Construindo Roteadores com Linux

by Christian Lyra
Pedro Torres

Conteúdo:

Introdução

Um roteador, em sua essência, é apenas uma máquina que recebe pacotes, toma alguma decisão com base neles e, finalmente, os repassa (ou não) adiante. No entanto, espera-se muito mais de um roteador do que apenas repassar pacotes. Funções que permitam o tráfego de diferentes protocolos, filtros de pacotes e geração de estatísticas são desejáveis e até mesmo essenciais.

No presente documento veremos como uma máquina com Linux poderá cumprir todas essas tarefas. Utilizaremos para isso o Sistema Operacional GNU/Linux com a distribuição Debian. Vale lembrar que estaremos tratando de roteadores com interfaces ethernet apenas!

A versão atualizada desse documento pode ser encontrada em http://lyra.soueu.com.br/tiki-index.php?page=LinuxRouter

Agradecimentos

Nossos agradecimentos a todos que nos ajudaram a escrever esse documento. Em especial aos colaboradores:

- Gentil de Bortoli Júnior (correções e sugestões no texto)
- Edson Flávio de Souza (correções no texto)

Motivação

Mas, afinal, por que usar PCs e Linux para construir roteadores quando temos disponíveis no mercado roteadores com hardware dedicado de marcas como a Cisco, Juniper, entre outras? Bem, mesmo levando em conta uma situação ideal onde houvesse recursos financeiros para adquirir tais equipamentos, ainda assim teríamos argumentos em favor de construir roteadores com Linux. Como vantagens podemos citar:

- Preço
- Recursos
- Disponibilidade

Comparativamente, a primeira vantagem é óbvia, o custo de aquisição de um PC pode ser algumas ordens de grandeza mais barato do que um roteador dedicado. Mesmo que tenhamos um roteador dedicado, um PC pode ser utilizado como backup em casos de emergência. Uma segunda vantagem que podemos citar são as features (recursos). Um roteador dedicado necessita, em muitas ocasiões, de módulos adicionais ou upgrades de firmware para que possam suportar um novo recurso ou um novo protocolo. Em um PC isso é mais fácil de ser configurado/implementado, permitindo também a pesquisa e o desenvolvimento de novos recursos e protocolos. E, por fim, em casos de pressa ou emergência é mais fácil adquirir um PC no mercado ou mesmo aproveitar uma máquina que já possuímos do que aguardar a importação e a liberação de um hardware dedicado ou mesmo de algumas de suas peças.

Essa abordagem possui algumas desvantagens também e entre elas poderíamos citar:

- Confiabilidade
- Desempenho
- Interfaces

Um hardware dedicado possui uma confiabilidade maior do que um PC comum. Podemos diminuir essa desvantagem utilizando-se um hardware mais robusto, que possua, por exemplo, fontes redundantes, um sistema de arquivos implementado em flash-ram, etc. O desempenho, embora possa ser considerado uma vantagem, pode ser também uma desvantagem. O desempenho de um PC comum é bem maior do que o de um roteador de pequeno ou médio porte. No entanto, se considerarmos um roteador de grande porte, a situação se inverte. E, por fim, um roteador dedicado normalmente apresenta uma variedade maior de interfaces (normalmente possui interfaces seriais e ethernet). Como estamos considerando aqui apenas interfaces ethernet, essa não chega a ser uma grande desvantagem!

O Hardware

Ao montarmos uma máquina para trabalhar como um roteador Linux, algumas considerações merecem atenção. A principal delas diz respeito a velocidade/capacidade de processamento. Apesar do Linux ser bastante eficiente em termos de roteamento e mesmo máquinas modestas poderem funcionar muito bem como roteadores, se o intuito for rodar nessa máquina também um filtro de pacotes (firewall) ou um IDS (detecção de intrusão), a capacidade de processamento faz diferença. Mais adiante seguem exemplos dos desempenhos esperados.

Outra preocupação que devemos ter são as placas de rede. Placas baratas, comumente encontradas no mercado, não possuem um desempenho aceitável, apesar de todas anunciarem ser "100Mbits". Nossa experiência tem mostrado que placas que possuem um bom desempenho são as 3COM e as placas baseadas no chipset Tulip. As placas baseadas no Chipset Realtek 8139 possuem um desempenho intermediário, enquanto que as placas baseadas no chipset SiS900? possuem um desempenho péssimo.

A memória é um outro fator que não pesa muito, ao menos que se queira executar um daemon de roteamento, como o Zebra, e se queira trabalhar com grandes tabelas de rotas. Em geral, 128MB são o suficiente.

O gabinete, embora não tenha influência no desempenho da máquina, pode ser um fator a ser considerado caso queira se colocar a máquina em um rack, por exemplo. Nesse caso, gabinetes do tipo desktop são mais apropriados.

E, por fim, outro fator que devemos considerar são as partes "móveis" da máquina. Os ventiladores da fonte e do processador, com o tempo acumulam poeira e tendem a parar. Eles merecem atenção regular. Em alguns casos, é possível utilizar apenas dissipadores de calor no processador, principalmente para máquinas de menor clock.

Desempenho

Teoriacamente falando, o barramento PCI de um PC comum funciona a 33Mhz e tem largura de 32bits, logo, isso daria uma capacidade de transportar aproximadamente 1Gbit/s. Claro que, na prática, obtemos valores bem inferiores a esse pois outros fatores como, o esquema de IRQs do PC, acabam trabalhando contra.

O Kernel

O kernel ou núcleo do sistema operacional GNU/Linux é o responsável, entre outras funções, pelo acesso aos dispositivos e protocolos de rede. Em muitos casos ele é auxiliado por daemons (processos executados em background), e por comandos que configuram parâmetros. Como o kernel é a peça central de tudo que faremos daqui em diante, é importante que ele tenha sido corretamente compilado para executar as funções que vamos exigir dele!

Em geral os kernels que acompanham as distribuições de Linux são compilados com uma grande quantidade de módulos e isto é suficiente na maioria dos casos. No entanto, é importante que o administrador de redes/sistemas saiba compilar seus próprios kernels. Veremos a seguir algumas opções importantes ao compilarmos um kernel para uma máquina que vai se transformar em um roteador. Vamos partir do princípio de que o leitor já possui alguma experiência para compilar e instalar um kernel.

Vamos começar pelas opções encontradas dentro do menu "Networking Option --->" (supondo que o leitor esteja usando um "make menuconfig"). Segue abaixo as opções que comumente utilizamos:

```
Packet socket
    *   Packet socket: mmapped IO
     Netlink device emulation
    * Network packet filtering (replaces ipchains)
    _  Network packet filtering debugging (NEW)
    * Socket Filtering
     Unix domain sockets
    * TCP/IP networking
    *   IP: multicasting
    *   IP: advanced router
    *     IP: policy routing (NEW)
    *       IP: use netfilter MARK value as routing key (NEW)
    *       IP: fast network address translation (NEW)
    _   IP: equal cost multipath (NEW)
    *     IP: use TOS value as routing key (NEW)
   _    IP: verbose route monitoring (NEW)
    _  IP: kernel level autoconfiguration
     M  IP: tunneling
     M  IP: GRE tunnels over IP
    *     IP: broadcast GRE over IP (NEW)
    *   IP: multicast routing
```

```
_   IP: PIM-SM version 1 support (NEW)
*     IP: PIM-SM version 2 support (NEW)
_  IP: TCP Explicit Congestion Notification support
_  IP: TCP syncookie support (disabled per default)
  IP: Netfilter Configuration  --->
  IP: Virtual Server Configuration  --->
M   The IPv6 protocol (EXPERIMENTAL) (NEW)
  IPv6: Netfilter Configuration  --->
M  802.1Q VLAN Support
---
< > The IPX protocol
< > Appletalk protocol support
Appletalk devices  --->
< > DECnet Support
 M  802.1d Ethernet Bridging
QoS and/or fair queueing  --->
Network testing  --->
```

OBS1: No próprio menu é possível encontrar uma breve descrição do que significa cada opção.

OBS2: Para habilitar o IPv6 é necessáro antes selecionar a opção "Prompt for development and/or incomplete code/drivers" no menu "Code maturity level options --->"

Dentro dos menus "IPv6: Netfilter Configuration --->", "IP: Netfilter Configuration --->" e "QoS and/or fair queueing --->" marque todas as opções como módulo. Assim poderemos escolher futuramente que opções de firewall/QOS utilizar.

O suporte às diferentes placas de rede pode ser habilitado em "Network device support --->", Ethernet (10 or 100Mbit) --->". Compilar o suporte para várias placas como módulo é uma boa idéia. Além do mais, se houverem placas diferentes na máquina, será mais fácil designar quem é quem (que placa será a eth0, eth1, etc).

Configurando o Console na Porta Serial

Para evitar ter que manter monitor e teclado em uma máquina que é somente um roteador, é interessante habilitar o console via porta serial, assim é possível conectar um terminal (burro, ou um kermit) na porta serial exatamente como se faria com um roteador de "verdade". Para fazer isso é necessário uma opção do

kernel e pequenas alterações em alguns arquivos de configuração.

Na configuração do kernel, dentro do menu "Character devices --->" marque a opção "Support for console on serial port". E faça as seguintes alterações (supondo que a serial a ser usada é a ttyS0):

- No arquivo de configuração do lilo, acrescente: append="console=/dev/ttyS0,9600" para permitir o log na porta serial enquanto a máquina boota;
- No arquivo /etc/inittab, descomente a linha "T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100";
- E, finalmente, no arquivo /etc/securetty, acrescente uma linha com o valor "ttyS0".

Comandos Básicos

Veremos a seguir os principais comandos utilizados para configurarmos interfaces, manipular rotas e até mesmo alguns comandos muito úteis para solucionar problemas. Não se deixe enganar pelo título, pois alguns comandos aqui apresentados são essencias na resolução de problemas bastante complexos!

Configuração da Interface de Rede

modprobe / insmod / lsmod / rmmod / modinfo

O primeiro passo para configurar um interface é garantir que o kernel saiba como "conversar" com esse dispositivo. Para tanto, o kernel deverá ou ter embutido o "driver" (que no Linux chamamos de módulo) ou carregar dinâmicamente esse "driver". Os comandos "mod" servem para carregar, descarregar e listar módulos, e são bastante simples de usar:

Para listar os módulos carregados pelo kernel use o comando lsmod, exemplo:

```
# lsmod
 Module                  Size  Used by    Not tainted
 e100                   49064   1
```

Para carregar um módulo é só utilizar o comando insmod ou o comando modprobe. A diferença entre os dois é que o comando modprobe resolve e carrega as dependências do módulo que estamos tentando carregar, isto é, se o módulo "A" necessita que o módulo "B" esteja carregado, usar "modprobe A" carregará os dois! Exemplo:

```
# modprobe 8139too
```

Alguns módulos aceitam parâmetros ao serem carregados como, por exemplo, IRQ, I/O, opções de velocidade, etc. Para obter mais informações sobre um módulo, utilize o comando modinfo. Por exemplo, para obter a lista de parâmetros de um módulo use "modinfo -p módulo".

O comando rmmod serve para remover um módulo que não está mais em uso. Também podemos usar o comando modprobe com a opção -r, para se obter um

efeito análogo ao carregamento de módulos com dependências, mas para remover, é claro.

mii-tool

O comando mii-tool serve para verificar e alterar o estado "Media-Independent? Interface" de uma interface. O MII é um recurso utilizado pela maioria dos adaptadores fast ethernet para negociar a velocidade do link e o full/half duplex. Atenção, nem todos os adaptadores suportam esse recurso!

Para verificar o estado de uma interface, basta usar o comando sem parâmetros. Exemplo:

```
# mii-tool
eth0: negotiated 100baseTx-FD, link ok
```

Para forçar que uma placa anuncie somente certas capacidades, ou para desabilitar completamente a auto-negociação e forçar uma certa velocidade, use os parâmetros -A e -F, respectivamente. Exemplo:

Anuncia que a placa entende 100Mbit/s Half e Full Duplex:
```
# mii-tool -A 100BaseTx-FD,100BaseTx-HD eth0
```

Força a placa a entrar no modo 10Mbit/s Full Duplex:
```
# mii-tool -F 10BaseT-FD eth0
```

ifconfig

O ifconfig é o principal comando para configurarmos interfaces no Linux. A sua forma geral é "ifconfig opções | endereço". Consulte a página do manual para ver todas as opções (man ifconfig). Veremos a seguir alguns exemplos:

Para listar as interfaces e seus endereços:

```
# ifconfig
eth1      Link encap:Ethernet  HWaddr 02:60:8C:F1:EB:CF
      inet addr:10.10.10.1  Bcast:10.255.255.255  Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:11 errors:0 dropped:0 overruns:0 frame:0
      TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:858 (858.0 b)  TX bytes:0 (0.0 b)
      Interrupt:5 Base address:0x2440

lo        Link encap:Local Loopback
```

```
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      RX packets:55870 errors:0 dropped:0 overruns:0 frame:0
      TX packets:55870 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:124540531 (118.7 MiB)  TX bytes:124540531 (118.7 MiB)
```

Para configurar e levantar uma interface com um determinado IP:

```
# ifconfig eth0 10.0.0.1 netmask 255.255.255.0 up
```

Para desabilitar uma interface:

```
# ifconfig eth0 down
```

Para configurar um "alias" em uma interface, juntamente com outro endereço IP:

```
# ifconfig eth0:1 192.168.1.1 netmask 255.255.255.0
```

Para mudar a MTU de uma interface:

```
# ifconfig eth0 mtu 1440
```

Para configurar placa com uma conexão ponto-a-ponto:

```
# ifconfig eth0 192.168.2.1 netmask 255.255.255.255 pointopoint 192.168.2.2
```

Para colocar e retirar uma interface do modo "Promíscuo" (a interface aceita pacotes destinados a qualquer IP):

```
# intertace eth0 promisc
# interface eth0 -promisc
```

O ifconfig também pode ser utilizado para se alterar o endereço MAC da placa! Para alterar esse endereço é necessário que a placa esteja inativa. Exemplo:

```
# ifconfig eth0 down
# ifconfig eth0 hw ether 00:11:22:33:44:55
# ifconfig eth0 up
```

Configuração de Rotas

route

O comando route serve para mostrar e manipular a tabela principal de rotas. Quando utilizado sem argumentos ele mostra a tabela de rotas. Exemplo:

