



Linuxdays 2007

Netfilters and QOS

http://www.lilux.lu/presentations/2007/LinuxDays/Netfilters_QOS/

Thierry Coutelier <Thierry.Coutelier@lilux.lu>

Table of Contents

1	Netfilters.....	3
1.1	What are Netfilters.....	3
1.2	What you can do with Netfilters:.....	3
1.3	What is Needed.....	3
1.4	How to use the iptables.....	5
1.5	What is connection tracking?	6
1.6	Examples:.....	6
2	QOS.....	7
2.1	What is QOS.....	7
2.2	What you can do with QOS:.....	7
2.3	What is Needed.....	7
2.4	Kernel modules needed:.....	8
2.5	The tc commands.....	8
2.5.1	Queue disciplines:.....	8
2.5.2	Classes:.....	9
2.5.3	Filters:.....	9
2.6	Queuing disciplines.....	9
2.7	Usage with iptables.....	10
2.7.1	Using the Netfilter CLASSIFY Target.....	10
2.7.2	Using the Netfilter MARK Target.....	10
2.8	Testing tools.....	11
2.8.1	Netem.....	11
2.8.2	IPTraf.....	11
2.8.3	Netcat.....	11
2.8.4	iperf.....	12
2.9	Example.....	12
2.10	Exercise.....	12
3	Links.....	13
4	Appendix A: iptables man page.....	13
5	Appendix B: tc man pages:.....	40
6	Appendix C: A firewall sample.....	49

1 Netfilters

1.1 What are Netfilters.

Netfilters is a set of hooks in the Linux Kernel that allow to catch packets and filter, change (mangle) or transform (NAT) them.

1.2 What you can do with Netfilters:

- Select packets based on many parameters like source/destination IP address or port, state of the packet, owner of the packet or flag.
- Drop, accept, reject packets
- Mangle packets, that is mark them or change some of their flags
- NAT (Network Address Translation) which means changing their IP address (source or destination).
- Classify packets.

1.3 What is Needed

- The iptables tools. Those are included in most GNU/Linux distributions.
- A linux kernel with a version 2.2 or above.

The netfilters need to be enabled in a Linux Kernel (version 2.2 or above)

Options for 2.6 Kernel:

