmente o acesso fica quase interdito aos outros utilizadores.

Este projecto apresenta soluções no sentido de resolver esses problemas, dando ferramentas e técnicas que permitem o controlo tanto do upload como do download. Permitem também o balanceamento de carga, ou seja, a divisão do tráfego dos utilizadores por duas ou mais ligações a ISP's.

Agradecimentos

Quero agradecer aos meus orientadores do INESC, Jorge Mamede e Rui Moreira, pela boa orientação e também pelas constantes e saudáveis discussões.

Um também especial agradecimento ao Prof. Ruela, pelos ensinamentos transmitidos e pelas sempre pertinentes questões.

Aos meus colegas Bruno e Vítor. Pelo bom trabalho desenvolvido e constante coordenação.

Aos meus pais que sempre acreditaram em mim.

À Marta. Pela força, amor e carinho que sempre me transmitiu.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Enquadramento

Este projecto desenvolve um conjunto de ferramentas para ser usadas num router Linux.

1.2 Motivação

Com este trabalho será possível implementar novas funcionalidades num router. A divisão da largura de banda será feito de um modo inovador e ao mesmo tempo simples para quem o usa.

1.3 Objectivos

Os objectivos deste trabalho são so seguintes:

- Controlar o upload e o download de um conjunto de utilizadores.
- Manter disponível, o uso total da largura de banda.
- Divisão diferenciada da largura de banda agrupando utilizadores em classes de serviço.
- Divisão da largura de banda tendo em conta os utilizadores presentes.
- Distribuir o tráfego por diversas ligações a ISP's.

1.4 Estrutura do relatório

Este relatório divide-se em 6 capítulos. Nos capítulos 4 e 5, antes do trabalho do autor precede uma introdução teórica sobre temas ou ferramentas usadas.

No apêndice encontra-se o código das ferramentas implementadas.

Capítulo 2

Descrição do problema

2.1 Caracterização do cenário

Na figura 2.1 está exemplificado o cenário. Neste cenário temos duas ligações à Internet, através do ISP1 e do ISP2. Temos também três utilizadores ligados à gatebox.

2.2 Funcionalidades de um router

Um router é um equipamento que, estando situado na camada 3 do modelo OSI[1], permite que dois computadores, situados em redes diferentes, possam comunicar. Podem até permitir comunicação entre tipos de rede diferentes. Basicamente, o router escolhe o melhor caminho possível para os pacotes recebidos. Para tal, ele mantém tabelas de rotas usando processos

[1] Open Systems Interconnection

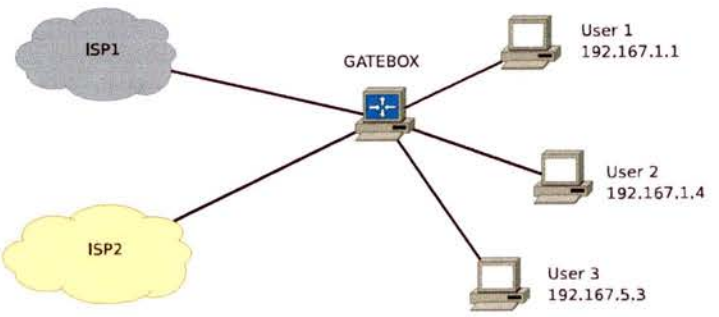

Figura 2.1: Cenário de implementação

e protocolos de actualização de rotas. É comum o router ter uma saída por defeito, para a qual vão todos os pacotes que o router não tenha já uma rota definida.

2.3 Identificação de requisitos

No âmbito deste trabalho é pedido que haja uma agregação do tráfego por utilizador. Cada utilizador pertencerá a uma classe. Cada classe representa diferentes larguras de banda e níveis de prioridade. No entanto, e ao contrário no que normalmente acontece, a largura de banda disponível pode sempre ser utilizada por qualquer dos utilizadores, fazendo com que neste caso, a divisão por largura de banda ocorra apenas quando vários utilizadores comunicam em simultâneo.

No cenário da figura 2.1 estão ligados três utilizadores, no entanto este número pode ser bem maior. Desta forma, para que não haja saturação da ligação externa, poderá ser necessário mais que uma ligação ao exterior. Claro que tudo isto depende do número de utilizadores e/ou da velocidade de acesso das ligações.

2.4 Arquitectura do sistema

O esquema deste sistema está descrito na figura 2.4. O sistema consiste num servidor PPPoE responsável pela autenticação, recorrendo a um servidor Radius. Neste servidor Radius encontra-se a informação do IP a ser entregue ao utilizador. O Route Manager é um componente desenvolvido neste trabalho que, obtendo periodicamente informação da tabela de routing do sistema, vai gerindo a mesma e ao mesmo tempo controlando os dois Flow Manager. No Flow Manager Upload é, tal como o nome faz crer, controlado o upload. No Flow Manager Download é controlado o download. A configuração, instalação e manutenção do servidor PPPoE e da base de dados Radius é feita no âmbito de um outro trabalho [OS06].

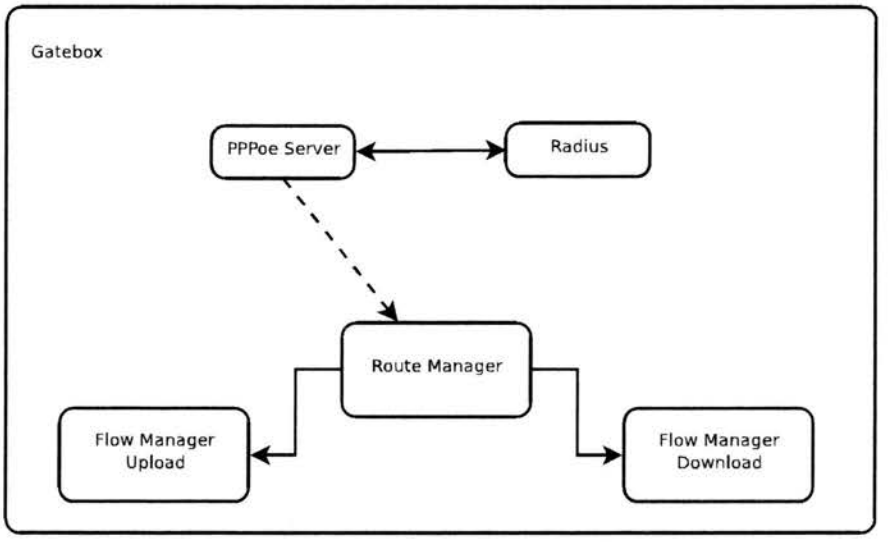

Figura 2.2: Arquitectura do sistema

Capítulo 3

Flow Manager

3.1 Introdução teórica

3.1.1 Filas para controlo de largura de banda

No Linux, desde a versão 2.2, existem ferramentas que permitem controlar a largura de banda. Estas ferramentas são filtros e filas.

Para controlar os filtros e filas do kernel, existe o comando *tc*. Este comando comunica directamente com o kernel, tal como está exemplificado na figura 3.1.1. Para informação detalhada sobre o uso do *tc* consultar [HGM+06]

Dentro do kernel

No kernel, cada interface possui dois nós. O ingress de entrada e o egress de saída. As filas, no entanto, só podem ser colocadas no nó de saída (egress).

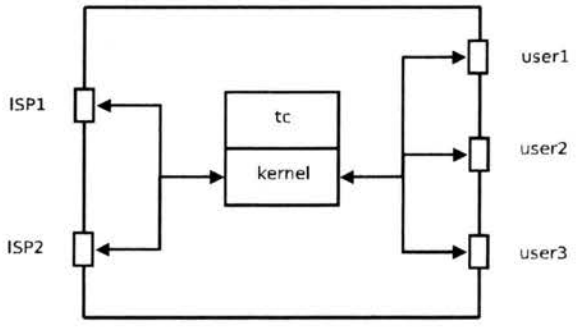

Figura 3.1: Modo de funcionamento do tc

Desta forma só podemos controlar o que enviamos.

A forma de o kernel despachar os pacotes de um interface tem dois passos distintos. Em primeiro lugar o kernel coloca os pacotes no nó egress do interface. Assim que o interface estiver pronto para enviar um pacote, o kernel pede ao egress um pacote para enviar. Este funcionamento leva a que se possa ter um controlo da forma como os pacotes são enviados, pois um pacote recebido pode ser enviado antes de outros pacotes que já estavam à espera. Quer isto dizer que o kernel não conhece a forma da fila nem o modo como ela se processa. Ele só se preocupa em enviar e receber os pacotes através deste nó.

Existem dois tipos de filas. As filas com classes e as filas sem classes. As classes não são mais que filas internas na fila principal, mas às quais é possibilitado o acesso para ser possível a colocação de outro tipo de fila ou para a colocação de pacotes usando os filtros disponíveis. Desta forma, numa fila com classes, podemos criar árvores de modo a dar prioridades diferentes aos pacotes. Na figura 3.1.1 está esquematizado a forma como os pacotes são escalonados.

Na Internet, nunca é possível saber à priori o tráfego que as pessoas vão gerar. Desta forma, podemos apenas controlar aquilo que enviamos, mas não conseguimos controlar aquilo que nos chega. Para poder controlar o download dos utilizadores, temos de criar filas associadas à interface que envia os pacotes, ou seja, ao upload através do router.