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.10.0      *                  255.255.255.0   U       0      0        0     et
h0
default           10.10.10.10  0.0.0.0               UG    0      0        0     et
h0
```

Um parâmetro que pode ser útil ao mostrarmos rotas é o "-n", que faz com que o comando route não tente resolver nomes.

Para inserir uma rota use o comando no formato: route add [-net | -host] target [netmask Nm] [gw Gw] [[dev] If]. Exemplos:

Inserir uma rota para uma rede:

```
# route add -net 192.168.5.0 netmask 255.255.255.0 gw 192.168.1.3
```

Inserir uma rota para um host, utilizando uma determinada interface ponto-a-ponto:

```
# route add -net 192.168.10.1 dev ppp0
```

Para selecionar a rota default:

```
# route add default gw 192.168.1.200
```

Para remover rotas usamos praticamente a mesma sintaxe, mas ao invés do "add" usamos o "del".

Comandos úteis na resolução de problemas

Problemas no nível 2

Quando falamos em roteamento, estamos trabalhando basicamente com a camada 3 (IP) do modelo OSI. No entanto, existem muitos casos em que é necessário conhecer e trabalhar com o nível 2 (enlace), mais especificamente estaremos falando, no nosso caso, de ethernet.

arp

O comando arp serve para listar e modificar a tabela arp do kernel. A tabela arp associa endereços IP à endereços de "hardware", (no caso do ethernet, aos MAC address). Na sua forma mais básica, sem parâmetros, o comando arp simplesmente lista a tabela arp (utilize a opção -n para não resolver nomes), exemplo:

```
# arp
Address HWtype HWaddress Flags Mask Iface
maquina1.teste ether 00:00:0C:95:0B:00 C eth0
maquina2.teste ether 00:60:08:A1:48:A6 C eth0
```

Para remover uma entrada da tabela utilize opção "-d host". exemplo:

```
# arp -d maquina2.teste
```

O comando arp permite também que entradas sejam adicionadas manualmente na tabela. Normalmente isso não é necessário mas em algumas situações poderá ser, por exemplo, quando configuramos uma conexão ponto-a-ponto, ou dial-up. Nesse caso é preciso que a máquina roteadora seja capaz de aceitar pacotes que tenham como destino a outra máquina da conexão. Para adicionarmos uma entrada usamos a opção "-s host hw_address" ou para forçarmos que uma interface responda a requisições arp utilizando o endereço de outra (ou dela mesmo) interface, podemos usar as opções "-i", para definir a interface, e a opção "-D", para dizer de qual interface queremos "copiar" o endereço de

hardware. Exemplo

```
# arp -s 192.168.0.3 -i eth0 -D eth0 pub
```

O exemplo acima faria com que a interface eth0 respondesse a requisições de arp para o endereço 192.168.0.3 utilizando o seu próprio endereço de hardware. A opção pub faz com que a interface passe a "divulgar" esse endereço (e não apenas a aceitar pacotes para ele).

arping

O comando arping serve para enviar requisições arp e/ou ICMP. O comando arping, além de aceitar nome de hosts, endereços IPs e endereços de hardware, pode usar também como origem e/ou destino endereços de broadcast. Por exemplo, se quisermos verificar se um determinado IP está sendo utilizado antes de o configurarmos na placa de rede poderíamos fazer:

```
# arp -0 192.168.0.1
```

O resultado disso é:

```
# tcpdump arp
tcpdump: listening on eth0
14:02:48.327806 arp who-has 192.168.0.1 tell 0.0.0.0
14:02:48.327939 arp reply 192.168.0.1 is-at 0:60:8:a1:48:a6
```

Conectividade e Rotas

Para diagnosticar problemas de conectividade ou de roteamento os comandos a seguir são bastante úteis.

ping

O ping já deve ser um velho conhecido de todos que trabalham com redes de computadores. Ele serve para enviar requisições ICMP de "echo request" e fica escutando pelos pacotes ICMP de "echo reply". No entanto, o ping serve também para mostrar a rota que os pacotes ICMP fazem, tanto na "ida" quanto na "volta". Para tanto, basta utilizar a opção "-R" do ping, como no exemplo abaixo:

```
# ping -c 1 -R 200.Z.Z.21
PING 200.X.X.21 (200.X.X.21) 56(124) bytes of data.
64 bytes from 200.X.X.21: icmp_seq=1 ttl=254 time=5.16 ms
RR:     200.X.X.36
       200.X.Y.21
       200.Z.Z.20
```

```
       200.Z.Z.21
       200.Z.Z.21
       200.X.Y.22
       200.X.X.41
       200.X.X.36

--- 200.Z.Z.21 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.165/5.165/5.165/0.000 ms
```

A opção "-c N" serve para especificar quantos "pings" queremos enviar.

traceroute

O comando traceroute é utilizado para traçar rotas até um determinado IP. Ao contrário do "ping -R" o traceroute resolve os nomes (podendo-se desabilitar isso com a opção "-n"), e é capaz de traçar uma rota mesmo que o host não exista ou esteja inacessível. Obviamente o comando só irá traçar a rota até que um roteador responda com um "unreacheable" ou até o TTL expirar. Exemplo:

```
# traceroute 200.Z.Z.21
traceroute to 200.Z.Z.21 (200.Z.Z.21), 30 hops max, 38 byte packets
 1  maquina1 (200.X.X.41)  2.501 ms  1.289 ms  1.316 ms
 2  maquina2 (200.X.Y.22)  1.246 ms  1.358 ms  1.117 ms
 3  destino (200.Z.Z.21)  1.153 ms  2.126 ms  1.004 ms
```

tracepath

O tracepath é um comando bastante similar ao traceroute, mas mostra o pmtu do caminho. Exemplo:

```
# tracepath 200.Z.Z.21
 1?: LOCALHOST     pmtu 1500
 1:  maquina1 (200.X.X.41)                          4.243ms
 2:  maquina2 (200.X.Y.22)                          4.242ms
 3:  destino  (200.Z.Z.21)                            7.171ms reached
    Resume: pmtu 1500 hops 3 back 3
```

tcpdump

O comando tcpdump serve para fazer um dump do tráfego de rede. Na sua operação normal ele coloca a interface de rede em modo promíscuo e imprime na tela uma breve descrição de todos os pacotes que chegam à interface. O tcpdump foi utilizado no exemplo acima sobre arping para mostrar as requisições

e respostas arp na rede. Nesse documento vamos apenas citar alguns exemplos e deixar a cargo do leitor que olhe no manual do comando a imensa lista de opções que ele é capaz de aceitar.

Para mostrar os pacotes do tipo ICMP que chegam à interface eth0:

```
# tcpdump -i eth0 icmp
```

Para mostrar os pacotes com origem na máquina x.y.z.z:

```
# tcpdump src x.y.z.z
```

Para mostrar pacotes que tenham como origem ou destino a máquina x.y.z.z

```
# tcpdump host x.y.z.z
```

O Comando IP

O comando IP merece um tópico a parte, pois ele resume em apenas um comando funcionalidades do ifconfig, arp e route, além de permitir a manipulação de tabelas de rotas especiais e túneis. A sua forma básica é ip [opções] OBJETO { COMMAND | help }. Onde objeto os principais objetos são: link, addr, neigh, e route. Veremos a seguir em mais detalhes esses objetos. Os objetos tunnel e rule serão vistos em outra ocasião.

Vale lembrar que utilizando simplesmente "ip" ou "ip objeto help" o comando imprimirá um ajuda rápida.

ip link

O objeto link serve para mostrar e manipular opções dos dispositivos de rede. Através dele é possível ativar ou desativar uma interface, mudar os seus endereços de "hardware", opções de arp e multicast e até mesmo "renomear" uma interface. Veremos aqui algumas opções mais utilizadas:

Para mostrar os dispositivos de rede (ao contrário do ifconfig, o ip link pode mostrar dispositivos que estejam inativos também):

```
# ip link show
1: lo:  mtu 16436 qdisc noqueue
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0:  mtu 1500 qdisc pfifo_fast qlen 1000
   link/ether 00:02:55:5d:07:c6 brd ff:ff:ff:ff:ff:ff
```

```
3: eth1:  mtu 1440 qdisc pfifo_fast qlen 1000
   link/ether 11:22:33:44:55:66 brd ff:ff:ff:ff:ff:ff
```

Para ativar uma interface:

```
# ip link set eth0 up
```

Para mudar a mtu de um dispositivo:

```
# ip link set eth0 mtu 1480
```

ip addr

O objeto addr server para listar e manipular os endereços IPv4 / IPv6 das interfaces (virtuais ou não). Ao contrário do comando ifconfig, o comando ip addr é capaz de atribuir e mostrar mais de um endereço IPv4 para a mesma interface. Atenção, o comando ip addr mostra e espera como entrada endereços no formato addr/masklen. Exemplos:

Para mostrar os endereços das interfaces:

```
# ip addr show
1: lo: mtu 16436 qdisc noqueue
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
2: eth0: mtu 1500 qdisc pfifo_fast qlen 1000
  link/ether 00:02:55:5d:07:c6 brd ff:ff:ff:ff:ff:ff
  inet 200.X.X.36/27 brd 200.238.128.63 scope global eth0
```

Para adicionar um endereço em uma interface:

```
# ip addr add 192.168.0.1/24 dev eth0
```

Para se remover a sintaxe é semenhante, mas usamos o del e não o add.

Para configurar um endereço ponto-a-ponto:

```
# ip addr add 192.168.0.2 peer 192.168.0.1
```

ip neigh

O objeto neigh é semelhante em funcionalidade ao comando arp, ou seja, serve para mostrar e manipular a tabela arp. Exemplos:

Para listar a tabela arp associada a uma determinada interface:

```
# ip neigh show dev eth0
200.X.X.41 dev eth0 lladdr 00:00:0c:95:0b:00 nud reachable
200.X.X.40 dev eth0 lladdr 00:60:08:a1:48:a6 nud reachable
```

Para inserir uma entrada na tabela arp:

```
# ip neigh add 10.0.0.1 lladdr 00:11:22:33:44:55 dev eth0
```

Para remover uma entrada basta trocar o "add" por "del".

Para limpar toda a tabela arp associada a uma interface:

```
# ip neigh flush dev eth0
```

ip route

O objeto route é utilizado para mostrar e manipular as tabelas de rota do kernel. Ao contrário do comando route que só manipula a tabela de rotas principal, o ip route consegue manipular todas as tabelas de rotas (veremos em um outro capítulo mais detalhes sobre isso). Exemplos:

Para listar a tabela de rotas:

```
# ip route show
192.168.0.1 dev eth1  proto kernel  scope link  src 192.168.0.2
unreachable 10.20.20.0/24
blackhole 10.40.40.0/24
default via 200.X.X.41 dev eth0
```

Para mostrar uma rota para uma determinada rede ou host:

```
# ip route get 200.X.X.21
200.X.X.21 via 200.Z.Z.41 dev eth0  src 200.Z.Z.36
  cache  mtu 1500 advmss 1460
```

Para adicionar uma rota:

```
# ip route add 10.0.0.0/24 via 192.168.1.200
```

Para deletar rotas basta trocar o "add" por "del".

Para acrescentar uma rota "blackhole" (os pacotes são silenciosamente descartados):

```
# ip route add blackhole 10.0.1.0/24
```

Para acrescentar uma rota "unreachable" (é gerado um ICMP host unreachable):

```
# ip route add unreachable 10.0.2.0/24
```

Automatizando a Configuração de Redes/Rotas

Até aqui mostramos como configurar interfaces, endereços e rotas manualmente. É claro que isso também pode ser feito automaticamente. Cada distribuição Linux coloca seus arquivos de configurações de rede em locais um pouco diferentes. A saber:

RedHat (e Mandrake)

- definação de interfaces /etc/sysconfig/network-scripts/ifcfg-*
- hostname e rota padrão gateway definition /etc/sysconfig/network
- Rotas estáticas /etc/sysconfig/static-routes

Slackware

- Definição de interfaces e rotas /etc/rc.d/rc.inet1

Debian

- Definição de rotas e interfaces /etc/network/interfaces

Como nosso foco é a distribuição Debian, vamos dar uma rápida olhada no arquivo /etc/network/interfaces. Exemplo:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
     address 200.X.X.36
     netmask 255.255.255.224
     network 200.X.X.32
     broadcast 200.X.X.63
     gateway 200.X.X.41
```

Além das opções mostradas, cada interace aceita ainda as opções pre-up, up, down, post-down que podem ser utilizadas para se executar comandos ou scripts, antes de subir a interface, depois de subir, antes de descer e após descer a interface. Assim, por exemplo, se quisermos adicionarmos uma rota ao subir um

determinada interface, poderíamos usar a seguinte linha dentro das opções da interface:

```
iface eth0 inet static
  ...
  up ip route add 10.0.0.0/24 via 200.X.X.Y
```

O Diretório /proc

O sistema de arquivos /proc é um sistema de arquivos virtual. O que signfica que ele não existe fisicamente (O /proc descrito aqui é apenas para o Kernel Linux). Por um sistema de arquivo virtual entendemos como aquele que não deixa rastro em seus HD's, ou seja, este sistema de arquivos está em memória primária. O /proc é criado "on the fly" durante o processo de inicialização do SO.

O /proc contém diferentes estruturas de dados e informações colhidas do kernel. Iremos nos focar aqui somente ao que se relaciona com Internet Protocol version 4 que consta em /proc/sys/net/ipv4.