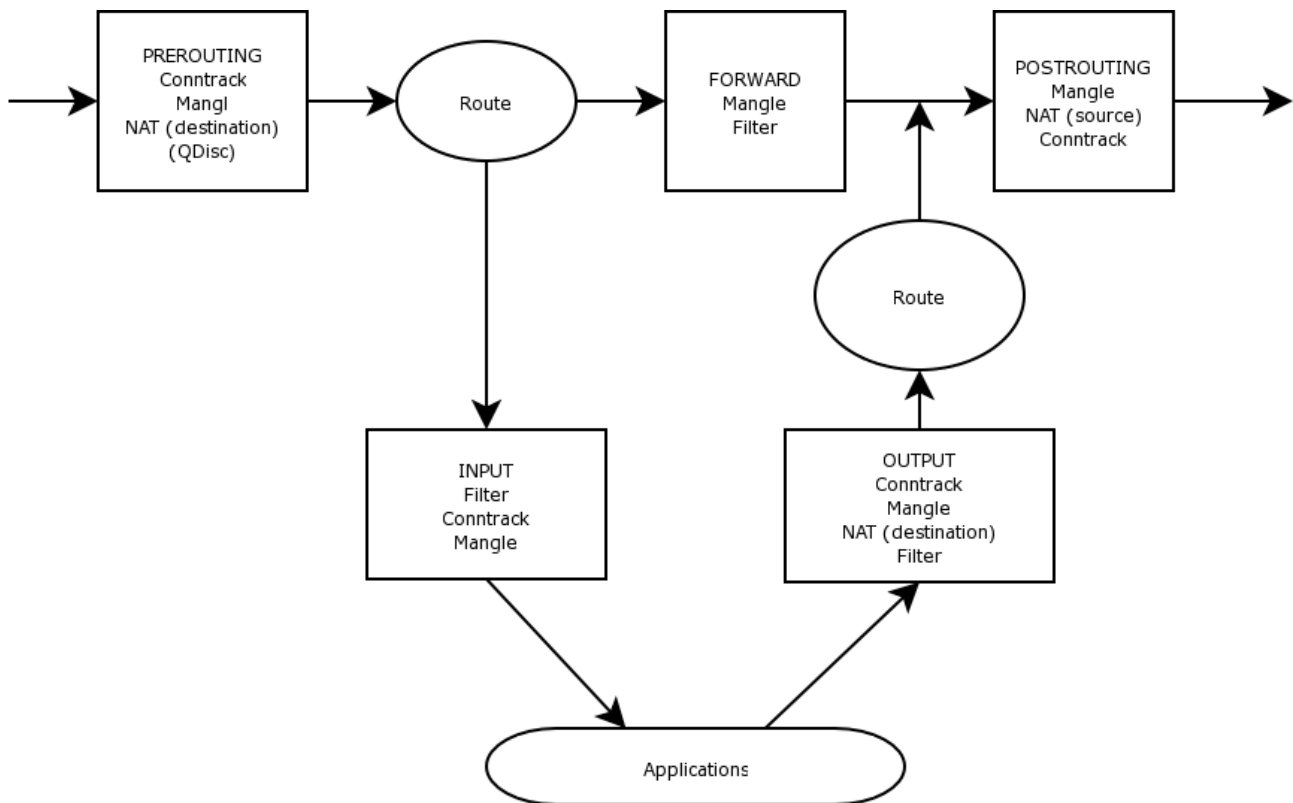
- > Device Drivers
- > Networking support
- > Networking options
- > Network packet filtering
- > IP: Netfilter Configuration

```

<M> Connection tracking (required for masq/NAT)
<M>   FTP protocol support
<M>   IRC protocol support
<M>   TFTP protocol support
<M>   Amanda backup protocol support
<M> Userspace queueing via NETLINK
<M> IP tables support (required for filtering/masq/NAT)
<M>   limit match support
<M>   IP range match support
<M>   MAC address match support
<M>   Packet type match support
<M>   IPsec policy match support
<M>   netfilter MARK match support
<M>   Multiple port match support
<M>   TOS match support
<M>   recent match support
<M>   ECN match support
<M>   DSCP match support
<M>   AH/ESP match support
<M>   LENGTH match support
<M>   TTL match support
<M>   tcpmss match support
<M>   Helper match support
<M>   Connection state match support
<M>   Connection tracking match support
<M>   Owner match support
<M>   Physdev match support
<M>   Packet filtering
<M>     REJECT target support
<M>     Full NAT
<M>     MASQUERADE target support
<M>     REDIRECT target support
<M>     NETMAP target support
<M>     SAME target support
[ ] NAT of local connections (READ HELP)
<M> Basic SNMP-ALG support (EXPERIMENTAL)
<M> Packet mangling
<M>   TOS target support
<M>   ECN target support
<M>   DSCP target support
<M>   MARK target support
<M>   CLASSIFY target support
<M> LOG target support
<M> ULOG target support
<M> TCPMSS target support
<M> ARP tables support
<M>   ARP packet filtering
<M>   ARP payload mangling
<M> ipchains (2.2-style) support
<M> ipfwadm (2.0-style) support
[*] Connection mark tracking support
<M> CONNMARK target support
<M> Connection mark match support
<M> CLUSTERIP target support

```

Network hooks and packet traversal.



1.4 How to use the iptables

First you need to know where you want to do something.

This is done by selecting the table and a chain.

The tables you may use are: filter (default), nat or mangle.

The chain may either be one of the built-in ones or one created by yourself.

Built-in chains are: INPUT, OUTPUT, FORWARD, PREROUTING and POSTROUTING (see diagram above).

Then you will have to decide what the criteria are that have to match. This is done by giving rules.

Next to you need to decide what to do. This is done by selecting a target. A target may be a user-defined chain or one of the special values: ACCEPT, DROP, QUEUE or RETURN.

You can either append (-A), insert (-I) or delete (-D) a rule. Rules may be numbered.

You may flush (-F) a chain, that is delete all the rules in a chain.

You may set the policy (-P) of a chain. The policy, for example DROP or ACCEPT (default), is

applied when there is no matching rules in a chain.

The best documentation is the man page! See Appendix A.

1.5 What is connection tracking?

Connection tracking refers to the ability to maintain state information about a connection in memory tables, such as source and destination ip address and port number pairs (known as *socket pairs*), protocol types, connection state and timeouts. Firewalls that do this are known as *stateful*. Stateful firewalling is inherently more secure than its "stateless" counterpart ... simple packet filtering.

Connection tracking is accomplished with the state option in iptables.