No âmbito deste projecto são usadas dois tipos de filas. Uma sem classes, que é a Stochastic Fairness Queueing (SFQ) e outra com classes que é a Hierarchical Token Bucket (HTB).

Stochastic Fairness Queueing

Este tipo de filas não apresenta classes. É sequer impossível anexar outra fila. É muito usada como folhas de uma árvore pois permite dar igualdade de tratamento, ou seja, igual largura de banda, aos diversos fluxos.

Internamente, esta fila cria um número fixo de filas. Depois possui um algoritmo de hash que vai distribuindo os pacotes pelas filas. Sequencialmente vai tirando um pacote de cada fila. Ou seja, devido ao número finito de filas, a igualdade podia estar comprometida, pois o algoritmo de hash podia colocar os vários pacotes de um fluxo nas várias filas e, desta forma, o fluxo tomava conta da largura de banda. Para prevenir isto, frequentemente é mudado o algoritmo de hash, para garantir uma maior justiça entre os fluxos. Alias, isto é um parâmetro de entrada na configuração da fila: tempo em segundos entre mudanças no algoritmo de hash.

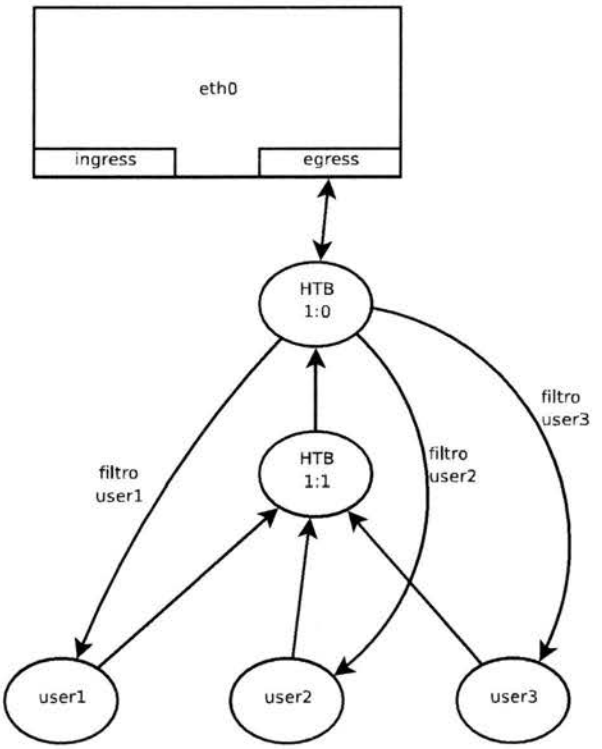

Figura 3.2: Ciclo dos pacotes nas filas

Hierarchical Token Bucket

Este tipo de fila é, tal como disse anteriormente, um tipo de fila com classes. Isto significa que, após inicializar a fila, é possível criar uma árvore de classes. É como colocar filas debaixo de filas. Depois, usando filtros, é possível seleccionar o tipo de pacotes que vão para uma determinada fila. Em cada nó desta árvore pode haver uma filtragem em que os pacotes que lá chegam colocados por outro filtro podem ser enviados para qualquer outro nó, pai ou filho. Existe um nó por defeito, para o qual vão todos os pacotes que não sejam sujeitos a filtragem.

O algoritmo de controle de largura de banda desta fila é o Token Bucket. O seu funcionamento baseia-se num balde com tokens. A cada token está associado uma unidade de bytes. Quando um nó quer transmitir, verifica se tem tokens para o fazer. Se tiver, pode enviar a quantidade de bytes relativa ao número de tokens presentes no balde. Se não tiver tokens, não transmite. Periodicamente, e consoante a largura de banda do nó, vão sendo adicionados tokens. Quando o balde está cheio, os tokens são que são criados são descartados.

É possível também emprestar largura de banda que não seja usada por um nó a outro nó. Isto só acontece quando os nós são filhos do mesmo pai. No caso de serem filhos do nó principal da fila, a largura de banda também não é emprestada. Como tal, é normal colocar uma classe, filho único, debaixo do nó principal a controlar a largura de banda da fila e de outras filas subsequentes.

Na figura 3.1.1 a árvore HTB do cenário de implementação apresentado na figura 2.1. Cada nó possui um identificador. A primeira parte do identificador, no caso do HTB é referente ao identificador da fila. A segunda parte é o identificador da classe. A classe, tal como foi descrito no capitulo 3.1.1 é uma fila interna. Tomando como exemplo o nó 1:503, o identificador 503:0 não tem que corresponder ao identificador da classe. No entanto, para uma melhor organização e percepção da estrutura é o que se faz.

Continuando com o exemplo da figura 3.1.1 vou explicar o modo como é feito o empréstimo de largura de banda. Neste caso só os nós 1:101, 1:104 e 1:503 podem partilhar largura de banda entre si. A distribuição da largura de banda é feito de uma forma hierárquica, dos pais para os filhos. Ou seja, começamos pelo nó principal do HTB (1:0). Neste nó não é definida a largura de banda. Como tal, a largura de banda é definida no nó 1:1. A soma da largura de banda dos seus filhos nunca ultrapassa a sua largura de banda. Já os nós 1:101 , 1:104 e 1:503 podem emprestar largura de banda entre si. Um nó dá aos seus filhos toda a largura de banda que conseguir, ou seja, a definida pelo nó mais a que conseguir emprestado de outros nós. A banda

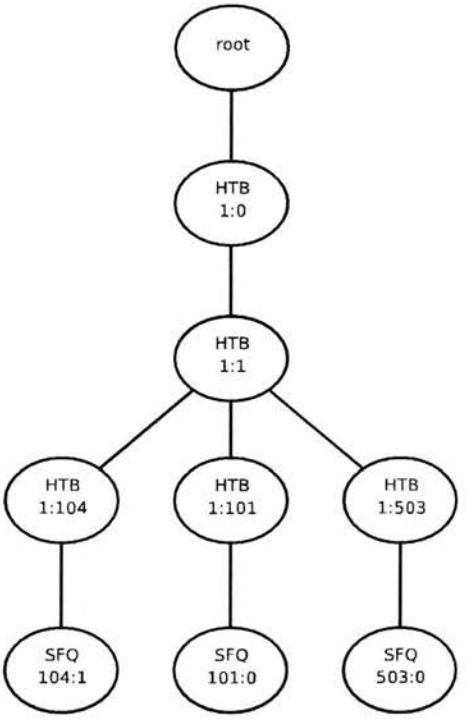

Figura 3.3: HTB e SFQ

que se pode pedir emprestada é também definida.

Além da largura de banda, podemos ainda definir níveis de prioridade. Existem oito níveis de prioridade que vão do zero ao sete, sendo o nível zero o de maior prioridade. Esta prioridade é relativa, ou seja, é apenas comparada com os nós irmãos. Desta forma, se um nó tiver um nível de prioridade inferior, enquanto que não ultrapassar a banda para si definida, os pacotes são logo enviados. Seguidamente são enviados os pacotes correspondentes ao nível de prioridade seguinte. Se pensarmos que o HTB percorre a árvore de cada vez que vai enviar um pacote, o que acontece é que os nós com mais baixa prioridade são visitados em primeiro lugar.

3.1.2 Filtros

Para que os pacotes cheguem à fila correcta é necessário que se proceda a uma filtragem. O método mais básico e ao mesmo tempo mais poderoso é o conjunto de filtros u8 u16 e u32. Estes filtros permitem uma filtragem ao nível do byte, correspondendo o número após o u ao número de bits que vão ser filtrados. com o u32 podemos, colocando-nos no local do IP de origem filtrar os pacotes por esse campo. O mesmo acontece para o IP de destino. Para um estudo aprofundado deste e de outro tipo de filtros consulte [HGM+06]

3.2 Implementação

Foi implementado em C++ um programa para controlar dinamicamente as filas e filtros de modo a separar o tráfego de cada utilizador. Foi dado o nome de flowmanager pois é através dele que se vai controlar o fluxo dos utilizadores.

3.2.1 Configuração

A configuração do flowmanager permite escolher qual o interface onde vai ser feita o controle de largura de banda. Permite também descrever o número de classes e os seus parâmetros, ou seja: o peso e a prioridade. O peso descreve a forma como é dividida a largura de banda e vai ser explicado no capitulo 3.2.2. A prioridade faz com que os pacotes de uma classe tenham prioridade relativamente aos pacotes de outra classe. Este é um exemplo do ficheiro de configuração:

```
CLASSES 5
MAXRATE 256 #In kbit
IFNAME eth0
```

```
PESOS 1 2 3 4 5
PRIO 5 4 3 2 1
DST
```

Tudo o que esteja a seguir a # são comentários. É necessário manter a ordem das opções do ficheiro de configuração.

CLASSES Corresponde ao número de classes presente. As classes são mapeadas no IP do cliente. ou seja. um cliente com o IP xxx.xxx.1.xxx será de classe 1. Como neste caso temos 5 classes, a nossa gama de endereços situa-se entre o xxx.xxx.1.xxx e o xxx.xxx.5.xxx.

MAXRATE Corresponde à largura de banda para a qual o interface vai ser limitado. Convém colocar este limiar abaixo da velocidade da ligação. para que os pacotes enviados pela gatebox não fiquem na cache do modem e, desta forma, passava a ser o modem a controlar o upload. o que não é desejável. Além de que iríamos perder interactividade. Isto aplica-se no caso do upload, pois o download é mais complexo e será explicado mais tarde.