Abaixo verificamos o conteúdo do /proc/sys/net/ipv4:

dr-xr-xr-x 9 root root 0 2003-10-09 08:49 conf
-rw-r--r-- 1 root root 0 2003-10-09 08:49 icmp_echo_ignore_all
-rw-r--r-- 1 root root 0 2003-10-09 08:49 icmp_echo_ignore_broadcasts
-rw-r--r-- 1 root root 0 2003-10-09 08:49 icmp_ignore_bogus_error_responses
-rw-r--r-- 1 root root 0 2003-10-09 08:49 icmp_ratelimit
-rw-r--r-- 1 root root 0 2003-10-09 08:49 icmp_ratemask
-rw-r--r-- 1 root root 0 2003-10-09 08:49 igmp_max_memberships
-rw-r--r-- 1 root root 0 2003-10-09 08:49 inet_peer_gc_maxtime
-rw-r--r-- 1 root root 0 2003-10-09 08:49 inet_peer_gc_mintime
-rw-r--r-- 1 root root 0 2003-10-09 08:49 inet_peer_maxttl
-rw-r--r-- 1 root root 0 2003-10-09 08:49 inet_peer_minttl
-rw-r--r-- 1 root root 0 2003-10-09 08:49 inet_peer_threshold
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ip_autoconfig
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ip_default_ttl
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ip_dynaddr
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ip_forward
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ipfrag_high_thresh
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ipfrag_low_thresh
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ipfrag_secret_interval
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ipfrag_time
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ip_local_port_range
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ip_nonlocal_bind
-rw-r--r-- 1 root root 0 2003-10-09 08:49 ip_no_pmtu_disc
dr-xr-xr-x 8 root root 0 2003-10-09 08:49 neigh
dr-xr-xr-x 2 root root 0 2003-10-09 08:49 route
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_abort_on_overflow
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_adv_win_scale
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_app_win
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_dsack
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_ecn
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_fack

```
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_fin_timeout
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_frto
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_keepalive_intvl
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_keepalive_probes
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_keepalive_time
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_low_latency
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_max_orphans
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_max_syn_backlog
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_max_tw_buckets
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_mem
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_orphan_retries
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_reordering
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_retrans_collapse
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_retries1
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_retries2
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_rfc1337
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_rmem
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_sack
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_stdurg
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_synack_retries
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_syn_retries
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_timestamps
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_tw_recycle
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_tw_reuse
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_window_scaling
-rw-r--r-- 1 root root 0 2003-10-09 08:49 tcp_wmem
```

Iremos chamar esses arquivos também de varáveis ipsysctl. Para atribuir novos valores para essas váriaveis existem pelo menos 2 formas:

- Através do comando sysctl, que já está diposnível por padrão na maioria das distribuições Linux.

O comando sysctl é um pouco mais complexo que o sistema de arquivos /proc, porém através de um arquivo de configuração é mais facil manter uma lista de alteração que queremos fazer.

- Diretamente no sistema de arquivos /proc

Através do sistema de arquivos /proc é mais simples quando desejamos pequenas modificações.

A aplicação sysctl pode ser utilizada para setar variáveis através da linha de comando ou ainda através de um arquivo de configuração.

Para saber a lista de todas as possíveis variáveis usamos o comando:

```
# sysctl -a.
```

Isto irá listar todas as variáveis e seus valores separados pelo sinal de igual "=".

Para saber o valor de uma varável usamos:

```
# sysctl net/ipv4/conf/all/rp_filter
```

Para atribuir um novo valor a variável usamos:

```
# sysctl -w net/ipv4/conf/all/rp_filter=0
```

O que mostramos acima com o comando sysctl pode ser feito também diretamente no /proc da seguinte maneira:

As variavéis que podem ser alteradas estão no diretório /proc/sys. As atribuicões que nos intereressam neste tutorial estão em /proc/sys/net/ipv4.

Para saber o conteúdo usamos o comando ls:

```
# ls
```

Para saber o valor usamos o comando cat:

```
# cat  /proc/sys/net/ipv4/conf/all/rp_filter
```

Para atribuir um novo valor usamos o comando echo:

```
# echo 1 > /proc/sys/net/ipv4/conf/all/rp_filter
```

Principais variáveis

Agora que já sabemos consultar e atribuir valores veremos as principais variáveis:

- ip _default_ttl

Conta ao kernel qual o Time to Live que será colocado nos pacotes que deixarão o host. O valor default é 64.

- ip_forward

É o que realmente queremos neste tutorial, que nosso unix-router encaminhe IP entre as diversas interfaces. O valor é um booleano e o default desta variável é 0 (desabilitado), para habilitar devemos trocar para 1.

- ip_no_pmtu_disc

O PMTU (Path Maximum Transfer Unit) discovery é um valor booleano também que pode ser habilitado para descobrir o MTU entre este host e um end-host. O PMTU e o MTU são coisas diferentes. O MTU conta ao kernel sobre a máxima unidade de transferncia para suas conexôes, mas não sobre toda conexão até o end-host. O PMTU tenta descobrir o MTU para um end-host incluindo todos os hops no caminho.
O valor default é TRUE (1) ou seja, o PMTU discovery está desabilitado.

- ip_nonlocal_bind

Permite que processos locais façam o bind com um endereço IP não-local. Isto é útil em alguns casos de snnifers para um determinado host. O valor é booleano e o default é 0.

- icmp_echo_ignore_all

Se este valor é 1 (verdadeiro) o kernel irá ignorar todas os ICMP echo requests.

- icmp_echo_ignore_broadcasts

Esta variável funciona como a icmp_echo_ignore_all exceto que irá ignorar somente mensagens ICMP enviadas para o endereço de broadcast ou multicast. O valor default é falso (0).

- icmp_ratelimit

É o taxa máxima na qual o kernel irá gerar mensagens ICMP do tipo especificado em icmp_ratemask. O valor é o número de jiffies (1 jiffy = 1/100 sc) entre o envio de duas mensagens semelhantes. O valor default é 100, o que significa qua um pacote ICMP pode ser enviado em 100 jiffies. O valor 0 (zero) significa não limitado envio de ICMP's.

- icmp_ratemask

Atribui a máscara do qual tipos de ICMP's serão limitados através da variável icmp_ratelimit. Este valor é a soma de 2^n, onde n é o cada tipo especificado de ICMP (como especificado no cabeçalho de arquivos do kernel). Estes valores estão disponiveis no arquivo netinet/ip_icmp.h (geralmente em /usr/include/netinet/ip_icmp.h). Para mais informações de tipo olhe a RFC792 - ICMP.
Alguns tipos:

0 /* Echo Reply */
3 /* Destination Unreachable */
4 /* Source Quench */
5 /* Redirect (change route) */
8 /* Echo Request */

11 /* Time Exceeded */
12 /* Parameter Problem */
13 /* Timestamp Request */
14 /* Timestamp Reply */
15 /* Information Request */
16 /* Information Reply */
17 /* Address Mask Request */
18 /* Address Mask Reply */

O valor default desta variável é 6168, o que significa que ICMP Destination Unreachable, ICMP Source Quench, ICMP Time Exceeded and ICMP Parameter Problem is in the mask. ICMP Destination Unreachable é igual a 3, ICMP Source Quench é igual a 4, ICMP Time Exceeded é igual a 11 e ICMP Parameter Problem é igual a 12. Desta forma, estão habilitados, ou seja:

2^3 + 2^4 + 2^11 + 2^12

Variáveis importantes no /conf

Algumas variáveis estão no diretório conf/ o qual podemos verificar abaixo:

dr-xr-xr-x 2 root root 0 Oct 9 11:09 all
dr-xr-xr-x 2 root root 0 Oct 9 11:09 default
dr-xr-xr-x 2 root root 0 Oct 9 11:09 eth0
dr-xr-xr-x 2 root root 0 Oct 9 11:09 eth1
dr-xr-xr-x 2 root root 0 Oct 9 11:09 eth2
dr-xr-xr-x 2 root root 0 Oct 9 11:09 lo

Observe que muitos dos arquivos se referem a interfaces do host. Estes diretórios permitem que sejam atribuídos diferentes valores para cada interface, ou o mesmo valor para o caso de se atribuir a uma variável do diretório all. O diretório default irá mudar os valores padrões, mas isso não mudará os valores já atribuídos, mas irá mudar apenas os valores de novos dispositivos que surgirem.

- accept_redirects

Esta variável irá contar ao kernel quando ele deve aceitar ICMP Redirects ou não. ICMP Redirects são usados para contar a um host que existe um caminho melhor para enviar pacotes para um específico host ou rede. Este valor é verdadeiro (1) por default.

- accept_source_route

Esta variável conta ao kernel se ele permite pacotes roteados pela origem. Este valor por default é verdadeiro (1).

- arp_filter

A variável arp_filter conta ao kernel se um endereço IP deve ser ligado com um endereço ARP específico ou não. O kernel decide, se responde para um pacote específico entrando numa interface, se ele envia a resposta pelo mesmo endereço da interface ou não. Se o arp_filter é verdadeiro (1) isso é o que acontece. O valor default é falso (0).

- forwarding

Esta variável habilita o encaminhamento de IP para dispositivos específicos. Isto é util para habilitar o encaminhamento entre 2 interfaces e uma terceira deixar desabilitado. O valor default desta variável é o mesmo valor de ipv4/ip_forward, assim se o valor atribuído a ip_forward for 1 todas as variáveis forwarding serão 1 também.

- log_martians

Esta variável irá permitir o log de todos os pacotes que contém endereço impossível. Um endereço impossível pode significar um endereço que nao sabemos entrar em contato. O valor default é falso (0).

- mc_forwarding

Isto permite o roteamento multicast para um dispositivo específico. Para isto, obviamente, o kernel precisa ter suporte para multicast. Esta opção por default é (0). Observe que para apenas escutar por pacotes multicast, esta opção não é necessária. Isto só é necessário para o encaminhamento multicast sobre o host.

- proxy_arp

Atribui Proxy ARP para uma interface. O Proxy ARP deve ser habilitado nas interfaces que se deseja para responder ARP queries. O valor default é falso (0).

- rp_filter

Esta variável atribui o filtro de caminho reverso (reverse path - rp) para uma interface. O RP serve para validar que o atual endereço de origem usados por alguns pacotes estejam correlacionados com uma tabela de rotas e que pacotes com este IP de origem são supostamente respondidos através da mesma interface. (Verificar os comportamentos na RFC 1812).

- secure_redirects

Esta variável permite redirecionamento seguro. Se não tiver habilitado o kernel irá aceitar ICMP redirect de qualquer host. Porém se estiver habilitado todos os ICMP redirects serão somente aceitos de gateways listados na lista de gateway default. Com isto podemos nos livrar de redirecionamentos ilegais que podem ser maliciosos. O valor default é verdadeiro (1). Note que esta variável pode ser sobrescrita pela variável shared_media, desta forma se você habilitar esta variável, habilite tambem a shared_media.

- shared_media

Esta variável permite informar ao kernel se a interface física conectada é compartilhada ou não. Isto é se diversas e diferentes redes IP com diferentes máscaras operarem sobre uma mesma mídia ou não. O objetivo principal dessa variável é informar ao kernel se deve enviar ICMP redirects para uma rede específica ou não. O valor default é verdadeiro (1).

Outras variáveis:

- /proc/sys/net/ipv4/route/flush

Esta variável não contém um dado real, então tentativas de leitura retornam um erro (tente ver em /proc/net/rt_cache). Mas qualquer modificação no conteúdo dessa variável será feito um flush em todo routing cache. Assim se você sabe que uma rota foi alterada em algum lugar, pode-se simplesmente fazer um echo de algo para este arquivo e a route cache será esvaziada.

VLAN 802.1q no Linux

Iremos tratar aqui de vlans no padrão IEEE 802.1q.

Uma implementação do protocolo VLAN 802.1q permite ter diversas Virtual LANs sobre um simples cabo ethernet. Isto é possivel adicionando um campo no header ethernet.
No linux conseguiremos criar várias interfaces ethernet com essa feature, essas interfaces podem ser manipuladas quase como se fossem uma interface ethernet comum, permitindo bridging, firewalling e é claro trafégo IP.

Características de VLAN 802.1q no Linux

- Suporta a especificação de VLAN 802.1q
- Implementa suporte para uma funcionalidade não-padrão: VLAN MAC-based
- Suporta até 4096 VLAN's por interface ethernet
- Boa escalabilidade.
- Suporte para Multicast
- Permite alterar o MAC da VLAN
- Suporte para múltiplas convenções de nomes e ajustável runtime

Instalando o suporte a VLAN 802.1q

Para se ter essa feature além de uma ferramenta para manipular as vlan, é necessário também ter suporte no kernel.
Nos kernel's com versão após a 2.4.14, basta apenas que esteja compilado (built-in ou como módulo) para os anteriores a esse é necessário aplicar um patch.
Tanto a ferramenta para manipular as vlans (vconfig) como os patchs estão no CVS do desenvolvedor: http://www.candelatech.com/~greear/vlan.html
Para usuários Debian basta instalar:

```
# apt-get install vlan
```

É importante lembrar de subir o módulo 802.1q

```
# modprobe 8021q
```

Manipulando VLAN's

Para criar uma vlan iremos utilizar o comando vconfig, como segue:

```
# vconfig add eth0 123
```

Isto irá criar uma VLAN com ID igual a 123. Para adicionar um IP para essa nova vlan-device, fazemos:

```
# ifconfig eth0.123 192.168.2.1 netmask 255.255.255.252 up
```

Pode se verificar informações referentes as VLAN em /proc/net/vlan/*

Caso se deseje remover a VLAN, fazemos:

```
# vconfig rem eth0.123
```

Existem alguns formatos para as vlan-device:

Name Type	Formato
VLAN_PLUS_VID	vlan0123
VLAN_PLUS_VID_NO_PAD	vlan123
DEV_PLUS_VID	eth0.0123
DEV_PLUS_VID_NO_PAD	eth0.123

O formato padrão é o DEV_PLUS_VID_NO_PAD, mas caso se deseja trocar o formato fazemos da seguinte forma:

```
# vconfig set_name_type VLAN_PLUS_VID_NO_PAD
```

/etc/network/interfaces (vlan)

O arquivo no formato interfaces contém informações sobre as interfaces de redes que serão manipulados pelos aplicativos ifup e ifdown.
Podemos adicionar informações nesse arquivo para manipular também vlan-devices. Existem pelo menos 4 formas de adicionarmos vlan's. Observe no exemplo abaixo as 4 formas de se adiconar uma VLAN com ID 123

```
iface eth0.123 inet static
   address 192.168.2.1
   netmask 255.255.255.252

iface vlan123 inet static
   vlan-raw-device eth0
   address 192.168.2.1
   netmask 255.255.255.252
```

```
iface eth0.0123 inet static
   address 192.168.2.1
   netmask 255.255.255.252

iface vlan0123 inet static
   vlan-raw-device eth0
   address 192.168.2.1
   netmask 255.255.255.252
```

Abaixo estão algumas opções extras:

- vlan-raw-device devicename

Indica a interface na qual a vlan será criada. Isto é ignorado no caso do nome da vlan possuir o nome da interface

- ip-proxy-arp 0|1

Habilita ou não o proxy-arp para a interface específica. O valor é buleano, ou seja, 1 para habilitar e 0 para desabilitar.

- ip-rp-filter 0|1|2

Habilita o filtro de caminho reverso.

- hw-mac-address mac-address

Irá setar um novo mac-address para a vlan-device. Isto funciona em qualquer dispositivo que permite setar o endereço do hardware com o comando ip.