Connection tracking is done either in the **PREROUTING** chain, or the **OUTPUT** chain for locally generated packets.

Connection tracking defragments all packets before tracking their state. This explains why there is no `ip_always_defrag` switch as there was in the 2.2 kernel.

The state table for udp and tcp connections is maintained in `/proc/net/ip_conntrack`.

The maximum number of connections the state table can contain is stored in `/proc/sys/net/ipv4/ip_conntrack_max`. This value is determined initially by how much physical memory you have (on my 512 Mb machine, `ip_conntrack_max` = 32760 by default).

1.6 Examples:

See appendix C for a complete sample script to do firewalling.

2 QOS

2.1 What is QOS

QOS stands for Quality of Service and permits a set of operations based on network packets. The operations include enqueueing, policing, classifying, scheduling, shaping and dropping. QOS is generally configured on a network interface.

2.2 What you can do with QOS:

- Limit total bandwidth to a known rate; [TBF](#), [HTB](#) with child class(es).
- Limit the bandwidth of a particular user, service or client; [HTB](#) classes and [classifying](#) with a [filter](#). traffic.
- Maximize TCP throughput on an asymmetric link; prioritize transmission of ACK packets, [wondershaper](#).
- Reserve bandwidth for a particular application or user; [HTB](#) with children classes and [classifying](#).
- Prefer latency sensitive traffic; [PRIO](#) inside an [HTB](#) class.
- Managed oversubscribed bandwidth; [HTB](#) with borrowing.
- Allow equitable distribution of unreserved bandwidth; [HTB](#) with borrowing.
- Ensure that a particular type of traffic is dropped; [policer](#) attached to a [filter](#) with a [drop](#) action.

2.3 What is Needed.

- A Linux 2.4.x or 2.6 kernel.
- QOS enabled in the kernel (most of the GNU/Linux distributions include such a kernel).
- The iproute2 package (also included in most distributions).

2.4 Kernel modules needed:

in /usr/src/linux (or where your kernel resides)

make menuconfig

-> Device Drivers

-> Networking support

-> Networking options

-> QoS and/or fair queueing

```
[*] QoS and/or fair queueing
    <M> CBQ packet scheduler
    <M> HTB packet scheduler
    <M> HFSC packet scheduler
    <M> CSZ packet sched
    <*> ATM pseudo-scheduler
    <M> The simplest PRIIO pseudoscheduler
    <M> RED queue
    <M> SFQ queue
    <M> TEQL queue
    <M> TBF queue
    <M> GRED queue
    <M> Diffserv field marker
    <M> Delay simulator
    <M> Ingress Qdisc
    [*] QoS support
        [*] Rate estimator
        [*] Packet classifier API
            <M> TC index classifier
            <M> Routing table based classifier
            <M> Firewall based classifier
            <M> U32 classifier
            <M> Special RSVP classifier
            <M> Special RSVP classifier for Ipv6
        [*] Traffic policing (needed for in/egress)
```

2.5The tc commands

From the user space the iproute2 packages offers two major commands:

- ip -> this is used to configure routing tables and network links (network adapters)
- tc -> this is the one used to configure the different parts of the QOS.

The tc command takes as a first parameter the object you want to work on:

Either: qdisc, class or filter.

tc help

```
Usage: tc [ OPTIONS ] OBJECT { COMMAND | help }
where  OBJECT := { qdisc | class | filter }
       OPTIONS := { -s[tatistics] | -d[etails] | -r[aw] | -b[atch] file }
```

2.5.1 Queue disciplines:

This is used to set the kind of queue you want to use on a specific interface.

Depending on the queue type the parameters will be different.

It is the first command you will use.

tc qdisc help

```
Usage: tc qdisc [ add | del | replace | change | get ] dev STRING
       [ handle QHANDLE ] [ root | ingress | parent CLASSID ]
       [ estimator INTERVAL TIME_CONSTANT ]
       [ [ QDISC_KIND ] [ help | OPTIONS ] ]
```

```
tc qdisc show [ dev STRING ] [ingress]
```

Where:

```
QDISC_KIND := { [p|b]fifo | tbf | prio | cbq | red | etc. }
OPTIONS := ... try tc qdisc add <desired QDISC_KIND> help
```

2.5.2 Classes:

tc class is used to configure a classes.

tc class help

```
Usage: tc class [ add | del | change | get ] dev STRING
       [ classid CLASSID ] [ root | parent CLASSID ]
       [ [ QDISC_KIND ] [ help | OPTIONS ] ]
```

```
tc class show [ dev STRING ] [ root | parent CLASSID ]
```