IFNAME Corresponde ao interface de rede que pretendemos controlar. Desta forma podemos ter mais que um flowmanager a correr controlando qualquer interface.

PESOS Define o peso que cada classe tem na divisão da largura de banda. O número de pesos tem que ser igual ao número de classes definido anteriormente. Na secção 3.2.2 vai ser explicado como a largura de banda é utilizada pelos utilizadores.

PRIO Define a prioridade de cada classe. Utilizadores da mesma classe vão receber igual prioridade. Também neste caso, o número de campos tem que ser igual ao número de classes definido anteriormente.

DST Opcional. Por defeito a filtragem é feita com base no IP de origem. Quando colocado faz com que o flowmanager faça a filtragem dos utilizadores com base no IP de destino. É colocada quanto o flowmanager faz escalonamento no download.

3.2.2 Divisão da largura de banda

A largura de banda vai ser dividida com base em dois factores. Os pesos das classes e o número de utilizadores presentemente ligados. A largura de banda de um utilizador pode ser calculada da seguinte forma.

Assumindo que T é o peso total do sistema, k é o número da classe, L o número de classes e que N_k é o número de utilizadores presentemente na classe k:

$$T = \Sigma_{k=1}^{L} Peso_k \times N_k \tag{3.1}$$

Sendo BWT a largura de banda disponível no interface, ou seja, aquela que foi configurada no ficheiro de configuração, a largura de banda do cliente BW_c será então:

$$BW_c = \frac{Peso_k}{T} \times BWT \tag{3.2}$$

Desta forma um cliente manterá uma largura de banda proporcional ao peso que a sua classe possui e ao número de utilizadores presente, mantendo sempre toda a largura de banda atribuída. Sempre que der entrada um novo utilizador, os cálculos são refeitos. Mais tarde falaremos na forma de entrada e saída dos utilizadores.

3.2.3 Disciplina de filas usada

Neste projecto foram usadas as filas HTB e as filas SFQ.

As filas SFQ, tal como já foi explicado no capitulo 3.1.1 são colocadas nas folhas da árvore HTB. Desta forma damos ao utilizador uma maior interactividade pois, mesmo tendo um fluxo a ocupar a sua largura de banda, se for enviado um pedido, este será rapidamente enviado. Com a cada vez maior utilização da tecnologias P2P[1] isto torna-se fundamental, pois a utilização total da largura de banda de upload levaria a DoS[2] de outros fluxos.

O uso das filas HTB prende-se com os requisitos impostos. É necessário que a largura de banda esteja totalmente disponível. Ora, no HTB, com a particularidade de "emprestar"largura de banda, se a quantidade máxima que pode ser emprestada for igual ao máximo da ligação, garantimos que, independentemente do número de utilizadores ligados, se apenas um deles estiver a transmitir, poderá fazê-lo ao máximo que a ligação permite. Como já foi explicado no capitulo 3.1.1, para que seja possível emprestar largura de banda, os nós têm que ser todos filhos do mesmo pai. Ora é isso mesmo que se passa.

[1] Peer to peer

[2] Denial of Service

A organização das filas é a demonstrada pela imagem 3.1.1. Cada nó filho do 1:1 corresponde a um utilizador. A identificação da classe é correspondente ao IP usado. Deste modo, facilmente se consegue associar o IP do utilizador à sua fila. Pegando por exemplo na fila do com identificador 1:503. Significa que é da classe 5 e que pertence ao utilizador 3 com IP 192.167.5.3. No entanto o valor 5 e 3 estão em hexadecimal. Se o utilizador tivesse o ip 192.167.5.15 a sua fila teria o id 1:50F. Desta forma, o número máximo de classes que um sistema controlado pelo flowmanager poderá ter é F, que corresponde a 15. Dentro de cada classe teremos no máximo 256 utilizadores. Este número de utilizadores pode parecer limitador mas, atendendo que provavelmente existirão poucas classes, podemos criar mais classes com o mesmo peso e prioridade, duplicando logo o número de utilizadores com a mesma prioridade e largura de banda.

Para que tudo isto funcione correctamente é necessário que haja uma filtragem dos pacotes para a fila correcta. Dado que é o IP do utilizador que o identifica, torna-se fácil colocar os pacotes do utilizador na fila correcta. Para tal, foi utilizado o filtro u32 que, tal como é dito em 3.1.2, é um comparador ao nível do byte. Nos filtros existe também um identificador. Por defeito o identificador de filtro é o 8000: sendo a outra parte do identificador idêntica ao já feito nas classes.

3.2.4 API

#flowmanager ID -OPT [DATA]

ID É o identificador do flowmanager. Para que seja possível ter duas instâncias do flowmanager a correr é necessário que os ID's sejam diferentes.

OPT	DATA	Descrição
A	IP do utilizador	Adiciona um utilizador.
D	IP do utilizador	Remove um utilizador.
L		Lista os utilizadores presentes.
I	Ficheiro de configuração	Inicializa o flowmanager.
S		Para o flowmanager.

Tabela 3.1: API do flowmanager

3.2.5 Modo de funcionamento

O flowmanager é um programa do tipo cliente servidor. Como tal, primeiro temos que iniciar o servidor designando um ID. Este ID é essencial para o cliente saber com qual servidor comunicar. O ID pode conter números ou letras. Neste caso definimos que o ID corresponde ao nome do interface presente no ficheiro de configuração usado na secção 3.2.1, ou seja, eth0. Pressupomos que o ficheiro de configuração se encontra no directório actual e que se chama eth0.conf. Então o comando para iniciar o servidor seria:

#flowmanager eth0 -I eth0.conf

Vamos supor que, seguidamente, entraria um utilizador. É necessário informar o flowmanager que chegou um utilizador, por forma a que possa criar as filas e os filtros para o tráfego do utilizador. Imaginemos que o IP atribuído ao utilizador era 192.167.5.3. Era portanto um utilizador de classe 5. O comando a executar era então:

#flowmanager eth0 -A 192.167.5.3

Se porventura o utilizador saísse, tinha-se que dizer ao flowmanager para o retirar, pois quanto maior o número de filas presentes, mais tempo de processamento é pedido ao kernel para fazer o escalonamento dos pacotes. Desta forma, quanto menos filas estiverem melhor. Neste comando o flowmanager retira em primeiro lugar o filtro que coloca os pacotes na fila e só depois retira a classe HTB correspondente. a fila SFQ que está anexa à fila SFQ. por deixar de ter ligação ao nó principal do interface, é automaticamente removida.

#flowmanager eth0 -D 192.167.5.3

Se quizessemos ter informação dos utilizadores presentes no flowmanager utilizaríamos o seguinte comando:

#flowmanager eth0 -L

Para parar o servidor bastaria utilizar o comando:

#flowmanager eth0 -S

Desta forma todas as filas e filtros usados seriam removidos.

3.2.6 Dependências

O flowmanager utiliza o comando tc para criar e modificar as filas. É portanto necessário instalar o pacote iproute2. Como usa as filas HTB e SFQ é necessário também que o kernel as tenha instaladas. Em todas as distribuições de Linux testadas, o kernel por defeito possui já suporte para este tipo de filas.

3.2.7 Instalação

O código fonte do flowmanager possui um Makefile que torna simples o processo de compilar e instalar o programa. Para compilar basta executar:

#make

É assim criado o executável *flowmanager*. Isto é tudo o que é necessário para posteriormente correr o flowmanager. Basta portanto copiar este ficheiro para um directório de executáveis. Como tem que ser corrido com permissões de root, decidimos colocá-lo em /usr/sbin.

Capítulo 4

Route Manager

O routemanager é um componente que, tal como o nome indica, controla as rotas do sistema, mas não só. Opera também o flowmanager, revisto no capitulo anterior.

4.1 Introdução teórica

Desde o kernel 2.2 que o Linux possui um sistema de rede renovado. Apesar disso, é comum na maioria das distribuições os velhinhos comandos *ifconfig*, *arp* e *route*. Na maioria das vezes funcionam correctamente, mas já existem situações em que os resultados acabam por ser inesperados. No entanto, usando o pacote iproute2 possuímos uma ferramenta muito mais poderosa que concentra aquelas três ferramentas numa só. Mais uma vez, para uma detalhada explicação das funcionalidades e modo de utilização deste pacote pode consultar [HGM+06]. Seguidamente daremos uma breve explicação de funcionalidades que utilizamos para a realização do componente.

4.1.1 Como é feito o route de um pacote em Linux

Quando o kernel tem que despachar um pacote, vai em primeiro lugar verificar as regras de routing. Nas regras de routing temos, por níveis de prioridade diversas tabelas. Se executar o comando:

```
#ip rule
```

obterá um resultado semelhante a este:

```
0: from all lookup local
32766: from all lookup main
```

32767: from all lookup default

Isto significa que em primeiro lugar o kernel tentará obter uma regra na tabela local, depois na tabela main e por fim na tabela default. Isto vê-se devido à prioridade da tabela, ou seja 0 para a local, 32766 para a main e 32767 para a default. A tabela local é onde estão os IP's que correspondem à máquina local. A tabela main é a tabela principal do sistema. É lá que são colocadas as rotas introduzidas quando não é especificada a tabela.

O kernel vai percorrendo as tabelas até encontrar a regra que vai aplicar. Quer isto dizer que, se introduzirmos uma tabela com prioridade inferior, por exemplo 32765, e se nesta tabela tiver uma rota por defeito, o pacote é enviado por essa regra e nunca chega a entrar nas tabelas main e default.