Roteamento Avançado

Iremos tratar aqui de algumas opções que dispomos para fazer uma configuração mais avançada em nosso roteador, como túneis e a criação de novas tabelas de roteamento. Muito do que veremos aqui exige que o kernel esteja compilado com IP: advanced router e IP: policy routing

Túneis

Túneis podem ser utilizados para várias coisas interessantes, mas isso pode ser uma dor de cabeça quando mal configurado, estes problemas ainda podem se estender a detalhes do nível 2 (MTU), já que o tunelamento adiciona um overhead do novo cabeçalho.
Existem algum tipos de túneis suportados pelo Linux, mas com certeza os mais utilzados são o IPIP e o túnel GRE.

Iremos ver aqui uma configuração de túnel utilizando o comando ifconfig e outa configuração usando o comando ip que é mais poderoso para novas funcionalidades.

Túnel IP dentro de IP (IP in IP)

Para podermos utilizar o tunelamento de IP in IP devemos ter o kernel preparado para isso. Caso esteja compilado como módulo devemos "subir" o módulo da seguinte forma:

```
# modprobe ipip
```

Imagine que temos uma rede A, uma rede B e no caminho dessas uma rede C (como a internet). Podemos fazer um túnel entre a rede A e rede B, da seguinte forma:

Rede A = 10.0.0.0/24 e o gateway tem uma interface local 10.0.0.1 e na Internet com o IP 172.10.11.12
Rede B = 192.168.0.0/24 e o gateway tem uma interface local 192.168.0.1 e na Internet com o IP 172.22.33.44

No roteador da rede A, devemos fazer o seguinte:

```
# ifconfig tunl0 10.0.0.1 pointopoint 172.22.33.44
# route add -net 192.168.0.0 netmask 255.255.255.0 dev tunl0
```

No roteador da rede B, devemos fazer o seguinte:

```
# ifconfig tunl0 192.168.0.1 pointopoint 172.10.11.12
# route add -net 10.0.0.0 netmask 255.255.255.0 dev tunl0
```

E com isso finalizamos o túnel.

Túnel GRE

GRE é um protocolo de tunelamento que originalmente foi desenvolvido pela Cisco e que pode fazer algumas coisas a mais do que o IP in IP, por exemplo um túnel para um tráfego IPv6. Tendo suporte no kernel do Linux para isso, se tiver compilado como módulo, devemos "subir" o modulo da seguinte forma:

```
# modprobe ip_gre
```

Imagine que temos a mesma situação acima, ou seja:

Rede A = 10.0.0.0/24 e o gateway tem uma interface local 10.0.0.1 e na Internet com o IP 172.10.11.12
Rede B = 192.168.0.0/24 e o gateway tem uma interface local 192.168.0.1 e na Internet com o IP 172.22.33.44

No roteador da rede A devemos fazer o seguinte:

```
# ip tunnel add redeb mode gre remote 172.10.11.12 local 172.22.33.44 ttl 255
# ip link set redeb up
# ip route add 192.168.0.0/24 dev redeb
```

No roteador da rede B, fazemos assim:

```
# ip tunnel add redea mode gre remote 172.22.33.44 local 172.10.11.12 ttl 255
# ip link set redea up
# ip route add 10.10.0.0/24 dev redea
```

Está pronto o túnel, podemos fazer desta mesma forma (usando o comando ip) os túneis IP in IP, apenas mudando o mode para ipip.
É aconselhavel também adicionar um endereço IP (de preferência /30) para o túnel (bom para teste de conectividade) como descrito abaixo:

```
# ip addr add 10.0.2.1/30 dev redea
```

Caso se deseje deletar o tunel:

```
# ip tunnel delete redea
```

Caso se deseje modificar alguma configuração:

```
# ip tunnel change redea ...
```

Caso se deseje verificar informações sobre o tunel:

```
# ip tunnel show redea
```

Tabelas de roteamento

Desde as versões do Linux-2.x pode-se ter múltiplas tabelas de roteamento no kernel. Estas tabelas são identificadas por um número variando de 1 a 255 ou por um nome de acordo com o arquivo /etc/iproute2/rt_tables.
Com novas tabelas pode-se criar uma flexível estrutura para implementar uma Politica de Roteamento.

Abaixo temos o conteúdo de /etc/iproute2/rt_tables.

```
#
# reserved values
#
255     local
254     main
253     default
0       unspec
#
# local
#
1       inr.ruhep
```

- 255 local: Tabela de roteamento especial mantida pelo kernel. Usuários não podem adicionar novas entradas nessa tabela
- 254 main: A principal tabela de roteamento. É onde ficam as rotas quando adicionamos com o comando route ou ip route.
- 253 default: Outra tabela especial.
- 0 unspec: Operações sobre esta tabela refletem em todas as tabelas
- 1 inr.ruhep: Isto é um exemplo indicando que a tabela 1 tem o nome inr.ruhep.

Entradas na Tabela de Roteamento.

Cada tabela de roteamento pode ter várias entradas. Abaixo temos alguns tipos de rotas que podem ser adicionadas com o comando ip route:

- unicast: Uma rota unicast é a mais comum na tabela. Isto é tipicamente um rota para uma rede de destino. Se o tipo de rota não é especificado é assumido como sendo unicast. Ex:

```
# ip route add unicast 192.168.0.0/24 via 192.168.100.5
# ip route add default via 193.7.255.1
# ip route add unicast default via 206.59.29.193
# ip route add 10.40.0.0/16 via 10.72.75.254
```

- broadcast: Esta rota é usada pela camada de link de dispositivos (placas Ethernet) o qual suportam a notação de endereço broadcast. Este tipo de rota é usado somente na tabela local e é tipicamente manuseado pelo kernel. Ex:

```
# ip route add table local broadcast 10.10.20.255 dev eth0 proto kernel scope link
src 10.10.20.67
# ip route add table local broadcast 192.168.43.31 dev eth4 proto kernel scope link
src 192.168.43.14
```

- local: O kernel irá adicionar entradas para a tabela de roteamento local quando endereços IP são adicionados para uma interface. Isto signigica que os IP's estão no próprio host.

```
# ip route add table local local 10.10.20.64 dev eth0 proto kernel scope host src 1
0.10.20.67
# ip route add table local local 192.168.43.12 dev eth4 proto kernel scope host src
192.168.43.14
```

- nat: Esta rota é adicionada para o kernel na tabela de roteamento local, quando o usuário tenta configurar stateless NAT. (Re-escreve o destino do pacote) Ex:

```
# ip route add nat 193.7.255.184 via 172.16.82.184
# ip route add nat 10.40.0.0/16 via 172.40.0.0
```

- unreachable: Quando um requisição para uma decisão de roteamento retorna um destino com rota do tipo unreachable, um ICMP unreachable é gerado e retornado para o endereço de origem. Ex:

```
# ip route add unreachable 172.16.82.184
# ip route add unreachable 192.168.14.0/26
# ip route add unreachable 209.10.26.51
```

- prohibit: Análogo ao unreachable mas gera um ICMP prohibit. Ex:

```
# ip route add prohibit 10.21.82.157
# ip route add prohibit 172.28.113.0/28
# ip route add prohibit 209.10.26.51
```

- blackhole: Um pacote que casa com uma rota do tipo blackhole é descartado. Ex:

```
# ip route add blackhole default
# ip route add blackhole 202.143.170.0/24
# ip route add blackhole 64.65.64.0/18
```

- throw: Este tipo de rota é conveniente, quando deseja-se que uma consulta na tabela de roteamento falhe, fazendo com que o pacote continue sendo analizado no RPDB. Ex:

```
# ip route add throw 10.79.0.0/16
# ip route add throw 172.16.0.0/12
```

Routing Policy Database (RPDB)

O Banco de Dados da Política de Roteamento (RPDB) controla a ordem na qual o kernel faz sua busca através das tabelas de roteamento. Cada regra possui uma prioridade, e as regras são examindadas sequencialmente da regra 0 até a regra 32767. Para construirmos a nossa politica iremos utilzar as regras criadas pelo comando ip rule e estas usam strings semelhantes a que vimos acima para especificar rotas.

- unicast: Uma entrada unicast é o tipo mais comum. Este tipo de regra simplesmente faz com que o kernel referencie uma tabela de roteamento na busca de uma rota. Ex:

```
# ip rule add unicast from 192.168.100.17 table 5
# ip rule add unicast iif eth7 table 5
# ip rule add unicast fwmark 4 table 4
```

- nat: A regra do tipo nat é requerida para corrigir operções de NAT stateless. (Re-escreve a origem do pacote)

```
# ip rule add nat 193.7.255.184 from 172.16.82.184
# ip rule add nat 10.40.0.0 from 172.40.0.0/16
```

- unreachable: Qualquer pacote casando com a regra irá fazer com que o kernel gere um ICMP unreachable.

```
# ip rule add unreachable iif eth2 tos 0xc0
# ip rule add unreachable iif wan0 fwmark 5
# ip rule add unreachable from 192.168.7.0/25
```

- prohibit: Qualquer pacote casando com a regra irá fazer com que o kernel gere um ICMP prohibit.

```
# ip rule add prohibit from 209.10.26.51
# ip rule add prohibit to 64.65.64.0/18
# ip rule add prohibit fwmark 7
```

- blackhole: Qualquer pacote casando com a regra de blackhole fará com que o pacote seja dropado.

```
# ip rule add blackhole from 209.10.26.51
# ip rule add blackhole from 172.19.40.0/24
# ip rule add blackhole to 10.182.17.64/28
```

Como fazer roteamento pela origem?

Agora que já sabemos os tipos de rotas e como inserí-las numa tabela, além de sabermos criar novas tabelas de roteamento e ainda manusearmos o processo de roteamento do RPDB, podemos fazer uma experiência, tentando fazer um roteamento pela origem.

Imagine que tudo que tenha origem na rede 10.11.12.0/24 queremos que seja roteado pela origem para um determinado local o qual não é a rota default do gateway.
Se não fizermos nada os pacotes com origem na rede 10.11.12.0/24 possivelmente casarão com a rota default ou com alguma rota mais específica para o destino da tabela main.

A primeira coisa que precisamos fazer para rotear pela origem é criar uma nova tabela de roteamento conforme segue:

```
# echo 200 vialink2 >> /etc/iproute2/rt_tables
```

Após isso adicionamos uma regra para informarmos que tudo que tenha origem na rede 10.11.12.0/24 utilize a tabela de roteamento chamada vialink2:

```
# ip rule add from 10.11.12.0/24 table vialink2
```

Agora que já criamos a regra, precisamos colocar uma rota default, esta é a rota que indica para onde queremos enviar os pacotes com origem na rede 10.11.12.0/24:

```
# ip route add default via 192.168.0.253  table vialink2
```

Feito isso basta apenas um flush e tudo deverá estar funcionando:

```
# ip route flush cache
```

Com o manuseamento correto do RPDB, várias tabelas de roteamento e diversos tipos de rotas, podemos fazer muitas coisas interessantes.

BGP e OSPF com o GNU Zebra / Quagga

O GNU Zebra é um software livre que gerencia protocolos de roteamento baseados em TCP/IP. O Zebra suporta BGP, RIP e OSPF
O Zebra foi idealizado pelo Kunishiro Ishiguro e após isso surgiu uma ramificação chamada Zebra-pj, que recentemente foi rebatizado de Quagga. A arvore de desenvolvimento do Quagga tem o objetivo de ser mais envolvida com a comunidade do que a do modelo centralizado do GNU Zebra.

Por que Quagga?

Quagga é uma espécie de Zebra que foi extinta porque comia muita grama. Mas quagga é utilizando, especialmente na Africa, para nomear qualquer tipo de Zebra.
O nome Quagga parece ter sido surgido do choro do animal "kwa-ha-ha, kwa-ha-ha," rapidamente repetido.

Arquitetura do Quagga

O Quagga possui um core daemon chamado zebra que atua como uma camada de abstração entre os Unix Kernel e Zserv API over a Unix ou TCP Stream to Quagga clients.

Os Zserv clients são:

- ospfd: Implementação do OSPFv2
- ripd: Implementação do RIPv1 e v2
- ospf6d: Implementação do OSPFv3 (IPv6)
- ripngd: Implementação do RIPv3 (IPv6)

- bgpd: Implementação do BGPv4+, incluindo suporte para IPv6 e multicast.

Instalando o GNU Zebra / Quagga

Para instalar o quagga ou o zebra não tem segredo, costuma ser como qualquer outro software, ou seja:

1) Pegar o Software:
- Zebra em:http://www.zebra.org/
- Quagga em: http://www.quagga.net/

2) Configurar a instalação
Executando o configure pode-se habilitar e desabilitar várias features assim como customizar a instação, executando um:

```
# ./configure --help
```

temos uma lista das opções diposníveis.
Nesse passo também é feita a checagem de dependencias. Good Luck!

3) Contruir o Software

```
# make
```

4) Instalar o software

```
# make install
```

Observe após a instalação as novas entradas no /etc/services

zebrasrv 2600/tcp # zebra service
zebra 2601/tcp # zebra vty
ripd 2602/tcp # RIPd vty
ripngd 2603/tcp # RIPngd vty
ospfd 2604/tcp # OSPFd vty
bgpd 2605/tcp # BGPd vty
ospf6d 2606/tcp # OSPF6d vty

Instalação no Debian

Para usuários do sistema Debian / GNU Linux a instalação pode ser feita através do comando apt-get. Na versão woody (stable) do Debian ainda não há versões

do Quagga, portanto deve-se instalar o zebra, como segue:

```
# apt-get install zebra
```

Para as demais versões do Debian, sarge (testing) e sid (unstable), já há versões do Quagga, portanto pode-se instalar da seguinte forma:

```
# apt-get install guagga
```

Iremos tratar daqui para frente para o formato de configuração usado no Debian.

Arquivos de configuração

Podemos observar após a instalação também os arquivos de configuração para cada daemon. Existem 5 daemons de roteamento:

- ripd, ripngd, ospfd, ospf6d, bgpd

E um daemon gerenciador:

- zebra

Os arquivos de configuração para cada daemon geralmente ficam em /usr/local/etc/, mas isso é de acordo de como foi configurado na instalação. Para pacotes de distribuição como o Debian os arquivos estão localizados em /etc/quagga ou /etc/zebra.