Where:

```
QDISC_KIND := { prio | cbq | etc. }
OPTIONS := ... try tc class add <desired QDISC_KIND> help
```

2.5.3 Filters:

Used to classify packets depending on their contents.

tc filter help

```
Usage: tc filter [ add | del | change | get ] dev STRING
       [ pref PRIO ] [ protocol PROTO ]
```

```
[ estimator INTERVAL TIME_CONSTANT ]
[ root | classid CLASSID ] [ handle FILTERID ]
[ [ FILTER_TYPE ] [ help | OPTIONS ] ]
```

```
tc filter show [ dev STRING ] [ root | parent CLASSID ]
```

Where:

`FILTER_TYPE` := { rsvp | u32 | fw | route | etc. }

`FILTERID` := ... format depends on classifier, see there

`OPTIONS` := ... try `tc filter add <desired FILTER_KIND> help`

2.6 Queuing disciplines

There are two types of qdisc.

1. Classless queuing disciplines. Like pfifo, prio, sfq ... those reorder packets based on some criteria.
2. Classful queuing disciplines. Like CQB, HTB ... for those packets may be switched to different classes.

The most used qdiscs are HTB to do bandwidth limitation, pfifo which is the default of any queue, and sfq which is used to do fair queuing on an interface.

2.7 Usage with iptables

2.7.1 Using the Netfilter CLASSIFY Target

Since Linux 2.6 the CLASSIFY target has been part of the standard distribution, so you need not patch your kernel. The CLASSIFY extension was added to *Netfilter* in version 1.2.9.

```
iptables -t mangle -A POSTROUTING -o eth2 -p tcp --sport 80 -j CLASSIFY --set-class 1:10
```

Briefly, iptables is being instructed to append a rule to the POSTROUTING section of mangle table. The rule matches TCP packets with a source port of 80 that are passing out of the eth2 network interface. The target of this rule is the CLASSIFY extension, which is directed to classify this traffic into the class described by the major node number 1 and the minor node number 10. The careful reader will notice that, based on the minor node number being greater than zero, the target must be a class assigned to a classful qdisc.

You can only use CLASSIFY from the POSTROUTING chain of the mangle table. It is prohibited elsewhere. If you find you need to classify packets elsewhere, you may need to use the MARK target instead.

2.7.2 Using the Netfilter MARK Target

If you cannot use the CLASSIFY target, you can use the mark target in conjunction with **tc** to classify flows.

```
iptables -t mangle -A POSTROUTING -o eth2 -p tcp --sport 80 -j MARK --set-mark 1
```

The above iptables rule will set an invisible mark on any packet it matches. The mark exists in kernel space only. The packet is not actually modified. The **tc** binary can be used to classify flows based on these marks.

```
tc filter add dev eth0 protocol ip parent 1:0 prio 1 handle 1 fw classid 1:10
```

The above **tc** command is not unlike the familiar qdisc and class variants, except now you're adding a filter instead. The *parent* parameter will always refer to the root qdisc for the given interface, which must exist prior to creating the filter. The actual parameter *handle* refers to the mark that you gave the flow earlier. The parameter *classid* refers to, unsurprisingly, the *handle* of the class you wish to assign this flow to. It's generally only useful to add filters for interfaces which have classful qdiscs configured.

2.8 Testing tools

2.8.1 Netem

Netem allows emulating the properties of wide area networks. The current version emulates variable delay, loss, duplication and re-ordering.network emulator.

<http://developer.osdl.org/shemminger/netem>

2.8.2 IPTraf

IPTraf is a console-based network statistics utility for Linux. It gathers a variety of figures such as TCP connection packet and byte counts, interface statistics and activity indicators, TCP/UDP traffic breakdowns, and LAN station packet and byte counts.

<http://iptraf.seul.org/>

2.8.3 Netcat



Netcat is a featured networking utility which reads and writes data across network connections, using the TCP/IP protocol.

It is designed to be a reliable "back-end" tool that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool, since it can create almost any kind of connection you would need and has several interesting built-in capabilities.

<http://netcat.sourceforge.net/>

2.8.4 iperf

Iperf is a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss.