4.1.2 Intermediate Queueing Device (IMQ)

o IMQ não é um novo tipo de fila mas está directamente ligado às filas. Tal como já foi dito na secção 3.1.1 as filas apenas podem ser colocadas no egress, ou seja, na saida dos interfaces. Desta forma surgem duas limitações:

- Só é possível controlar o upload.
- Uma fila só controla a largura de banda de um interface, o que pode levar a limitações.

O IMQ foi desenvolvido para acabar com estas limitações. Não é mais que um interface virtual. Na verdade pode ser mais que um interface virtual. Nele podemos colocar as filas tal como noutro qualquer interface. Para que tudo isto funcione, temos que fazer com que os pacotes de um ou mais interfaces passem pelo IMQ. Para tal existe uma extensão para o *iptables*. *Iptables* é a firewall do Linux, no entanto tem muito mais funcionalidades, entre as quais mais esta. Com o iptables podemos fazer com que pacotes provenientes de um ou vários interfaces sejam enviados para um IMQ, tornando desta forma possível agregar tráfego de vários interfaces como também fazer o controle de tráfego do download.

4.2 Implementação

4.2.1 Balanceamento de carga

O balanceamento de carga consiste em dividir as ligações TCP e UDP por dois ou mais interfaces. Esta divisão é feita ao nível do fluxo, ou seja, sempre

que uma ligação é estabelecida pelo ISP1 todos os pacotes dessa ligação vão pelo ISP1.

O comando *ip* permite balanceamento de carga. Para que isso aconteça é necessário activar no kernel a opção "IP: equal cost multipath"sem no entanto activar a opção "IP: equal cost multipath with caching support".

O balanceamento de carga é feito executando o comando:

```
#ip route add default equalize nexthop via ISP1 weight x nexthop
via ISP2 weight y
```

Com este comando adicionamos uma rota por defeito que é igualizada entre dois ISP's. O "weight"é o peso da ligação. Para ligações com larguras de banda diferentes coloca-se o peso consoante a proporção para que o kernel faça a divisão equitativamente.

Como as ligações são normalmente com IP's dinâmicos, torna-se mais difícil configurar isto, pois se o cliente dhcp for posto a correr vai adicionar uma nova rota "default", o que faz com que o balanceamento de carga deixe de ser possível. Devido a isto, a solução adoptada foi criar uma nova tabela de prioridade inferior à tabela main. Desta forma o kernel iria verificar as rotas em primeiro lugar nesta tabela. Assim podíamos continuar a correr os clientes dhcp por forma a actualizar as rotas para os ISP's.

O routemanager agrega então os dados das rotas "default"presentes na tabela main numa rota na nova tabela. A esta nova tabela foi dado o nome "balance".

A informação relativamente ao peso de cada ligação estará presente em "/etc/routemanaget/ifname"em que "ifname"é o interface ou interfaces dos ISP's. O conteudo de cada um dos ficheiros é apenas:

```
WEIGHT=1
```

Se porventura as subredes dos dois ISP's for a mesma, o routemanager também igualiza as connecções na rede local.

No caso da nossa máquina ligada a dois ISP's a tabela balance poderia ser a seguinte:

```
192.167.1.1 dev ppp2 proto kernel scope link src 192.168.1.1
192.167.1.4 dev ppp0 proto kernel scope link src 192.168.1.1
192.167.5.3 dev ppp1 proto kernel scope link src 192.168.1.1
194.117.26.0/24 equalize
nexthop dev eth3 weight 1
nexthop dev eth2 weight 1
192.168.0.0/24 dev eth1 scope link
default equalize
```

```
nexthop via 194.117.26.254 dev eth3 weight 1
nexthop via 194.117.26.254 dev eth2 weight 1
```

Já a tabela main que lhe deu origem seria algo assim:

```
192.167.1.1 dev ppp2 proto kernel scope link src 192.168.1.1
192.167.1.4 dev ppp0 proto kernel scope link src 192.168.1.1
192.167.5.1 dev ppp1 proto kernel scope link src 192.168.1.1
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.1
194.117.26.0/24 dev eth2 proto kernel scope link src 194.117.26.114
194.117.26.0/24 dev eth3 proto kernel scope link src 194.117.26.126
default via 194.117.26.254 dev eth3
default via 194.117.26.254 dev eth2
```

4.2.2 Controlo de Admissão

Como foi explicado no capitulo 2.4 a autenticação é feita usando o PPPoE. Desta forma, por cada utilizador que se connect é criada uma ligação ppp. O routemanager, sempre que detecta que existe uma nova ligação ppp adiciona-a à tabela balance.

O routemanager adiciona todas as ligações ppp e também as que pertençam às rotas "default". Desta forma, mesmo que um utilizador mal intencionado coloque o IP manualmente não conseguirá comunicar. Isto acontece porque, como é nesta tabela que está a rota "default", mesmo que o dito utilizador envie um pedido, o router não consegue encontrar a rota que lhe permite devolver o pedido.

4.2.3 Interacção com Flow Manager

A continua verificação das tabelas de rotas, permite que, quando existe uma nova ligação ppp, mas que não esteja nas rotas "default"é executado o flowmanager para adicionar o utilizador. Quando uma ligação é terminada é executado o flowmanager para remover o utilizador. Na verdade executa duas vezes o flowmanager. Uma para o upload e outra para o download.

4.2.4 Esquema de funcionamento

Tal como foi dito na secção anterior, existem dois flowmanager a funcionar. Estes são colocados cada um num IMQ. No imq0 é colocado o flow-

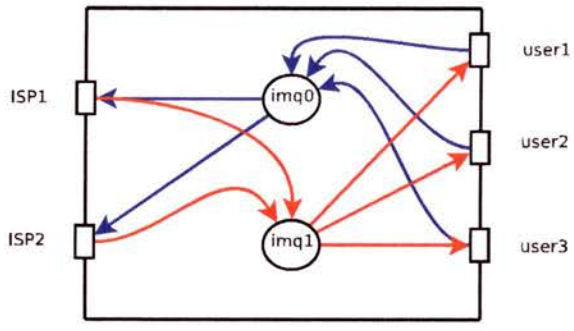

Figura 4.1: Uso do IMQ

manager que controla o upload. No imq1 é colocado o flowmanager que controla o download.

Na figura 4.2.4 está esquematizado a forma como é feita a agregação do tráfego dos vários interfaces pelos imq's.

Tal como foi dito na secção 4.1.2 cabe ao *iptables* colocar os pacotes provenientes dos interfaces no respectivo imq.

4.2.5 Controlo de download

O controlo de download é algo muito difícil de fazer. Como não controlamos o tráfego que nos chega parece mesmo impossível conseguir fazer no download o mesmo que é feito no download, ou seja, divisão de largura de banda por utilizador com base nas classes. Como é iniciado um flowmanager para o upload e outro para o download, dá-nos ainda mais variàvies de controlo das classes. Claro que isto apenas é possível se na verdade for possível controlar o download.

Acontece que a grande maioria do tráfego Internet é feito usando TCP/IP. Este protocolo possui propriedades que permitem reduzir o débito. Se porventura os pacotes se perdem ou demoram muito a chegar o débito é automaticamente reduzido. Vamos tentar tirar proveito desta particularidade.

Para conseguir com que haja controlo da largura de banda tem que ser possível manter em fila durante algum tempo os pacotes de fluxos que queiramos reduzir o débito. Temos então que deixar uma parte da largura de banda livre para que seja utilizada pelos fluxos que podem aumentar o débito. No capítulo 5 é apresentada uma solução para a largura de banda a reservar para este efeito.

4.2.6 instalação e dependências

Para o routemanager funcionar é necessário que o kernel tenha suporte IMQ. Para tal é necessário que seja feito o "patch"ao kernel, visto que este modulo não está presente em nenhuma distribuição testada.

É também essencial que o iptables tenha o suporte para o IMQ. Neste caso é também aplicado um "patch".

Como se trata de um script não precisa de instalação, basta correr o script.

Capítulo 5

Análise de resultados

Tal como foi explicado no 4.2.5 é necessário deixar uma certa largura de banda livre para que seja possível o controlo do download.

Nos nossos testes configuramos o flowmanager para uma ligação a 2Mbps. Seguidamente, simulamos uma ligação, através de uma outra máquina com mais 500kbps. Na figura 5 podemos ver que que o controlo do download não funcionou pois o utilizador da classe 5 deveria ter ficado com quase a totalidade da largura de banda.