Abaixo uma lista dos arquivos de configuração:

bgpd.conf - Arquivo padrão de configuração do bgpd
daemons - Arquivo contendo opções para iniciar os daemos
ospf6d.conf - Arquivo padrão de configuração do ospfv3
ospfd.conf - Arquivo padrão de configuração do ospfv2
ripd.conf - Arquivo padrão de configuração do rip
ripngd.conf - Arquivo padrão de configuração do ripv3
vtysh.conf - Arquivo padrão de configuração de um nova shell integrada
zebra.conf - Arquivo de configuração do gerenciador

O arquivo daemons

Este arquivo conta ao initscript quais daemons devem iniciar. O formato é:
daemon=(yes|no|priority)

Por exemplo, caso deseja-se iniciar os daemons zebra e bgp apenas o arquivo

deve estar assim:

zebra=yes
bgpd=yes
ospfd=no
ospf6d=no
ripd=no
ripngd=no

O initscript para tratar os daemos é:

/etc/init.d/quagga start|stop|restart|priority

O daemon zebra

o arquivo zebra.conf configura o gerenciador zebra, que controla os outros módulos. Abaixo temos uma configuração mínima:

hostname unixrouter
password zebra
enable password zebra
log file /var/log/quagga/zebra.log

Após iniciar o daemon pode-se ter um sessão interativa através de um telnet:

```
# telnet 127.0.0.1 zebra
```
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is ']'.

Hello, this is quagga (version 0.96.2).
Copyright 1996-2002 Kunihiro Ishiguro.

User Access Verification

Password:
Router> en
Password:
Router#

```
configure  Configuration from vty interface
copy       Copy configuration
debug      Debugging functions (see also 'undebug')
disable    Turn off privileged mode command
end        End current mode and change to enable mode.
```

```
exit       Exit current mode and down to previous mode
help       Description of the interactive help system
list       Print command list
no         Negate a command or set its defaults
quit       Exit current mode and down to previous mode
show       Show running system information
terminal   Set terminal line parameters
who        Display who is on vty
write      Write running configuration to memory, network, or terminal
```
Router#

Com o zebra podemos verificar a tabela de rotas do kernel, assim como colocar rotas estáticas e configurar interfaces.

Abaixo temos um exemplo de atribui um IP para uma interface, adicionar ums rota estática e consultar a tabela de rotas.

unixrouter# conf t
unixrouter(config)# interface eth1
unixrouter(config-if)# ip address 192.168.0.2/30
unixrouter(config-if)# quit
unixrouter(config)# ip route 10.10.10.10/24 192.168.0.1
unixrouter(config)# write
Configuration saved to /etc/quagga/zebra.conf
unixrouter(config)# exit
unixrouter# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
```
    B - BGP, > - selected route, * - FIB route
```

S>* 10.10.10.0/24 [1/0] via 192.168.0.1, eth1
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.0.0/30 is directly connected, eth1
unixrouter#

Não é necessário ter este daemon executando, só precisamos utiliza-lo caso se deseje que ele ensine as rotas para o kernel, mas no caso de um looking glass, por exemplo, isto não é preciso.

O daemon bgpd

O arquivo bgpd.conf é o arquivo de configuração do bgp, abaixo temos um arquivo com configuração mínima:

hostname unixrouter

password zebra
log file /var/log/quagga/bgpd.log

Após iniciar o daemon pode-se também ter um sessão interativa através de um telnet:

```
# telnet 127.0.0.1 bgpd
```
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is ']'.

Hello, this is quagga (version 0.96.2).
Copyright 1996-2002 Kunihiro Ishiguro.

User Access Verification

Password:
unixrouter> en
Password:
unixrouter#

```
clear      Reset functions
configure  Configuration from vty interface
copy       Copy configuration
debug      Debugging functions (see also 'undebug')
disable    Turn off privileged mode command
end        End current mode and change to enable mode.
exit       Exit current mode and down to previous mode
help       Description of the interactive help system
list       Print command list
no         Negate a command or set its defaults
quit       Exit current mode and down to previous mode
show       Show running system information
terminal   Set terminal line parameters
undebug    Disable debugging functions (see also 'debug')
who        Display who is on vty
write      Write running configuration to memory, network, or terminal
```
unixrouter#

Abaixo temos um exemplo de configuração de uma sessão BGP, assim como a sumarização das sessões e uma consulta da tabela de rotas BGP.

unixrouter# conf t
unixrouter(config)# router bgp 65002
unixrouter(config-router)# bgp router-id 192.168.0.2

```
unixrouter(config-router)# network 20.10.0.0/16
unixrouter(config-router)# network 20.20.0.0/16
unixrouter(config-router)# network 20.30.0.0/16
unixrouter(config-router)# network 20.40.0.0/16
unixrouter(config-router)# neighbor 192.168.0.1 remote-as 65001
unixrouter(config-router)# quit
unixrouter(config)# wr
Configuration saved to /etc/quagga/bgpd.conf
unixrouter(config)# exit
unixrouter# show ip bgp summary
BGP router identifier 192.168.0.2, local AS number 65002
2 BGP AS-PATH entries
0 BGP community entries

Neighbor V AS MsgRcvd? MsgSent TblVer? InQ OutQ Up/Down State/PfxRcd
192.168.0.1 4 65001 6 10 0 0 0 00:01:57 4

Total number of neighbors 1

unixrouter# sh ip bgp
BGP table version is 0, local router ID is 192.168.0.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf? Weight Path
*> 10.10.0.0/16     192.168.0.1              0             0 65001 i
*> 10.20.0.0/16     192.168.0.1              0             0 65001 i
*> 10.30.0.0/16     192.168.0.1              0             0 65001 i
*> 10.40.0.0/16     192.168.0.1              0             0 65001 i
*> 20.10.0.0/16     0.0.0.0                  0         32768 i
*> 20.20.0.0/16     0.0.0.0                  0         32768 i
*> 20.30.0.0/16     0.0.0.0                  0         32768 i
*> 20.40.0.0/16     0.0.0.0                  0         32768 i

Total number of prefixes 8
unixrouter#
```

O daemon ospfd

O arquivo ospf.conf é o arquivo de configuração do ospf, baixo temos um arquivo com configuração mínima:

```
hostname unixrouter
password zebra
log file /var/log/quagga/ospf.log
```

Após iniciar o daemon pode-se também ter um sessão interativa através de um telnet:

```
# telnet 127.0.0.1 ospfd
```
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is ']'.

Hello, this is quagga (version 0.96.2).
Copyright 1996-2002 Kunihiro Ishiguro.

User Access Verification

Password:
unixrouter> en
Password:
unixrouter#

```
clear      Reset functions
configure  Configuration from vty interface
copy       Copy configuration
debug      Debugging functions (see also 'undebug')
disable    Turn off privileged mode command
end        End current mode and change to enable mode.
exit       Exit current mode and down to previous mode
help       Description of the interactive help system
list       Print command list
no         Negate a command or set its defaults
quit       Exit current mode and down to previous mode
show       Show running system information
terminal   Set terminal line parameters
who        Display who is on vty
write      Write running configuration to memory, network, or terminal
```
unixrouter#

Abaixo temos o exemplo de configuração de um processo OSPF, assim como a consulta de rotas:

```
unixrouter# conf t
unixrouter(config)# router ospf
unixrouter(config-router)#  redistribute connected
unixrouter(config-router)#  redistribute static
unixrouter(config-router)#  network 192.168.0.0/30 area 0.0.0.0
unixrouter(config-router)# wr
Configuration saved to /etc/quagga/ospfd.conf
unixrouter(config-router)# exit
unixrouter(config)# exit
unixrouter# show ip ospf route
========= OSPF network routing table ======
```

```
N    192.168.0.0/30        [10] area: 0.0.0.0
                directly attached to eth1

====== OSPF router routing table =======
R    192.168.0.1           [10] area: 0.0.0.0, ASBR
                via 192.168.0.1, eth1

====== OSPF external routing table ========
N E2 172.173.0.0/16        [10/20] tag: 0
                via 192.168.0.1, eth1

unixrouter#
```

Multicast

Iremos tratar aqui de algumas implementações de protocolos de roteamento multicast para Linux. É importante lembrarmos que esses daemons precisam de suporte no kernel do Linux.

Antes de entrarmos nos detalhes de cada daemon que implementa os diversos algoritmos de roteamento multicast iremos ver alguns comandos que são uteis.

ip maddress

Com o comando abaixo é possivel verficar os grupos sobre as interfaces

```
# ip maddr show  eth0
```
2: eth0

```
    link  01:00:5e:00:00:01

    inet  224.0.0.1
```

O próximo comando permite que se anexe um grupo a um endereço na camanda de enlace de uma interface.

```
# ip maddress add 01:00:5e:60:e0:e0 dev eth0
```

Podemos retirar a entrada acima através do comando:

```
# ip maddress del 01:00:5e:60:e0:e0 dev eth0
```

ip mroute

Com o comando abaixo podemos consultar as entradas na cache criadas pelos daemons (pimd ou mrouted)

```
# ip mroute show
```

Mrouted

O mrouted é uma implementação do protocolo DVMRP (Distance-Vector Multicast Routing Protocol), conforme especificado na RFC-1075.
Para fornecer suporte a esse protocolo o kernel deve estar compilado com algumas características:

```
CONFIG_IP_MULTICAST=y
CONFIG_IP_MROUTE=y
CONFIG_NET_IPIP=m
```

O daemon do mrouted está disponível em ftp://ftp.research.att.com/dist/fenner/mrouted/. Para instalá-lo basta descompactá-lo:

```
# tar xvzf mrouted-3.9-beta3.tar.gz
```

Para funcionar de acordo no Linux deve-se baixar um patch que se encontra no repositório da Debian, http://www.debian.org/distrib/packages e aplicá-lo, conforme-segue:

```
# cd mrouted-3.9-beta3
# zcat ../mrouted_3.9-beta3-1.1.diff.gz | patch -p1
```

Após aplicar o patch, basta compilar o mrouted:

```
# make
```

Para usuário do Debian GNU/LInux, basta fazer:

```
# apt-get install mrouted
```

Sinais

O mrouted pode receber alguns sinais:

- HUP: Reinicia o mrouted. Ex:

```
# kill -HUP <pid>
```

- INT: Termina a execução corretamente. Ex:

```
# kill -INT <pid>
```

- TERM: Mesmo que INT

- USR1: Gera um arquivo em /var/tmp/mrouted.dump com as tabelas de roteamento internas. Ex:

```
# kill -USR1 <pid>
```

- USR2: Gera um arquivo em /var/tmp/mrouted.cache com as tabelas de cache internas. Ex:

```
# kill -USR2 <pid>
```

- QUIT: Gera a tabela de roteamento para stderr. Ex:

```
# kill -QUIT <pid>
```

Configurando o mrouted.

O arquivo de configuração do mrouted encontra-se geralmente em /etc/mrouted.conf e é altamente recomendado que se leia os comentários do arquivos que ja apresentam alguns exemplos:

```
# mrouted.conf,v 3.8 1995/11/29 22:40:47 fenner Rel
#
#   This is the configuration file for "mrouted", an IP multicast router.
#   mrouted looks for it in "/etc/mrouted.conf".
#
#   Command formats:
#
# name <boundname> <scoped-addr>/<mask-len>
# cache_lifetime 3600           # seconds
# pruning on
#
# phyint <local-addr> disable metric <m> threshold <t> rate_limit <b>
#                         <scoped-addr>/<mask-len>)
#                         <subnet>)
# tunnel <local-addr> <remote-addr> srcrt metric <m>
#                                     threshold <t> rate_limit <b>
#                         <scoped-addr>/<mask-len>)
#
#   NOTE: any phyint commands MUST precede any tunnel commands
#   NOTE: the mask-len is the no. of leading 1's in the mask
#   NOTE: rate_limit is in kilobits, and defaults to 500 for tunnels
#
# Example of named bounary:
#name LOCAL 239.255.0.0/16
#name EE 239.254.0.0/16         # i.e. the EE dept wants local groups
#
# Example of use of named boundary
#phyint le1 boundary EE         # le1 is our interface to comp sci,
#                               # keep them away from our local groups
#
#
# Template tunnel for mcast_install
#tunnel 128.4.0.77 128.4.0.8 metric 1 threshold 64 rate_limit 500  # <-- REPLACE
#       boundary LOCAL
#
# You might want to specify a boundary on your tunnel to the outside world,
# as above.
#
# NOTE: ONLY uncomment the following if you are running mrouted.snmp!
#sysName        "mymrouter"
#sysContact     "Me <me@me.com> +x.yyy.zzz-zzzz"
#sysVersion     "MyOS 4.1.3 and mrouted"
#sysLocation    "The MBONE"
```

Configurando um Túnel

Uma configuração muito interessante que comumente é usada é a criação de túneis, abaixo temos um exemplo de túnel entre a Rede A e a Rede B

Rede A: 10.0.0.0/24 mrouted rodando em 10.0.0.1
Rede B: 192.168.0.0/24 mrouted rodando em 192.168.0.1

Os túneis devem ser configurados desta forma:

```
# tunnel local-addr|ifname remote-addr|remote-hostname
```

Devemos configurar o host 10.0.0.1 da seguinte maneira:

```
tunnel 10.0.0.1 192.168.0.1
```

E devemos configurar o host 192.168.0.1 da seguinte maneira:

```
tunnel 192.168.0.1 10.0.0.1
```

Várias opções interessantes podem ser adicinados na configuração do tunel (man mrouted).

Ferramentas úteis.

Quando estamos utilizando o mrouted, algumas ferramentas podem ser úteis para o gerenciamento e na resolução de problemas:

- map-mbone: desenha o mapa de conexões multicast
- mrinfo: mostra informações de configuração de um roteador multicast

PIM - Protocol Independent Multicast

O protocolo PIM (Protocol Independent Multicast), é um protocolo de roteamento multicast amplamente utilizado na Internet e é o padrão para roteadores Cisco.

PIM - SM

Há uma impementação de PIM para o GNU/Linux que compreende a versão SMv2 do protocolo. Existem alguns problemas na implementação do pimd SMv2,

como a eleição de RP. Veja o mail abaixo:

Pedro,

In your case you need to configure pimd with static RP address. Unfortunately, pimd doesn't support static RP configuration. Adding that to pimd is not trivial for variety of reasons.

However, I should let you know that some time ago I did a complete reimplementation of PIM-SM (as part of the XORP project http://www.xorp.org/), and that reimplementation does support static RPs. Pimd itself has a number of problems hence I decided to discontinue working on it, and completely retire it after the XORP PIM-SM implementation is officially ready for prime time. Though, even now I have much more faith in the new implementation, hence I'd recommend to use it instead of pimd. Probably the only major drawback of the new implementation is that currently its
configuration is a bit odd (via a shell script), but this should be fixed in the future.