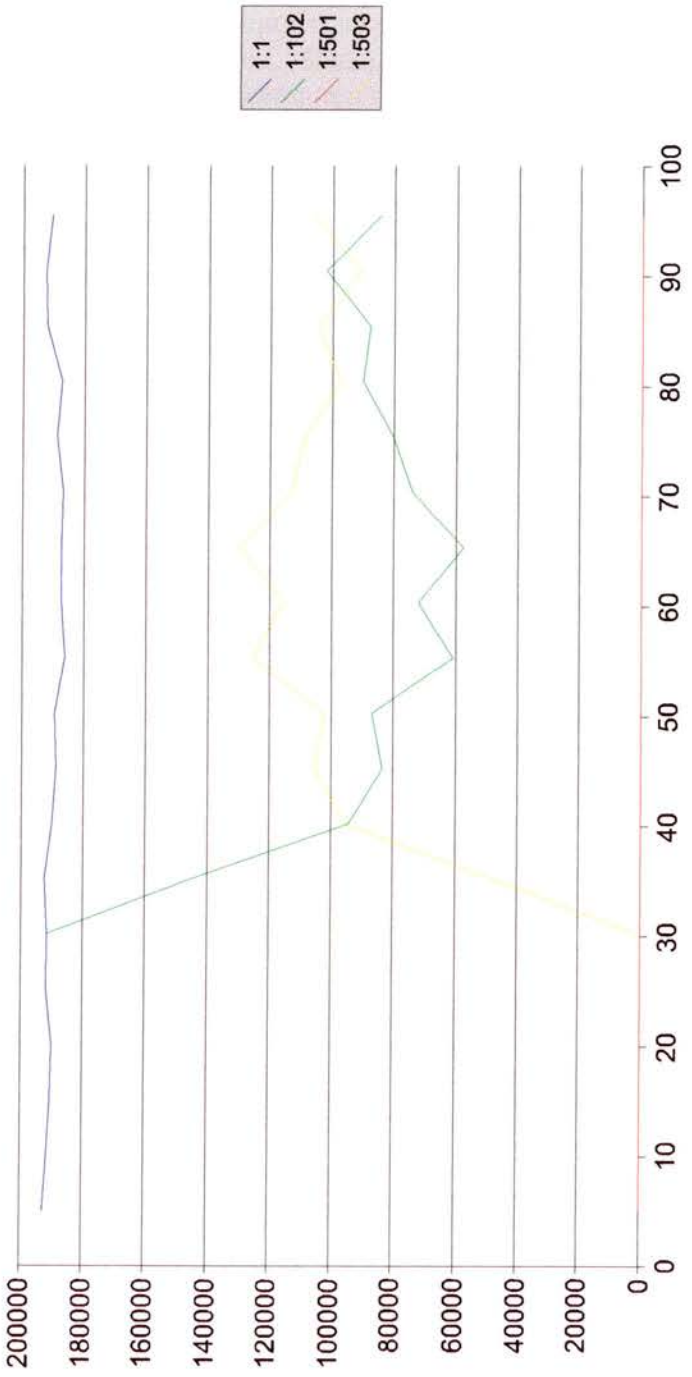

Figura 5.1: Diferença da largura de banda de 500kbps para ligação a 2,5Mbps

Seguidamente utilizamos a mesma ligação a 2Mbps mas desta vez configuramos a ligação à outra máquina para emular uma diferença de largura de banda progressivamente maior. Como fica patenteado na figura, quando a largura de banda entre a recebida e a configurada no flowmanager, torna-se possível o controlo do download. Vê-se claramente que, enquanto o utilizador de classe 5 não transmite, a banda de download é toda ocupada pelo utilizador de classe 1. No entanto, assim que o utilizador de classe 5 recebe um pedido, o débito do utilizador de classe cai abruptamente. Assim que o utilizador de classe 5 deixa de receber o pedido, imediatamente o utilizador de classe 1 volta a ficar com toda a largura de banda disponível.

Este teste foi feito para uma largura de banda de 3Mbps na ligação à rede e configurado a 2Mbps no flowmanager. Importa no entanto saber se esta diferença é sempre igual independentemente da velocidade da ligação ou se, pelo contrário, se esta diferença é proporcional à velocidade da ligação.

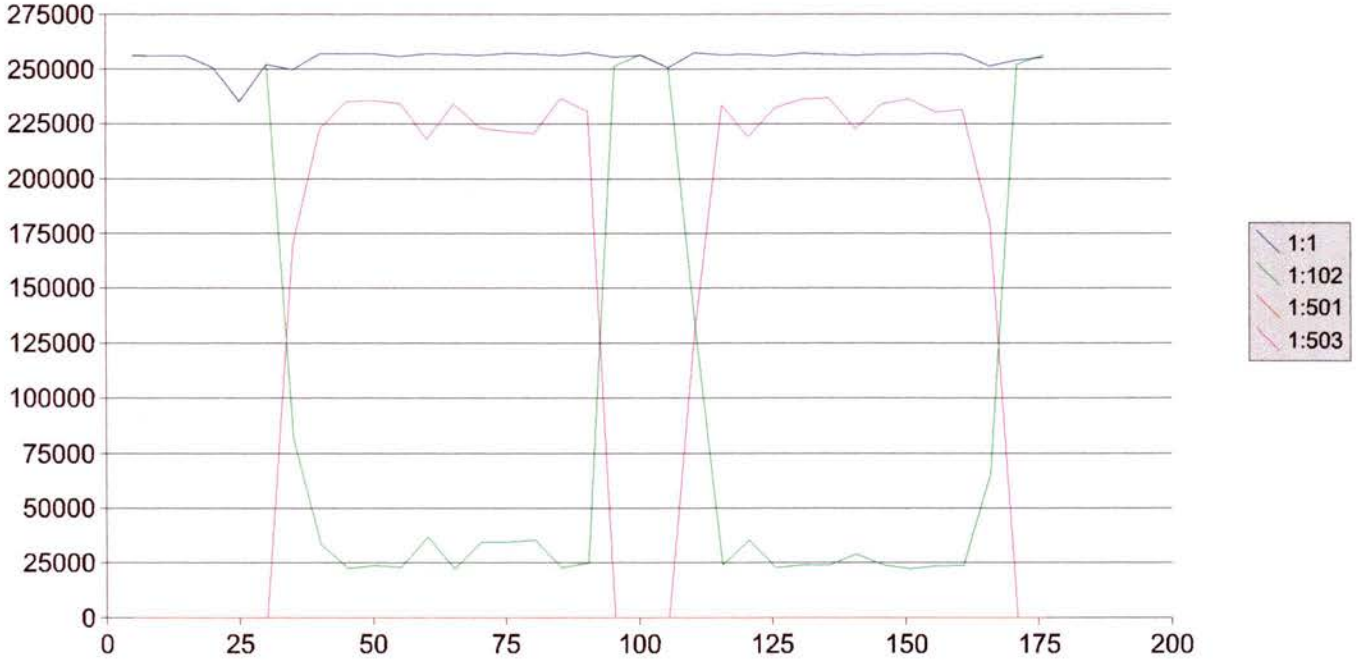

Figura 5.2: Diferença da largura de banda de 1mbps para ligação a 3Mbps

Neste teste emulamos uma ligação a 5Mbps. Vamos tentar mostrar que a largura de banda que é necessário reservar para que ocorra controlo de download se mantenha independentemente da velocidade. Se isto não acontecer torna-se impraticável utilizar este método pois utilizaria grande parte da largura de banda disponível.

Como se pode ver pela figura 5 a largura de banda de reserva necessária para controlo de download é independente da velocidade e igual a 1Mbps. Este resultado é muito importante e mostra que podemos ter controlo sobre o download das ligações, isto apenas para ligações TCP.

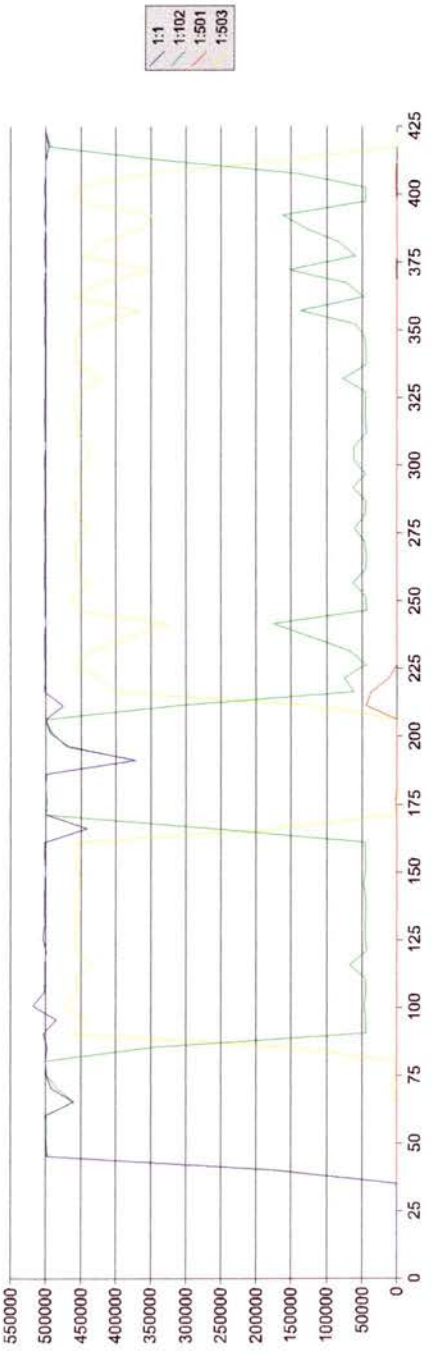

Figura 5.3: Diferença da largura de banda de 1Mbps para ligação a 5Mbps

Capítulo 6

Conclusão

A realização deste projecto conjuntamente com o projecto Gatebox deu origem à caixa Gatebox.

Os resultados obtidos são animadores, principalmente ao nível no controlo do download. Este é um ponto muito debatido mas com poucas implementações práticas. Por norma a solução obvia que se propõem é a obtenção de acordos com o operador de forma a que seja ele a fazer o controlo.

É importante salientar que praticamente todos os objectivos iniciais foram cumpridos, faltando apenas alguns testes reais para validar o trabalho feito até então.

Ao nível da fiabilidade convém referir que o routemanager e o flowmanager têm estado a correr interruptamente à mais de 300 horas sem qualquer falha.

6.1 Trabalho futuro

Como trabalho futuro importa testar este projecto numa situação real.

É importante testar o controlo do download noutros ambientes, nomeadamente em situações reais com ligações ADSL e também usando o balanceamento de carga.

Bibliografia

[HGM+06] Bert Hubert, Thomas Graf, Greg Maxwell, Remco van Mook, Martijn van Oosterhout, Paul B Schroeder, Jasper Spaans, and Pedro Larroy. *Linux Advanced Routing & Traffic Control*. www.lartc.org/howto, Junho 2006.

[OS06] Bruno Oliveira and Vítor Silva. Gatebox. Technical report, Faculdade de Engenharia da Universidade do Porto, Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL, Junho 2006.

Apêndice A

Código fonte do Flowmanager

A.1 conffile.h

```
001 #ifndef _CONFFILE_H_
002 #define _CONFFILE_H_
003
004 #include <iostream>
005 #include <fstream>
006 #include <string>
007 #include <vector>
008
009 using namespace std;
010 #define MAXBUF 1024
011
012
013
014 class ConfFile
015 {
016   unsigned int _maxclass;
017   unsigned int _maxrate;
018   string _ifname;
019   int _filter;
020   vector<int> _pesos;
021   vector<int> _prio;
022
023   public:
024   ConfFile();
025   ConfFile(char *confname);
026   ConfFile(string confname);
027   bool open(char *confname);
```

```
069                 pos= line.find_first_not_of(' ',pos);
070                 _ifname=line.substr(pos);
071               }
072             }
073             else //not IFNAME
074             {
075               pos = line.find("DST",0);
076               if(pos != string::npos)
077               {
078                 _filter=16;
079               }
080               else // not DST
081               {
082                 pos = line.find("PESOS",0);
083                 if(pos != string::npos)
084                 {
085                   pos=line.find_first_of(' ',pos);
086                   if(pos != string::npos)
087                   {
088
089                     for(unsigned int i=0;i<_maxclass;i++)
090                     {
091                       pos= line.find_first_not_of(' ',pos);
092                       line=line.substr(pos);
093                       pos = line.find_first_of(' ');
094                       val=line.substr(0,pos);
095                       _pesos[i]=atoi(val.c_str());
096
097                     }
098                   }
099                 }
100                 else
101                 {
102                   pos = line.find("PRIO",0);
103                   if(pos != string::npos)
104                   {
105                     pos=line.find_first_of(' ',pos);
106                     if(pos != string::npos)
107                     {
108
109                       for(unsigned int i=0;i<_maxclass;i++)
110                       {
111                         pos= line.find_first_not_of(' ',pos);
```

```
026       if(line.length())  //lines with only the \n
027       {
028         string::size_type pos=line.find('#',0);
029
030         if(pos != string::npos)
031         {
032           line.erase(pos,line.length()-pos);
033         }
034         //Only what we want
035         pos = line.find("CLASSES",0);
036         if(pos != string::npos)
037         {
038           pos=line.find_first_of(' ',pos);
039           if(pos != string::npos)
040           {
041             pos= line.find_first_not_of(' ',pos);
042             val=line.substr(pos);
043             _maxclass=atoi(val.c_str());
044             _pesos.resize(_maxclass);
045             _prio.resize(_maxclass);
046           }
047         }
048         else //not CLASSES
049         {
050           pos = line.find("MAXRATE",0);
051           if(pos != string::npos)
052           {
053             pos=line.find(' ',pos);
054             if(pos != string::npos)
055             {
056               pos= line.find_first_not_of(' ',pos);
057               val=line.substr(pos);
058               _maxrate=(unsigned int)atoi(val.c_str());
059             }
060           }
061           else //not MAXRATE
062           {
063             pos = line.find("IFNAME",0);
064             if(pos != string::npos)
065             {
066               pos=line.find_first_of(' ',pos);
067               if(pos != string::npos)
068               {
```

```
028   const char* ifname();
029   unsigned int maxclass();
030   unsigned int maxrate();
031   int peso(int i);
032   int prio(int i);
033   int filter();
034   virtual ~ConfFile();
035 };
036
037
038
039 #endif //_CONFFILE_H_
040
041
```

A.2 conffile.cpp

```
001 #include "conffile.h"
002
003
004 ConfFile::ConfFile()
005 {
006   _maxclass=0;
007   _maxrate=0;   //In Kbit
008   _filter=12; //position of the src ip on a IP packet.
009 }
010
011 bool ConfFile::open(char *confname)
012 {
013   fstream conffile;
014   char tmp[MAXBUF];
015   string line,val;
016
017   conffile.open(confname,fstream::in);
018
019   if(conffile.is_open())
020   {
021     conffile.getline(tmp,MAXBUF);
022     while(conffile.gcount())
023     {
024       line.clear();
025       line=tmp;
```

```
112                         line=line.substr(pos);
113                         pos = line.find_first_of(' ');
114                         val=line.substr(0,pos);
115                         _prio[i]=atoi(val.c_str());
116
117                       }
118                     }
119                   }
120                 }
121               }
122             }
123
124           }
125         }
126
127       }
128       conffile.getline(tmp,MAXBUF);
129     }
130   }
131   else
132   {
133     return false;
134   }
135   return true;
136 }
137 ConfFile::ConfFile(string confname)
138 {
139   ConfFile(confname.c_str());
140 }
141 ConfFile::~ConfFile()
142 {
143
144 }
145
146 ConfFile::ConfFile(char *confname)
147 {
148   open(confname);
149 }
150
151
152 const char* ConfFile::ifname()
153 {
154   return _ifname.c_str();
```

012
013

A.4 tcapi.h

```
001 #ifndef _TCAPI_H_
002 #define _TCAPI_H_
003
004 #include <iostream>
005 #include <fstream>
006 #include <string>
007 #include <sstream>
008 #include "conffile.h"
009 #define BURST 1514
010 #define CBURST 1514
011 #define LATENCY 50
012
013 using namespace std;
014
015
016 struct DATA
017 {
018   string ip;
019   unsigned int id;
020 };
021
022 class TcApi
023 {
024   DATA **data;
025   ConfFile conf;
026   vector<int> nclients;
027
028   public:
029   TcApi();
030   virtual ~TcApi();
031   void init();
032   void read_conf(char* confname);
033   //If necessary to backup data....
034   //void Read(string filename);
035   //void Write(string filename);
036   bool add(char* ip, unsigned int classid);
037   bool del(char* ip, unsigned int classid);
```

```
155 }
156
157 unsigned int ConfFile::maxclass()
158 {
159   return _maxclass;
160 }
161
162 unsigned int ConfFile::maxrate()
163 {
164   return _maxrate;
165 }
166
167
168 int ConfFile::peso(int i)
169 {
170   return _pesos[i];
171 }
172
173 int ConfFile::prio(int i)
174 {
175   return _prio[i];
176 }
177 int ConfFile::filter()
178 {
179   return _filter;
180 }
181
182
```

A.3 fifo_data.h

```
001 #ifndef _FIFO_DATA_H_
002 #define _FIFO_DATA_H_
003
004 struct FIFO_DATA
005 {
006   char opt;
007   char address[20];
008   unsigned int classid;
009 };
010
011 #endif //_FIFO_DATA_H_
```

```
038   bool clean();
039   bool flush();
040   void reset();
041   bool change();
042   bool list();
043 };
044
045
046 #endif //_DATABASE_H_
047
048
```