If you decide to try the XORP PIM-SM implementation, I'd recommend that you get the lastest version from the CVS repository instead of the lastest Release-0.3 In any case, please let me know if you have any problems with it.

Regards,
Pavlin

Instalação:

O código fonte dessa implementação está dispoível em http://netweb.usc.edu/pim/ . Há também o binário para usuários do Debian GNU/Linux, bastando:

```
# apt-get install pimd
```

Configuração:

Abaixo temos um exemplo comentado de um arquivo de configuração:

```
#   This is the configuration file for "pimd", an IP multicast router.
#   pimd looks for it in "/etc/pimd.conf".
#
#   $Id: pimd.conf,v 1.17 2001/09/10 20:31:37 pavlin Exp $
#
#   Command formats:
##########
```

```
# default_source_preference <preference>
# default_source_metric <metric>
#
# phyint <local-addr | ifname> disable  threshold <t> preference <p> metric <m> alt
net   <net-addr> masklen <masklen>
scoped <net-addr> masklen <masklen>
#
# cand_rp <local-addr> priority <number>  time <number>
# cand_bootstrap_router <local-addr> priority <number>
#
# group_prefix <group-addr> masklen <masklen>
# group_prefix <group-addr> masklen <masklen>
# .
# .
# group_prefix <group-addr> masklen <masklen>
#
#
# switch_data_threshold rate <number> interval <number>
#
# switch_register_threshold rate <number> interval <number>
##########
# By default PIM will be activated on all interfaces.  Use phyint to
# disable on interfaces where PIM should not be run.
#
# Preferences are used by assert elections to determine upstream routers.
# Currently pimd cannot reliably obtain preferences and metrics from the
# unicast routing protocols, so a default preference may be configured.
# In an assert election, the router advertising the lowest assert preference
# will be selected as the forwarder and upstream router for the LAN.
# 101 should be sufficiently high so that asserts from Cisco or GateD
# routers are prefered over poor-little pimd.
#
# It is reccommended that preferences be set such that metrics are never
# consulted.  However, default metrics may also be set and will default to
# 1024.
#
#
```

Firewall

Um firewall, ou filtro de pacotes, é um recurso utilizado para proteger uma máquina ou uma rede através do controle e filtragem dos pacotes de dados que entram ou que saem. No Linux esse recurso é implementado diretamente no kernel e recebe, nas versões de kernel >2.4, o nome de netfilter. Além da simples filtragem de pacotes, o netfilter é capaz de manipular campos dos cabeçalhos de pacotes, fazer a tradução de endereços de rede (NAT), "marcar" pacotes e fazer o acompanhamento de conexões (connection track). Este último recurso faz com o que o netfilter seja um firewall do tipo "statefull", ou seja, capaz de reconhecer o "estado" de uma conexão.

Introdução ao Netfilter

Para entendermos como funciona o netfilter, é importante compreendermos alguns coneceitos utilizados por ele.

Tabelas e Chains

Tabela é o local utilizado para armazenar chains que por sua vez contém regras. O netfilter possui três tabelas: filter, mangle e nat. Cada tabela é utilizada para propósitos diferentes: a tabela filter é utilizada para aceitar ou rejeitar pacotes; a tabela mangle é usada para manipular alguns campos do cabeçalho IP, como por exemplo o campo TOS e o campo TTL. A tabela mangle também pode ser utilizada para se inserir "marcas" no pacote; a tabela nat é utilizada para fazer Traduções de Endereços de Rede (NAT).

Uma chain é simplesmente um conjunto de regras, no entanto existem dois tipos de chains: as chains do kernel e as chains criadas pelo usuário. As chains do kernel - PREROUTING, INPUT, FORWARD, OUPUT e POSTROUTING - estão ligadas a pontos especiais no caminho que os pacotes precorrem ao entrar e sair da máquina. Enquanto que as chains do usuário não estão ligadas a ponto algum, logo, é necessário que uma chain do kernel tenha como alvo uma chain de usuário para que os pacotes percorram essa chain.

A chain PREROUTING é percorrida pelos pacotes que chegam à interface de rede, mas antes da consulta às tabelas de roteamento. Após esse ponto é feito a decisão de roteamento e o pacote poderá ser destinado à máquina local, e nesse caso percorrerá a chain INPUT, ou será roteado para outra máquina e percorrerá então a chain FORWARD. O pacotes antes de saírem pela interface de rede percorrem a chain OUTPUT, caso tenham sido originados na máquina local,

e a chain POSTROUTING, que é chain consultada após a tomada de decisão de roteamento, quando os pacotes estão prestes a sair da máquina.

É importante ressaltar que a filtragem de pacotes é feita em basicamente 3 lugares: nas chains INPUT e OUTPUT para pacotes com origem e destino na máquina local, e na chain FORWARD para pacotes que atravessam o roteador.

Regras

Uma regra do netfilter descreve o que fazer, onde e com que tipo ou padrão de pacote. Ao criarmos uma regra, especificamos em que tabela e chain estamos inserindo essa regra, definimos um padrão que irá casar com os pacotes, utilizando para isso os campos do cabeçalho do pacote e finalmente especificamos um "alvo" para a regra, que poderá ser uma ação como aceitar ou rejeitar o pacote, ou poderá ser um "jump" para uma outra chain.

Quando um pacote "entra" numa chain, cada regra é avaliada, de maneira sequencial, até que o pacote case com uma regra, ou o pacote atinja o final da chain. Quando um pacote atinge o final de uma chain sem que tenha casado com alguma regra, é aplicada então a política padrão da chain. Por padrão a política padrão da chain é "ACCEPT" (aceita), mas isso pode ser alterado.

Estados

Como foi dito anteriormente, o netfilter é capaz de reconhecer o estado de uma conexão. Isso além de facilitar a configuração de regras, permite um maior controle sobre o que passa pelo firewall. O netfilter pode atribuir quatro estados para um pacote: NEW, ESTABLISHED, RELATED e INVALID.
Pacotes são classificados como NEW quando são os primeiros pacotes de um fluxo. Quando chega ao firewall a resposta à esse pacote inicial, os pacotes desse fluxo passam a ser identificados como ESTABLISHED. Pacotes que de alguma maneira se relacionam à uma conexão já estabelecida são classificados como RELATED. Ao se fazer uma conexão FTP, por exemplo, é aberto duas conexões: uma onde trafegam dados e outra informações/controle. Essa segunda conexão poderia ser vista pelo netfilter como sendo "RELATED". E por fim pacotes que por alguma razão não foram identificados podem receber o estado de INVALID.

É importante ressaltar que mesmo protocolos não orientados a conexão, como UDP e ICMP podem ser tratados como orientados a conexão pelo firewall. Vejamos o caso do ping, por exemplo. Quando o firewall recebe um pacote ICMP echo request com um determinado ip de origem/destino, esse pacote é marcado como NEW. Ao receber o pacote ICMP echo reply correspondente, os pacotes passam a ser classificados como ESTABLISHED.

A utilização dos estados simplifica bastante a configuração do firewall. Imagine por exemplo que quiséssemos fazer um firewall que permitisse qualquer conexão iniciada pela intranet, mas não permitisse que nenhuma conexão iniciada externamente chegasse a intranet. Bastaria então que permitíssemos pacotes NEW, ESTABLISHED e RELATED vindos da intranet, mas apenas pacotes ESTABLISHED e RELATED vindos da internet.

Configurando um Firewall

Entendido os conceitos expostos, e tendo bem claro o que bloquear e o que não bloquear, fica fácil construir um firewall. A questão "o que bloquear" fica a cargo do administrador de rede, e não trataremos dela aqui.

O comando utilizado para manipular as opções do netfilter chama-se "iptables".

Criando e Removendo Regras

A sintaxe geral para se criar ou remover uma regra é a seguinte:

```
# iptables -[AID] CHAIN [N] [-t TABLE] MATCH -j TARGET
```

A opção -A é usada para se fazer o "append" de uma regra à uma chain, isto é, a regra será inserida como a última da chain. A opção -I é usada para se inserir uma regra na posição N, e a opção -D é usada para se deletar uma regra. Para deletar podemos simplesmente indicar o número da regra, ou podemos usar as mesmas opções que foram usadas para se criar a regra.

A opção -t admite as 3 tabelas descritas anteriormente. Caso ela não seja definida o iptables considera que a tabela seja a "filter".

As opções que definem o padrão de casamento de uma regra são muitas, como podemos verificar na tabela (incompleta) abaixo:

opção (short)	opção (long)	parâmetro	significado
-s	--src	X.X.X.X/Y	casa com endereço de origem do pacote
-d	--dst	X.X.X.X/Y	casa com o endereço de destino do pacote
-i	--in-interface	interface	casa com a interface pela qual o pacote chegou

-o	--out-interface	interface	casa com a interace pela qual o pacote vai sair
-p	--protocol	tcp, udp, icmp	casa com o protocolo do pacote
--sport	--source-port	porta[:porta]	especifica a porta ou intervalo de portas de origem
--dport	--destination-port	porta[:porta]	especifica a porta ou intervalo de portas de destino
	--icmp-type	tipo/código	especifica o tipo de pacote icmp (iptables -p icmp -h para info
	--match	habilita o modo de opções extendido	

Além das opções acima, ao se especificar a opção --p PROTOCOL ou --match, poderemos usar mais opções, entre elas:

opção	parâmetro	significado
--limit	X/time	limita a média de matchs. Time pode ser / second, /minute, /hour ou /day
--limit-burst	X	Número máximo inicial de matchs. Esse valor é "recarregado" em 1 a cada intervalo de tempo em que o limite não foi alcançado, até o valor especificado. Utilize essa opção para permitir rajadas.
--mask	X	casa com um pacote "marcado" com o valor X.
--state	NEW,ESTABLISHED, RELATED, INVALID	casa com o estado de uma conexão.
--tcp-flags	MASK FLAGS	Examina as flags "MASK" e casa com FLAGS. Exemplo: SYN,ACK ACK = examina SYN e ACK e casa com pacotes com SYN desligado e ACK ligado.
--tos	name	casa com o campo Type of Service. User iptables -m tos -h para ver os nomes/valores
--ttl	valor	casa com o valor do campo ttl do pacote

Uma opção pode ser "negada" se a precedermos com um "!". Assim, por exemplo, ! -s 192.168.0.1/32 casará com qualquer origem. menos com 192.168.0.1/32.

E finalmente devemos especificar o alvo de uma regra. Como cada tabela possui alvos diferentes, com opções diferentes, veremos os alvos agrupados por tabelas:

Tabela filter:

Alvo	opções	significado
ACCEPT		Aceita um pacote
REJECT		Rejeita um pacote
REJECT	--reject-with type	Rejeita um pacote e envia um pacote ICMP type para a origem
DROP		Descarta silencionsamente o pacote
LOG		Loga os headers do pacote
LOG	--log-level X	Loga utilizando o nível X (veja o arquivo syslog.conf)
LOG	--log-prefix	permite acrescentar um prefixo de 29 letras. É útil para identificar o tipo pacote/regra

Tabela mangle:

Alvo	opções	significado
MARK	--set-mark X	marca um pacote com o valor X
TCPMSS	--set-mss X	ajusta o valor MSS para o valor X
TOS	--set-tos tos	ajusta o valor do tos. Use iptables -j tos -h para ver os valores possíveis

Tabela nat:

Alvo	opções	significado
DNAT	--to-destination ipadd [-ipaddr][:port-port]	modifica o destino do pacote - Destination Nat
MASQUERADE		o masquerade é um tipo de SNAT, mas é utilizado quando ip do roteador é dinâmico

REDIRECT	[--to-port]	altera o destino do pacote para a máquina local, opcionalmente altera também a porta de destino
SNAT	--to-source ipaddr [-ipaddr][:port-port]	altera a origem dos pacotes - Source NAT. opcionalmente altera também as porta de origem

Além dos alvos específicos uma regra pode ter como alvo outra chain. Suponha que tenhamos criado uma chain somente para tratar de pacotes ICMP e tenhamos chamado essa regra de ICMP_FILTER. Poderíamos ter uma regra então na chain FORWARD do tipo:

```
# iptables -A FORWARD -p icmp -j ICMP_FILTER
```

Pacotes que casassem com a regra acima seriam jogados na chain ICMP_FILTER. Para fazer com que um pacote volte para chain de onde foi encaminhado, utilizamos o alvo "RETURN". Assim, por exemplo, se quiséssemos que pacotes com origem em 192.168.1.1 retornassem a chain FORWARD poderíamos criar a regra:

```
# iptables -A ICMP_FILTER -s 192.168.1.1 -j RETURN
```

Alguns alvos não são "terminais" ou seja, o netfilter continua percorrendo a chain mesmo que uma regra tenha sido casada. Os alvos LOG, MARK, TCMPSS e TOS são desse tipo.

EXEMPLOS:

Dropa tudo que chegue ao roteador com destino ao host 192.168.0.3:

```
# iptables -A FORWARD -d 192.168.0.3/32 -j DROP
```

Bloqueia o acesso a porta 23 da máquina local de acessos vindos pela interface eth0:

```
# iptables -A INPUT -i eth0 -p tcp --dport 23 -j REJECT
```

Loga tentativas de acesso a porta 161 UDP vindas de fora da rede 192.168.0.0/24, limitando o número de match para não encher rapidamente nosso log:

```
# iptables -A FORWARD -p udp --dport 161 ! -s 192.168.0.0/24 -j LOG --limit 1/secon
d
```

Faz NAT para a rede interna:

```
# iptables -A POSTROUTING -t nat -s 192.168.0.0/24 -o eth1 -j SNAT --to-source 200.
X.X.36
```

Supondo que tenhamos um proxy rodando na máquina, e queremos fazer um proxy "transparente":

```
# iptables -A PREROUTING -p tcp --dport 80 -s 192.168.0.0/24 -j REDIRECT --to-ports
3128
```

Supondo que tenhamos dois servidores dentro da nossa intranet (ip inválido) e queremos balancear as conexões entre os dois:

```
# iptables -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT --to-source 192.168.1.5,
192.168.1.6
```

Marcando um pacote. Essa marca poderá ser utilizada depois por um outro filtro.

```
# iptables -A PREROUTING -s 192.168.1.5 -j MARK --set-mark 5
```

Casando pacotes com determinada marca:

```
 # iptables -A FORWARD --match --mark 5 -j ACCEPT
```

Listando Regras

Para se listar o contéudo de uma chain basta utilizar o comando iptables -L [CHAIN] [table]. Lembrando que quando não especificado nenhuma tabela, o padrão é mostar a tabela filter. Exemplos:

Mostra todas as chains da tabela filter:

```
# iptables -L
```

Mostrando todas as chains da tabela mangle:

```
# iptables -L -t mangle
```

Mostrando a chain PREROUTING da tabela nat:

```
# iptables -L PREROUTING -t nat
```

Uma opção muito útil ao se listar chains é o "-v". Utilizando essa opção é mostrado quantos pacotes e quantos bytes percorreram cada chain/regra.