A.5 tcapi.cpp

```
001 #include "tcapi.h"
002
003
004 TcApi::TcApi()
005 {
006   data=NULL;
007 }
008
009
010 TcApi::~TcApi()
011 {
012
013 }
014
015 void TcApi::init()
016 {
017   stringstream value;
018   clean();
019   stringstream tmp,sclassid;
020   tmp <<"tc qdisc add dev " << conf.ifname() <<" root handle 1:
htb default ffff";
021   string command =tmp.str();
022   system(command.c_str());
023   tmp.str("");tmp.clear();
024   tmp << "tc class add dev "<<conf.ifname() << " parent 1:0
classid 1:1 htb rate "<<conf.maxrate()<<"kbit burst " << BURST ;
025   command = tmp.str();
026   system(command.c_str());
```

```
027   tmp.str("");tmp.clear();
028   tmp << "tc class add dev "<<conf.ifname() << " parent 1:1
classid 1:ffff htb rate 1kbit prio 7 burst " << BURST ;
029   command = tmp.str();
030   system(command.c_str());
031
032 }
033
034 void TcApi::read_conf(char * confname)
035 {
036   if(!conf.open(confname))
037   {//Error in conf File
038     cout <<"Error in "<< confname << endl;
039     exit(1);
040   }
041   nclients.resize(conf.maxclass());
042   data = (DATA**) new DATA*[conf.maxclass()];
043   for(unsigned int i=0;i<conf.maxclass();i++)
044   {
045     nclients[i]=0;
046     data[i]= (DATA*) new DATA[256];
047     for(int j=0;j<256;j++)
048     {
049       (data[i][j]).id=0;
050     }
051   }
052 }
053
054
055 bool TcApi::add(char* ip, unsigned int classid)
056 {
057   stringstream tmp,clientid,value;
058   unsigned int subip;
059   if(!data)
060   {
061     cout << "Not read_conf" << endl;
062     return false;
063   }
064   if(classid > conf.maxclass())
065   {
066     return false;
067   }
068
```

```
069   string ip_str=ip;
070   string::size_type pos;
071   pos=ip_str.find('.',0);
072   pos=ip_str.find('.',pos+1);
073   pos=ip_str.find('.',pos+1);
074   ip_str=ip_str.substr(pos+1);
075   tmp.str("");
076   tmp.clear();
077   tmp << ip_str;
078   tmp >> subip;
079
080
081   tmp.str("");
082   tmp.clear();
083
084   clientid.str("");clientid.clear();
085   value.str("");value.clear();
086
087   clientid << hex << subip+classid*256;
088
089   value << hex << classid;
090   //put the information
091   if((data[classid-1][subip]).id==0)
092   {
093     (data[classid-1][subip]).id=subip+classid*256;
094     if((data[classid-1][subip]).ip.length() ==0)
095     {
096       (data[classid-1][subip]).ip=ip;
097       tmp << "tc class add dev "  <<  conf.ifname() << " parent
1:1 classid 1:" << clientid.str() <<" htb rate
"<<conf.maxrate()<<"kbit burst " << BURST  <<" prio "
<<conf.prio(classid-1);
098       string command = tmp.str();
099
100       system(command.c_str()) ;
101       tmp.str("");
102       tmp.clear();
103       command.clear();
104       tmp << "tc qdisc add dev " << conf.ifname() << " parent
1:" << clientid.str() << " handle " << clientid.str() << ": sfq
perturb 5 ";
105       command=tmp.str();
106       system(command.c_str());
```

```
107         tmp.str("");
108         tmp.clear();
109         command.clear();
110         tmp << "tc filter add dev "<<conf.ifname() << " parent
1:0 handle 0x"<< clientid.str() << " protocol ip prio 1 u32  match
u32 "<< subip <<"  0x000000FF at "<< conf.filter() << " match u32
0x"<< value.str() <<"00 0x0000FF00 at " << conf.filter() << "
flowid 1:" << clientid.str();
111         command = tmp.str();
112         system(command.c_str());
113
114       }
115
116       nclients[classid-1]++;
117       change();
118       return true;
119     }
120     return false;
121 }
122
123 bool TcApi::del(char* ip, unsigned int classid)
124 {
125   stringstream tmp,clientid,value;
126   unsigned int subip;
127   if(!data)
128   {
129     return false;
130   }
131   if(classid>conf.maxclass())
132   {
133     return false;
134   }
135
136   string ip_str(ip);
137   string::size_type pos;
138   pos=ip_str.find('.',0);
139   pos=ip_str.find('.',pos+1);
140   pos=ip_str.find('.',pos+1);
141   ip_str=ip_str.substr(pos+1);
142   tmp.str("");
143   tmp.clear();
144   tmp << ip_str;
145   tmp >> subip;
```

```
146
147
148   if((data[classid-1][subip]).id!=0)
149   {
150     (data[classid-1][subip]).id=0;
151     ((data[classid-1][subip]).ip).clear();
152
153     // run tc:
154     clientid.str("");clientid.clear();
155     value.str("");value.clear();
156     tmp.str("");tmp.clear();
157
158     clientid << hex << subip+classid*256;
159     value << hex << classid;
160     tmp << "tc filter del dev "<<conf.ifname() << " parent 1:0
handle 800::"<< clientid.str() << " protocol ip prio 1 u32  match
u32 "<< subip <<"  0x000000FF at "<< conf.filter() << " match u32
0x"<< value.str() <<"00 0x0000FF00 at "<< conf.filter() <<" flowid
1:" << clientid.str();
161     string command = tmp.str();
162     system(command.c_str());
163     tmp.str("");
164     tmp.clear();
165     command.clear();
166     tmp << "tc class del dev "  <<  conf.ifname() << " parent
1:1 classid 1:" << clientid.str() <<" htb rate
"<<conf.maxrate()<<"kbit burst " << BURST  <<" prio " <<
conf.prio(classid-1);
167     command = tmp.str();
168     system(command.c_str());
169
170     nclients[classid-1]--;
171     change();
172     return true;
173   }
174   return false;
175 }
176
177 bool TcApi::change()
178 {
179   unsigned int rate[conf.maxclass()];
180   unsigned long int total=0;
181   stringstream clientid;
```

```
182   cout << "Clients:" << endl;
183   for(unsigned int i=0;i<conf.maxclass();i++)
184   {
185     total+=conf.peso(i)* nclients[i];
186   }
187   if(total==0)
188   {
189     return true;
190   }
191   for(unsigned int i=0;i<conf.maxclass();i++)
192   {
193     rate[i]=conf.maxrate() * conf.peso(i)*nclients[i] /total;
194     if(rate[i] !=0)
195     {
196       stringstream tmp;
197       string command;
198       for(int j=0;j<256;j++)
199       {
200         if((data[i][j]).id)
201         {
202
203
204           tmp.str("");tmp.clear();
205           clientid.str(""); clientid.clear();
206           clientid << hex << (data[i][j]).id;
207           cout << (data[i][j]).ip << "\t1:" <<clientid.str()
<<"\t" << rate[i]/nclients[i] <<"kbit"  << endl;
208           tmp << "tc class change dev "  <<  conf.ifname() << "
parent 1:1 classid 1:" << clientid.str() <<" htb rate "<<
rate[i]/nclients[i] <<"kbit ceil "<< conf.maxrate() <<"kbit burst "
<< BURST   << " prio " << conf.prio(i); //<< " cburst " << CBURST
209           command=tmp.str();
210           system(command.c_str());
211         }
212       }
213     }
214   }
215
216
217   return true;
218 }
219 bool TcApi::clean()
220 {
```

```
221   string command = "tc qdisc del dev " ;
222   command += conf.ifname();
223   string tmp= " root 2>/dev/null";
224   command +=tmp;
225   system(command.c_str());
226   return true;
227 }
228 bool TcApi::list()
229 {
230   unsigned int rate[conf.maxclass()];
231   unsigned long int total=0;
232   stringstream clientid;
233
234   if(!data)
235   {
236     return false;
237   }
238
239   for(unsigned int i=0;i<conf.maxclass();i++)
240   {
241     total+=conf.peso(i)* nclients[i];
242   }
243   cout << "Clients:" << endl;
244   if(total)
245   {
246     for(unsigned int i=0;i<conf.maxclass();i++)
247     {
248       rate[i]=conf.maxrate() * conf.peso(i)*nclients[i] /total;
249       if(rate[i] !=0)
250       {
251         for(int j=0;j<256;j++)
252         {
253           if(data[i][j].id !=0)
254           {
255             clientid.str(""); clientid.clear();
256             clientid << hex << (data[i][j]).id;
257             cout << (data[i][j]).ip << "\t1:" <<clientid.str()
<<"\t" << rate[i]/nclients[i] <<"kbit"  << endl;
258           }
259         }
260       }
261     }
262   }
```

```
263   return true;
264 }
265
266
```

A.6 flowfunc.h

```
001 #ifndef _FLOWFUNC_H_
002 #define _FLOWFUNC_H_
003
004 #include <string>
005 #include <iostream>
006 #include <sstream>
007 #include <sys/types.h>
008 #include <sys/stat.h>
009 #include <fcntl.h>
010 #include <unistd.h>
011 #include "conffile.h"
012 #include "tcapi.h"
013 #include "fifo_data.h"
014
015
016
017 void add(int fifo,char * ip);
018 void del(int fifo,char *ip);
019 void stop(int fifo);
020 void usage(char *progname);
021 void loop(TcApi &tc,string fifo);
022 void list(int fifo);
023
024 #endif //_FLOWFUNC_H_
025
026
```

A.7 flowfunc.cpp

```
001 #include "flowfunc.h"
002
003 void add(int fifo,char * ip)
004 {
005   string ip_str=ip;
```

```
006   int classid;
007   stringstream tmp;
008   string::size_type init, end;
009   init=ip_str.find('.',0);
010   init=ip_str.find('.',init+1);
011   end=ip_str.find('.',init+1);
012   ip_str=ip_str.substr(init+1,end-init-1);
013   tmp.str("");
014   tmp << ip_str;
015   tmp >> classid;
016   FIFO_DATA data;
017   data.opt='A';
018   for(int i=0;((data.address)[i]=ip[i]);i++);
019   data.classid=classid;
020   write(fifo,&data,sizeof(FIFO_DATA));
021 }
022
023 void del(int fifo,char *ip)
024 {
025   string ip_str=ip;
026   int classid;
027   stringstream tmp;
028   string::size_type init, end;
029   init=ip_str.find('.',0);
030   init=ip_str.find('.',init+1);
031   end=ip_str.find('.',init+1);
032   ip_str=ip_str.substr(init+1,end-init-1);
033   tmp.str("");
034   tmp << ip_str;
035   tmp >> classid;
036   FIFO_DATA data;
037   data.opt='D';
038   for(int i=0;((data.address)[i]=ip[i]);i++);
039   data.classid=classid;
040   write(fifo,&data,sizeof(FIFO_DATA));
041 }
042
043 void stop(int fifo)
044 {
045   FIFO_DATA data;
046   data.opt='S';
047   write(fifo,&data,sizeof(FIFO_DATA));
048 }
```

```
049 void list(int fifo)
050 {
051   FIFO_DATA data;
052   data.opt='L';
053   write(fifo,&data,sizeof(FIFO_DATA));
054 }
055 void usage(char *progname)
056 {
057   cout <<endl << progname << " ID OPT" << endl;
058   cout <<"OPT := [-A IP] [-D IP] [-S] [-L] [-I CONF]" << endl;
059 }
060
061
062 void loop(TcApi &tc, string fifoname)
063 {
064   char state=0;
065   FIFO_DATA data;
066   int fifo=open(fifoname.c_str(),O_RDONLY);
067   while(state!='S')
068   {
069
070     int bytes = read(fifo,&data,sizeof(FIFO_DATA));
071     if(bytes >0)
072     {
073       switch(data.opt)
074       {
075         case 'A':
076           tc.add(data.address,data.classid);
077         break;
078         case 'D':
079           tc.del(data.address,data.classid);
080         break;
081         case 'S':
082           state='S';
083           tc.clean();
084           close(fifo);
085           unlink(fifoname.c_str());
086         break;
087         case 'L':
088           tc.list();
089           break;
090
091       }
```

```
092     }
093     else
094     {
095       close(fifo);
096       fifo=open(fifoname.c_str(),O_RDONLY);
097     }
098   }
099 }
100
101
```