Criando e Removendo Chains

Chains de usuárias são muito utéis quando a quantidade de regras do firewall é grande. Imagine que tenhamos 30 regras para pacotes TCP, mais 20 para pacotes UDP e 10 regras para pacotes ICMP. Um pacote que não case com nenhuma dessas regras percorreria 60 regras! Para evitar isso poderíamos criar chains específicas para cada protolo. Para se criar um chain de usuário basta utilizar o comando:

```
# iptables -t tabela -N chain_do_usuario
```

Para se remover uma chain usamos:

```
# iptables -t tabela -X chain
```

Ou para remover todas as chains criadas pelo usuário:

```
# iptables -X
```

Limpando Chains

Já vimos como remover regras individualmente de uma chain usando a opção "-D", mas e se quiséssemos remover todas as regras de uma chain? Bastaria utilizar a comando:

```
# iptables - F chain -t table
```

Controle de Tráfego

Controle de tráfego é o nome dado ao conjunto de sistemas de enfileiramento (queuing) e mecanismos pelos quais os pacotes são recebidos e transmitidos em um roteador. Isso inclui que pacotes aceitar a que velocidade e em qual interface e que pacotes devem ser priorizados. O termo QoS (Quality of Service) é frequentemente utilizado como sinônimo de Controle de Tráfego.

O controle de tráfego no Linux se encontra em um estado maduro e bastante avançado em termos de recursos.

Conceitos

Para entendermos melhor o controle de tráfego, vamos nos familiarizar com alguns conceitos utilizados nele:

Filas (queues)

Uma fila é um local (ou buffer) contendo um número finito de itens aguardando por uma ação ou serviço. Em redes, uma fila é local onde os pacotes aguardam antes de serem enviados pelo hardware.

Shapping

Shapping é o ato de "atrasar" o envio de pacotes de forma que o tráfego fique dentro de um determinado limite de velocidade. Como efeito colateral um shapper pode deixar um determinado tráfego mais regular, ao remover ou "achatar" as rajadas de pacotes.

Scheduling

Scheduling ou escalonamento é o ato de ordenar ou re-ordenar os pacotes para serem enviados. O método mais comum de escalonamento é o FIFO (First-in First-out, "o primeiro a chegar é o primeiro a sair"). Outros métodos de escalonamento incluem o SFQ, que procura dar a cada "flow" uma chance de transmitir seus pacotes, ou o RED, que procura descartar pacotes de forma randômica ao se atingir uma determinada condição de forma a evitar que o

backbone fique saturado.

Classifying

Classificação (classifying) é o mecanismo pelo qual os pacotes são separados de forma a terem tratamentos diferentes, possivelmente colocando-os em diferentes filas de saída. Uma pacote ou flow pode ser classificado de várias maneiras diferentes ao atravessar um roteador ou rede.

Policing

Policing é o mecanismo pelo qual o tráfego pode ser limitado. Assim é possível garantir de que um determinado tipo de tráfego não consuma mais banda do que foi destinado a ele.

Componentes do Controle de Tráfego do Linux

O Linux possui os seguintes componentes: qdisc, class e filters. Antes de vermos com mais detalhes cada um deles, vamos fazer uma rápida relação desses elementos com os conceitos expostos:

Conceito	Componente
Shapping	As "class" oferecem a capacidade de "shapping"
Scheduling	um qdisc é um escalonador. Ele pode ser simples como um FIFO (fila) ou pode conter classes dentro dele
Classifying	A classificação é feita dentro de um "filter" através de um "classificador"
Policing	O policing é implementado no Linux como uma parte de um filter

Qdisc

Os qdiscs (abreviação de Queueing Disciplines) são as filas de saída dos pacotes. Um qdisc exerce a função de escalonador de pacotes também.

O Linux possui dois tipos de qdisc: classless e classfull qdiscs. Os qdics classless não podem conter classes definidas pelo usuário, embora alguns deles possuam mais de uma fila de saída. Já os qdiscs classfull podem conter subclasses definidas pela usuário, permitindo assim a separação e a atribuição de

quantidades diferentes de banda para cada tipo de fluxo.

Cada interface de rede possui um qdisc "root". O padrão do Linux é atribuir um qdisc do tipo "pfifo_fast" para o qdisc root, mas obviamente, isso pode ser mudado.

Como não pretendemos aqui fazer um "tratado" sobre QoS no Linux veremos apenas alguns qdiscs, entre eles três classless, pfifo_fast, sfq e tbf e um classfull, o htb. Embora um dos qdiscs classfull mais populares seja o cbq (talvez por ser um dos mais antigos), não trataremos dele, pois é mais complexo e pode ser substituído com vantagens pelo htb.

Classless Qdiscs

PFIFO_FAST

O pfifo_fast é o qdisc padrão do Linux. Ele é baseado no FIFO, mas possui internamente três filas, onde determinado tipo de tráfego pode ser priorizado. Os pacotes são colocados nestas filas de acordo com o seu TOS, assim pacotes pertencentes a tráfegos do tipo interativo recebem maior prioridade e são sempre servidos primeiro. Pacotes na fila 0, são servidos primeiro, e somente quando essa fila se encontra vazia, são servidos os pacotes da fila 1. De maneira análoga são servidos os pacotes da fila 2.

SFQ

O qdisc SFQ (Stochastic Fair Queuing) é um qdisc que procura distribuir de maneira igual a oportunidade para cada flow ser servido. O SFQ mantém internamente várias filas FIFO e utiliza uma função HASH para distribuir os pacotes entre essas filas. As filas são então servidas usando uma função "round-robin". Para evitar que a função hash escolhida acabe privilegiando ou prejudicando um determinado tipo de fluxo, o SFQ possui o parâmetro "perturb" que indica de quanto em quantos segundos a função HASH deve ser recalculada.

TBF

O TBF (Token Bucket Filter) utiliza o modelo de balde de fichas para "shapear" o tráfego. Normalmente o TBF é utilizado quando desejamos simplesmente limitar a velocidade de uma determinada interface. Os principais parâmetros do TBF são:

Parâmetro	significado

limit	Quantidade de bytes que podem aguardar na fila
burst	tamanho do "balde" em bytes. Esse parâmetro pode ser utilizado para permitir rajadas
rate	velocidade com que os "tokens" chegam ao balde.

Classful Qdiscs

HTB

O HTB (Hierarchical Token Bucket) utiliza o conceito de balde de fichas combinado com um sistema de classes e filtros, que permite a configuração de um controle do tráfego bastante preciso e complexo.

Como o HTB é um qdisc com suporte a classes, ele pode ser utilizado como um qdisc (um escalonador/shaper) ou como uma classe. Quando utilizado como classe o htb possui apenas um parâmetro (opcional) "default" que define qual subclasse deverá ser utilizada caso um pacote não seja classificado por nenhum filtro.

Como um escalanador o HTB suporta, dentre outros, os seguintes parâmetros:

Parâmetro	Significado
rate	velocidade de transmissão dos pacotes
burst	tamanho máximo de bytes que pode ser acumulado para rajadas
ceil	velocidade total da classe superior. Esse parâmetro é utilizado para permitir que uma classe tome "emprestado" banda disponível de outra classe

Class

As classes só existem dentro de qdiscs classful e são uma forma de se dividir o tráfego para um tratamento diferenciado. Uma classe não manipula um pacote diretamente, isso é feito pelo qdisc associado à ela. Além da classe "root", existem dois tipos de classes. As classes "leafs", que são classes que não possuem "filhos" e classes "inner" que são classes que possuem "filhos".

Filter

Os filtros exercem a função de separar o tráfego em classes dentro de um qdisc classful. Isso é feito com o auxílio de um "classificador". Além disso um filtro pode exercer a função de "policer" tomando uma ação sempre de acordo com os limites estabelecidos para o fluxo que ele está classificando. Normalmente um filtro tem como política descatar pacotes que excedam o limite estabelecido para o qdisc no qual ele está embutido, no entanto é possível configurar outras ações, como por exemplo, re-classificar o pacote utilizando outro filtro.

Classificadores

Dentro de um filtro, um classificador é utilizado para identificar certos padrões e/ou características dos pacotes e fluxos permitindo assim a separação em classes. Veremos a seguir apenas dois classificadores do Linux:

U32

O U32 é o principal classificador do Linux, pois ele permite a identificação de qualquer parte de um pacote IP (normalmente campos do cabeçalho). O U32 é bastante utilizado quando precisamos identificar IPs de origem e/ou destino, portas de origem/destino ou mesmo protocolo (icmp, tcp, udp, etc). Veremos a seguir alguns dos principais parâmetros do U32:

Parâmetro	Significado
ip src IP/MASK	Casa com pacotes com origem em IP/MASK
ip dst src IP/MASK	Casa com pacotes com destino IP/MASK
ip sport NN 0xffff	Casa com pacotes com porta de origem NN. O parametro 0xffff é uma máscara de 32 bits
ip dport NN 0xffff	Casa com pacotes com porta de destino NN. O parametro 0xffff é uma máscara de 32 bits
ip protocol NN 0xff	Casa com pacotes do tipo procolo NN. Olhe em /etc/protocols
ip tols 0xNN 0xMM	Casa com os bits NN do campo TOS. Utilize a mascara MM para escolher quais bits se deseja comparar
flowid X:Y	Os pacotes que casam com essa regra devem ser colocados na class X:Y

O U32 é bastante poderoso e pode procurar por padrões dentro de qualquer

lugar do cabeçalho ou mesmo do pacote inteiro. Para isso ele usa os seletores u32, u16, e u8 que casam com padrões de 32, 16 e 8 bits respectivamente. Esses seletores aceitam como parâmetros um padrão de bytes, uma máscara, um deslocamento em relação ao início do cabeçalho ou um deslocamento em relação ao próximo cabeçalho (cabeçalho da camada de transporte por exemplo). Mas vamos deixar essas opções mais avançadas para uma outra oportunidade.

FW

O classificador FW procura por "marcas" feitas por regras do netfilter. Sua utilização é bastante simples pois ele requer apenas dois parâmetros: handle N, informando qual "marca" ele deve procurar e flowid X:Y indicando em qual classe ele deverá colocar os pacotes encontrados.

Configurando o Controle de Tráfego

A configuração do controle de tráfego no Linux é toda feita através de um único comando: o "tc". O comando tc faz parte do pacote iproute2 - o mesmo do comando ip. O funcionamento do comando tc é similar ao do comando ip, e ele tem a forma geral:

```
# tc [opções] OBJETO PARÂMETROS
```

Os objetos do comando tc são: qdisc, class e filter. Veremos a seguir como manipular esses objetos.

NOTA: o tc utiliza as seguintes regras para a especificação de velocidades/banda:

mbps = 1024 kbps = 1024 * 1024 bps => byte/s
mbit = 1024 kbit => kilobit/s.
mb = 1024 kb = 1024 * 1024 b => byte
mbit = 1024 kbit => kilobit

tc qdisc

Utilizamos o objeto qdisc para adicionar, remover e listar qdiscs. Veremos como executar cada uma dessas operções:

Adicionando e removendo qdiscs

Para adicionar um qdisc precisamos indicar em que interface estamos adicionando, qual é "handle" (identificador) do qdisc, quem é o "pai" do qdisc, e finalmente o tipo de qdisc e suas opções. O pai de um qdisc pode ser "root" caso

estejamos adicionando um qdisc root ou "parent X:Y" onde X:Y é a classe a qual esse qdisc está sendo adicionado. Exemplos:

Adicionando um qdisc root:

```
# tc qdisc add dev eth0  root handle 1: tbf rate 5mbit burst 100kbit limit 5k
```

Adicionando um qdisc a uma classe:

```
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
```

Adicionando um qdisc classful e definindo uma classe "padrão":

```
# tc qdisc add dev eth0 root handle 1: htb default 10
```

Para remover um qdisc basta utilizar a opção "del" em vez de add.

Listando qdiscs

Para lista um qdisc basta utilizar a opção show, não esquecendo de informar o dispositivo de rede, exemplo:

```
# tc qdisc show dev eth0
```

tc class

Como vimos anteriormente as classes funcionam como uma espécie de "subdivisão" de um qdisc classful, permitindo a separação do tráfego em outras classes/qdiscs. No caso específico do qdisc classful que veremos, o htb, é nas classes "leafs" que é feita o shaping.

Adicionando e removendo classes

Para criar uma classe precisamos informar a interface, o qdisc/classe pai dessa classe, o identificador de classe, o tipo de classe e suas opções. Exemplos:

Adicionando uma classe a um qdisc root:

```
# tc class add dev eth0 root classid 1:0 htb rate 2mbit
```

Adicionando uma classe como filha de outra classe:

```
# tc class add dev eth0 parent 1:0 classid 1:100 htb rate 100kbit
```

Vale lembrar que uma classe "leaf" pode ter um qdisc anexada a ela, mas um classe "inner" não.

Listando classes

Para listar as classes, basta usar o comando com a opção show. Adicionalmente podemos utilizar as opções -s para obtermos a quantidade de bytes e pacotes enviados por cada classe. Exemplo:

```
# tc -s class show dev eth0
```

tc filter

Os filtros devem ser "anexados" às classes para efetuarem a separação do tráfego. Um ponto importante a lembrar é a de que as classes possuem os filtros e não o contrário, ou seja, somente quando os pacotes "entram" em uma classe os filtros associadas à ela serão executados. Quando temos uma configuração com classes é importante definir uma classe "raiz" (uma classe no qdisc raiz com o "minor" 0, ex: 1:0), pois a partir dela poderemos criar filtros que apontem para outras classes.