A.8 flowmanager.cpp

```
001 #include "tcapi.h"
002 #include "flowfunc.h"
003 #include "fifo_data.h"
004 #include <sys/types.h>
005 #include <sys/stat.h>
006 #include <fcntl.h>
007 #include <unistd.h>
008
009 int main(int argc, char *argv[])
010 {
011   TcApi tc;
012   int fifo;
013   string fifoname="/var/tmp/flow-";
014   if(argc>2)
015   {
016     fifoname+=argv[1];
017     if(argv[2][0] == '-')
018     {
019       switch (argv[2][1])
020       {
021         case 'A': //ADD
022           if(argc == 4)
023           {
024             fifo=open(fifoname.c_str(),O_WRONLY|O_NONBLOCK);
025             if(fifo==-1)
026             {
027               cout << "Server not running" << endl;
028               exit(1);
029             }
```

```
030             add(fifo,argv[3]);
031           }
032           else
033           {
034             usage(argv[0]);
035           }
036         break;
037         case 'D': //Delete
038           if(argc == 4)
039           {
040             fifo=open(fifoname.c_str(),O_WRONLY|O_NONBLOCK);
041             if(fifo==-1)
042             {
043               cout << "Server not running" << endl;
044               exit(1);
045             }
046             del(fifo,argv[3]);
047           }
048           else
049           {
050             usage(argv[0]);
051           }
052         break;
053         case 'F': //Flush
054           if(argc == 3)
055           {
056             //tc.flush();
057           }
058           else
059           {
060             usage(argv[0]);
061           }
062         break;
063         case 'I': //INIT
064           if(argc == 4)
065           {
066             tc.read_conf(argv[3]);
067             pid_t pid=fork();
068             if(pid==0)
069             {
070               if(mkfifo(fifoname.c_str(),0x600)==0)
071               {
072                 tc.init();
```

```
073                 loop(tc,fifoname);
074               }
075               else
076               {
077                 cout << "ERROR: Daemon Allready running or
invalid shutdown"<<endl;
078               }
079             }
080             if(pid==-1)
081             {
082               cout <<"Fork Error" << endl;
083             }
084           }
085           else
086           {
087             usage(argv[0]);
088           }
089         break;
090         case 'L':
091           fifo=open(fifoname.c_str(),O_WRONLY|O_NONBLOCK);
092           list(fifo);
093         break;
094         case 'S':
095           fifo=open(fifoname.c_str(),O_WRONLY|O_NONBLOCK);
096           stop(fifo);
097         break;
098         default: //Usage
099           usage(argv[0]);
100         break;
101       }
102     }
103     else
104     {
105       usage(argv[0]);
106     }
107   }
108   else
109   {
110     usage(argv[0]);
111   }
112 }
113
114
```

Apêndice B

Código fonte do Route Manager

```
001 #!/bin/bash
002 COUNTER=1
003 TABLE=balance
004 LOG1=route.log1
005 LOG2=route.log2
006 DEFAULT=default.tmp
007 WEIGHT=/etc/routemanager
008 LOGPATH=/var/log/routemanager
009 FLOWMANAGER=/home/master/LUIS/flowmanager/src/flowmanager
010 FLOWCONF0=/home/master/LUIS/flowmanager/src/imq0.conf
011 FLOWCONF1=/home/master/LUIS/flowmanager/src/imq1.conf
012 FLOWLOG0=flow.log.0
013 FLOWLOG1=flow.log.1
014
015 pkill -9 flowmanager
016 rm -rf $LOGPATH/*
017 rm -rf /var/tmp/flow-*
modprobe imq
018 ip link set imq0 up
019 ip link set imq1 up
020 iptables -t mangle -F PREROUTING
021 iptables -t mangle -F POSTROUTING
022 iptables -t mangle -A PREROUTING -s 192.167.0.0/16 -d !
192.168.0.0/16 -j IMQ --todev 0
023 iptables -t mangle -A POSTROUTING -d 192.167.0.0/16 -s !
192.168.0.0/16 -j IMQ --todev 1
024 trap "$FLOWMANAGER 0 -S & $FLOWMANAGER 1 -S  & ip rule del
```

```
table $TABLE ; exit" INT TERM
025 $FLOWMANAGER 0 -I $FLOWCONF0 >>$LOGPATH/$FLOWLOG0 &
026 $FLOWMANAGER 1 -I $FLOWCONF1 >>$LOGPATH/$FLOWLOG1 &
027 sleep 5
028
029 echo " " > $LOGPATH/$LOG1
030 ip rule |grep $TABLE | while read line;do
031   ip rule del table $TABLE
032 done
033 ip rule add from all table $TABLE prio 1
034 ip route flush table $TABLE
035 if [ ! -d $LOGPATH ]; then
036   if [ -e $LOGPATH ]; then
037     rm $LOGPATH
038   fi
039   mkdir $LOGPATH
040 fi
041 while [ $COUNTER -ne 0 ]; do
042   if [ -e $LOGPATH/*-subnet ]; then
043     rm $LOGPATH/*-subnet
044   fi
045   if [ -e $LOGPATH/$DEFAULT ]; then
046     rm $LOGPATH/$DEFAULT
047   fi
048   ip r > $LOGPATH/$LOG2
049
050   if [ -n  "`diff $LOGPATH/$LOG1 $LOGPATH/$LOG2 |grep default`"
] ;then
051     # changes in default
052       cat $LOGPATH/$LOG2 |grep ^default |cut -d' ' -f 2- |while
read  line; do
053       ifname="`echo $line |cut -d ' ' -f 4`"
054       echo " nexthop $line weight `cat $WEIGHT/$ifname |grep
^WEIGHT | cut -d '=' -f 2`" >> $LOGPATH/$DEFAULT
055       subnet="`cat $LOGPATH/$LOG2 |grep $ifname |grep "scope
link"|cut -d ' ' -f 1`"
056       subnet1="`echo $subnet|cut -d '/' -f 1`"
057       subnet2="`echo $subnet|cut -d '/' -f 2`"
058       subnetfile="$LOGPATH/$subnet1-$subnet2-subnet"
059       echo " nexthop dev $ifname weight `cat $WEIGHT/$ifname
|grep ^WEIGHT | cut -d '=' -f 2`" >> $subnetfile
060     done
061     if [ -n "`ip route show table $TABLE |grep default`" ];
```

```
then
062       ip route del table $TABLE default >> /dev/null
063     fi
064     ip route add table $TABLE equalize default `cat
$LOGPATH/$DEFAULT` > /dev/null
065
066
067     for subnetfile in `ls $LOGPATH/*-subnet`; do
068       subnet1="`echo $subnetfile| cut -d '/' -f 5 |cut -d '-'
-f 1`"  #the -f 5 is because of the $LOGPATH depth
069       subnet2="`echo $subnetfile| cut -d '/' -f 5 |cut -d '-'
-f 2`"
070       if [ -n "`ip route show table $TABLE |grep
$subnet1/$subnet2`" ]; then
071         ip route del table $TABLE $subnet1/$subnet2 >>
/dev/null
072       fi
073       ip route add table $TABLE equalize $subnet1/$subnet2 `cat
$subnetfile` >> /dev/null
074
075     done
076
077   fi
078
079   diff $LOGPATH/$LOG1 $LOGPATH/$LOG2 |grep ^">" |grep ppp |grep
-v default| cut -d '>' -f 2-| while read line; do
080     if [ -z "`ip route show table $TABLE|grep "$line" `" ];
then
081     ip route add table $TABLE $line >> /dev/null
082     fi
083     line="`echo "$line" |cut -d ' ' -f 1`"
084     $FLOWMANAGER 0 -A $line
085     $FLOWMANAGER 1 -A $line
086   done
087   diff $LOGPATH/$LOG1 $LOGPATH/$LOG2 |grep ^"<" |grep ppp |grep
-v default| cut -d '<' -f 2-| while read line; do
088     line="`echo "$line" |cut -d ' ' -f 1`"
089     $FLOWMANAGER 0 -D $line
090     $FLOWMANAGER 1 -D $line
091   done
092
093 sleep 10
094 mv $LOGPATH/$LOG2 $LOGPATH/$LOG1
```

```
095 done
096
097
```