Adicionando e removendo filtros

Para adicionarmos um filtro utilizamos o comando "tc filter add" e informamos à interface, a classe na qual queremos inserir o filtro, o protocolo, o classificador a ser utilizado e suas opções. Exemplos:

Insere um filtro na classe "raiz", casando com pacotes que venham da rede 10.0.0.0/24:

```
# tc filter add dev eth0 parent 1:0 protocol ip u32 match ip src 10.0.0.0/24 flowid
1:10
```

Insere um filtro na classe 1:10, casando com pacotes que tenham como destino a porta 80 da máquina 192.168.0.1:

```
# tc filter add dev eth0 parent 1:10 protocol ip u32 match ip dst 192.168.0.1/32 ma
tch ip dport 80 0xffff flowid 1:100
```

Os filtros aceitam também o parametro "prio N" que define a prioridade com que os filtros de uma mesma classe devem ser utilizados. Exemplo:

Insere um filtro na classe 1:20 com prioriade 2. Esse filtro casa com pacotes que foram marcados pelo netfilter com a tag "6"

```
# tc filter add dev eth0 parent 1:20 protocol ip prio 2 handle 6 fw flowid 1:200
```

Para remover um filtro utilize o mesmo comando usado para criá-lo mas com a

opção "del" em vez de "add".

Listando filtros

De maneira análoga aos outros objetos, para listar os filtros utilize o comando "tc filter show", como no exemplo:

```
# tc filter show dev eth0
```

Mais Exemplos

Um recurso bastante interessante do controle de tráfego no Linux, e em particular do qdisc htb é a possibilidade de se criar classes "irmãs" que podem tomar "emprestado" banda de outra classe, caso essa banda esteja disponível. Vamos criar um exemplo prático para ver como isso funciona.

Vamos supor que temos um link de 5mbits e queremos destinar 2mbits para o tráfego web, mais 2mbits para ftp e 1mbit para o "resto" , no entanto queremos que caso haja banda disponível, o tráfego web possa chegar a 4mbits e o tráfego ftp a 3 mbits. Uma possível solução para esse cenário:

```
# tc qdisc add dev eth0 root handle 1: htb default 30
# tc class add dev eth0 root classid 1:0 htb rate 5 mbits
# tc class add dev eth0 parent 1:0 classid 1:10 htb rate 2mbit ceil 4mbit
# tc class add dev eth0 parent 1:0 classid 1:20 htb rate 2mbit ceil 3mbit
# tc class add dev eth0 parent 1:0 classid 1:30 htb rate 1mbit
```

Isso define as classes e suas velocidades. O ponto principal aqui é o parametro "ceil" que define um valor máximo de banda que pode ser utilizado caso haja banda disponível para se pegar "emprestado". Basta agora criar dois filtros para separar o tráfego web e ftp. Pacotes que não pertençam a essas classes caem no padrão definido na primeira regra.

```
# tc filter add dev eth0 parent 1:0 protocol ip u32 match ip dport 80 0xffff flowid
1:10
# tc filter add dev eth0 parent 1:0 protocol ip u32 match ip dport 20 0xffff flowid
1:20
# tc filter add dev eth0 parent 1:0 protocol ip u32 match ip dport 21 0xffff flowid
1:20
```

Obs: utilizamos as portas 20 e 21 para caracterizar o tráfego ftp.

Caso não haja tráfego algum para a classe 30, esses 1mbits podem ser divididos pelas classes 1:10 e 1:20.

SNMP

O SNMP - Simple Network Management Protocol - é um protocolo que permite a consulta e o gerenciamento de dispositivos de rede. Uma das utilizações mais comuns do SNMP é a consulta de estatisticas de trafego que permitam a construção de gráficos históricos de utilização de banda (o famoso MRTG).

Visto a sua utilidade, é de se esperar que um roteador possua suporte a SNMP. No Linux o suporte ao SNMP é feito através de ferramentas e daemons especiais, contidos no pacote Net-snmp (antigo ucd-snmp). Esse pacote contém ferramentas para consultar agentes SNMP (ex. snmpget, snmpwalk), um agente SNMP (snmpd) e um daemon para receber "traps" geradas por agentes. Vamos nos concentrar aqui no agente SNMP, e vamos partir do princípio que o leitor já tenha alguma familiaridade com o SNMP (e que possivelmente já o utilize em roteadores dedicados).

Instalação

O código fonte do net-snmp está disponível em http://net-snmp.sourceforge.net/.

Para usuários do SO Debian GNU/Linux basta fazer:

```
# apt-get install snmpd
```

O pacote snmpd contém o daemon, ou seja, o agente NET SNMP que permitirá as facilidades do protocolo SNMP.

```
# apt-get install snmp
```

O pacote snmp contém as aplicações do que permitem fazer as queries e outras funcionalidades.

O Daemon snmpd

O daemon snmpd é o responsável por receber requisições snmp e processá-las. No Linux ele é muito mais utilizado para consulta do que para efetuar alguma alteração na configuração. Na verdade nem é muito recomendado utilizá-lo de outra forma já que o protocolo SNMP foi feito para ser simples e não seguro!

Uma vez instalado basta fazer apenas algumas pequenas alterações na configuração para se ter um agente snmp funcionando. A primeira alteração diz respeito ao daemon snmpdtrap que é acompanha o snmpd. Como sua presença é

mais necessária em uma estação de gerenciamento e não no roteador, vamos desabilitá-lo, bastando para isso alterar o arquivo /etc/default/snmpd e alterar a opção "TRAPDRUN=yes" para "TRAPDRUN=no".

Configurando o snmpd

A primeira coisa a se configurar no snmpd são as permissões de acesso ao daemon. O snmpd possui um esquema de controle de acesso um pouco complicado, mas que permite bastante flexibilidade, permitindo definir quem acessa o quê e de onde! Vamos editar o arquivo de configuração /etc/snmp/snmpd.conf para permitir que as estações de gerenciamento possam acessar o daemon snmpd.

O protocolo snmp utiliza o conceito de "communities", ou comunidades, que funciona como uma espécie de senha. A versão 3 do protocolo snmp possui um esqueme de autenticação mais forte, mas as versões 1 e 2c utilizam apenas as communities. Como as versões mais utilizadas são a 1 e 2c vamos concentrar nossas atenções nela. No daemon snmpd as linhas "com2sec" são utilizadas para relacionar uma communty do snmp à um "security name" do daemon. Essas linhas tem o formato:

```
com2sec "security name"  origem_da_requisição      community
```

A origem da requisição pode ser informado na forma ip/prefixo ou pode-se utilizar a palavra "default" para indicar qualquer origem. Para fins didáticos vamos configurar um controle de acesso menos restritivo e permitir que qualquer um faça acessos de leitura apenas no daemon. Para isso vamos inserir a seguinte linha no arquivo:

```
com2sec readonly  default      public
```

Em seguida o snmpd espera que um grupos de acessos sejam associados à versão do protocolo snmp e a um "security name", desse modo podemos definir acessos a informações diferentes dependendo da versão do protocolo utilizado. As linhas que definem isso possuem a forma:

```
group grupo  versão_do_protocolo        "security_name"
```

Para prosseguir o nosso exemplo não precisaremos alterar nada nessa sessão, pois o "security_name" que defininos já está associado ao grupo "readonly". Em seguida são definidas no arquivo de configuração "views" ou seja partes da árvore de OIDs que poderão ser visualizadas. Mais uma vez não alteraremos as definições padrão.

Por fim os grupos de acessos recebem permissão de ver/alterar objetos dessas

views. Mais uma vez o padrão é o suficiente, ou seja, o grupo "MyROGroup" recebe a permissão de ver qualquer objeto.

Para finalizar a configuração básica, altere as linhas:

```
syslocation Unknown (configure /etc/snmp/snmp.local.conf)
syscontact Root  (configure /etc/snmp/snmp.local.conf)
```

De acordo com a localização e o seu contato técnico, por exemplo:

```
syslocation minhalocalização
syscontact  administrador
```

Pronto! Re-inicie o daemon snmpd e a partir da sua estação de gerenciamento teste-o com o comando:

```
# snmpwalk -v 2c -c public
```

OBS: o comando snmpwalk faz parte do pacote "snmp" no Debian.

Extendendo o snmpd

Por padrão o daemon snmpd é capaz de fornecer uma grande quantidade de informações sobre o Sistema Operacional, como processos em execução, sistemas de arquivos montados e suas capacidades, além é claro, de informações sobre os dispositivos de rede, como por exemplo quantidade de octetos que passaram por cada interface, erros, etc. No entanto, o administrador pode precisar de informações que por padrão o daemon não provê, como por exemplo informações do firewall. Para suprir esse tipo de necessidade o daemon provê suporte para que scripts sejam associados a certos MiBs/OIDs. Para tanto basta acrescentar ao arquivo de configuração linhas do tipo:

```
exec OID nome script
```

Por padrão o snmpd sugere que sejam utilizadas o ramo .1.3.6.1.4.1.2021. Se quiséssemos obter mais informações de nosso firewall poderíamos utilizar:

```
exec .1.3.6.1.4.1.2021.50 iptables /sbin/iptables -L -v
```

E para consultar:

```
# snmpwalk -v 2c -c public < ip > .1.3.6.1.4.1.2021.50
 UCD-SNMP-MIB::ucdavis.50.1.1 = INTEGER: 1
 UCD-SNMP-MIB::ucdavis.50.2.1 = STRING: "iptables"
 UCD-SNMP-MIB::ucdavis.50.3.1 = STRING: "/sbin/iptables -L -v "
 UCD-SNMP-MIB::ucdavis.50.100.1 = INTEGER: 0
```

```
 UCD-SNMP-MIB::ucdavis.50.101.1 = STRING: "Chain INPUT (policy ACCEPT 29261 packets
, 3262K bytes)"
 UCD-SNMP-MIB::ucdavis.50.101.2 = STRING: " pkts bytes target     prot opt in     o
ut     source destination "
 UCD-SNMP-MIB::ucdavis.50.101.5 = STRING: "Chain FORWARD (policy ACCEPT 0 packets,
0 bytes)"
 UCD-SNMP-MIB::ucdavis.50.101.6 = STRING: " pkts bytes target     prot opt in     o
ut     source destination"
 UCD-SNMP-MIB::ucdavis.50.101.8 = STRING: "Chain OUTPUT (policy ACCEPT 20224 packet
s, 2292K bytes)"
 UCD-SNMP-MIB::ucdavis.50.101.9 = STRING: " pkts bytes target     prot opt in     o
ut     source destination"
 UCD-SNMP-MIB::ucdavis.50.102.1 = INTEGER: 0
```

IPv6

Até aqui vimos como configurar o Linux para rotear pacotes IPv4 apenas. Veremos a seguir como transformá-lo em um roteador IPv6 também.

Ativando o Suporte a IPv6

Caso o kernel do sistema tenha sido compilado utilizando as opções sugeridas no capítulo sobre kernel, para habilitar o suporte a IPv6 bastará carregar o módulo ipv6 com o comando:

```
# insmod ipv6
```

Ao fazer isso será automaticamente adicionado um endereço IPv6 link-local (prefixo fe80::) a cada interface de rede. Verifique se isso ocorreu com o comando:

```
# ip addr show
```

Configurando Endereços IPv6

Endereços IPv6 poderão ser adicionados às interfaces de rede de dois modo: automaticamente, caso haja na rede da interface em questão um roteador fazendo anúncios RA, ou manualmente, de maneira bastante similar a configuração de endereços IPv4. Para adicionar um endereço IPv6 em uma interface utilize o comando "ip addr", exempo:

```
# ip addr add 4000::1/64 dev eth0
```

Para automatizar o esse procedimento é possível utilizar a opção "up" do arquivo de configuração de interfaces, como visto no capítulo sobre comandos básicos.

Configurando Rotas IPv6

Assim como endereços, as rotas IPv6 poderão ser adicionadas automaticamente através de mensagens RA na sua rede, ou manualmente com o comando "ip route". Exemplo:

```
# ip route add 4000:2::0/64 via 4000::2
```

Praticamente as mesmas opções do comando "ip route" utilizadas para IPv4

servem também para o IPv6.

Ativando o Roteamento IPv6

A configuração padrão do Linux é não rotear pacotes IPv6. Para ativar o roteamento IPv6 basta executar o comando:

```
# echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
```

ou:

```
# sysctl -w net.ipv6.conf.all.forwarding=1
```

Tunel IPv6-sobre-IPv4 (sit)

Para facilitar a transição de IPv4 para IPV6 o IEETF criou alguns mecanismos que permitessem que ilhas IPv6 se comuniquem mesmo que a conexão entre elas seja feito em IPv4. Um desses mecanismos são os túneis IPV6-sobre-IPv4.

Veremos como é possível criar túneis desse tipo para permitir interconectar nosso roteador Linux com outras ilhas/roteadores IPv6. O processo é bastante semelhante à criação de um túnel ip-ip ou gre. Começamos definindo o nosso túnel:

```
# ip tunnel add nome_do_tunnel mode sit remote endereço_remoto local endereço_local
```

Adicionamos um endereço IPv6 para o nosso lado do túnel, exemplo:

```
# ip addr add 4000::1/126 dev tunelv6
```

E levantamos o túnel:

```
# ip link set tunelv6 up
```

Agora é só acrescentar as rotas que forem necessárias. Para o caso do túnel ser nossa rota padrão poderíamos usar:

```
# ip route add ::0/0 dev tunelv6
```

A configuração da outra ponta do túnel é similar.

Radvd

O Router Advertisement Daemon é um daemon para fazer anúncios de rotas/prefixos para o Linux permitndo assim que máquinas conectadas ao roteador se autoconfigurem. Veremos como instalar e configurar esse daemon.

Para instalar o radvd no Debian basta usar o comando:

```
# apt-get install radvd
```

A configuração básica do radvd é bastante simples, basta editar o arquivo /etc/radvd.conf e ativar os anúncios e prefixos nas interfaces desejada. Exemplo:

```
interface eth0
{
  AdvSendAdvert on;
  prefix fec0::/64
  {
  };
};
```

Conclusão

Procuramos abordar nesse documento os vários aspectos, aplicações e configurações necessárias para se construir um roteador utilizando Linux. Muito mais do que um guia completo (o que seria praticamente impossível dada a quantidade de material e a velocidade com que o sistema evolui), procuramos aqui apenas indicar caminhos e possibilidades. Esperamos que o leitor vá além do conteúdo proposto. Essa busca pode começar a partir de alguns dos documentos que consultamos para escrever esse material, e que está listado em Referências.

Referências

Linux Advanced Routing and Traffic Control - http://lartc.org

Linux IP - http://linux-ip.net