<http://dast.nlanr.net/Projects/Iperf/>

2.9 Example

```
tc qdisc add dev eth2 parent root handle 1:0 htb default 20
tc class add dev eth2 parent 1:0 classid 1:1 htb rate 1000kbit
tc class add dev eth2 parent 1:1 classid 1:10 htb rate 500kbit
tc class add dev eth2 parent 1:1 classid 1:20 htb rate 500kbit
tc qdisc add dev eth2 parent 1:20 handle 2:0 sfq
```

We have a nested structure, with a **htb** classful qdisc assigned to the root hook, three **htb** classes, and a **sfq** qdisc as a leaf qdisc for one **htb** class. The other has an implicit **pfifo** attached. The careful reader will notice each qdisc has a minor node number of zero, as is required.

At the top of the hierarchy is a **htb** qdisc. Three classes are assigned to it. Only the first is immediately attached to it, using the *parent* 1:0. The other two classes are children of the first class. If you examine the **tc** command with the *class* option, you will see that the *parent* refers to the parent class in the hierarchy via its *classid*.

Each of the three **htb** classes attached to the **htb** qdisc are assigned a major node number of 1 for the *classid*, as the qdisc they are attached to has a *handle* with 1 as the major node number. The minor node number for each *classid* must merely be a unique number between 1 and ffff in hexadecimal.

Finally, a **sfq** qdisc is attached to the leaf class with *classid* 1:20. Notice the qdisc is added nearly

the same as the **htb**. However, instead of being assigned to the magic *root* hook, the target is 1:20. The *handle* is chosen based on the rules discussed earlier. Briefly, the major node number must be a unique number between 1 and ffff and the minor node must be 0.

Last, the whole structure can be deleted simply by deleting the root hook as demonstrated below.

```
tc qdisc del dev eth2 root
```

2.10 Exercise

Set the main outgoing max rate for the first interface to 600kbit/s

Limit outgoing port 6667 traffic to 100kbit/s

Limit outgoing port 6668 traffic to 200kbit/s

Limit the rest of the traffic to 500kbit/s

Use one filter and one iptables rule.

Tools: iperf

On the destination server:

```
./iperf -s -p 6667 -i 1
```

```
./iperf -s -p 6668 -i 1
```

```
./iperf -s -p 6669 -i 1
```

On the source PC (where the rules are added)

```
./iperf -c destination.host -p 6667
```

```
./iperf -c destination.host -p 6668
```

```
./iperf -c destination.host -p 6669
```

3 Links

- The basic must read for all networking: <http://lartc.org/>
- Complete documentation about linux networking: http://www.faqs.org/docs/linux_network/
- Detailed description of netfilter hooks (for programming):
<http://uqconnect.net/~zzoklan/documents/netfilter.html>
- Detailed description of QOS:
http://www.trekweb.com/~jasonb/articles/traffic_shaping/index.html
- Good examples: <http://www.docum.org/docum.org/>
- In depth QOS: <http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm>

4 Appendix A: iptables man page

IPTABLES(8)

IPTABLES(8)

NAME

iptables - administration tool for IPv4 packet filtering and NAT

SYNOPSIS

```
iptables [-t table] [-[AD] chain rule-specification
[options]
iptables [-t table] -I chain [rulenum] rule-specification
[options]
iptables [-t table] -R chain rulenum rule-specification
[options]
iptables [-t table] -D chain rulenum [options]
iptables [-t table] -[LFZ] [chain] [options]
iptables [-t table] -N chain
iptables [-t table] -X [chain]
iptables [-t table] -P chain target [options]
iptables [-t table] -E old-chain-name new-chain-name
```

DESCRIPTION

Iptables is used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel. Several different tables may be defined. Each table contains a number of built-in chains and may also contain user-defined chains.

Each chain is a list of rules which can match a set of packets. Each rule specifies what to do with a packet that matches. This is called a 'target', which may be a jump to a user-defined chain in the same table.

TARGETS

A firewall rule specifies criteria for a packet, and a target. If the packet does not match, the next rule in the chain is the examined; if it does match, then the next rule is specified by the value of the target, which can be the name of a user-defined chain or one of the special values ACCEPT, DROP, QUEUE, or RETURN.

ACCEPT means to let the packet through. DROP means to drop the packet on the floor. QUEUE means to pass the packet to userspace (if supported by the kernel). RETURN means stop traversing this chain and resume at the next rule in the previous (calling) chain. If the end of a built-in chain is reached or a rule in a built-in chain with target RETURN is matched, the target specified by the chain policy determines the fate of the packet.

TABLES

There are currently three independent tables (which tables are present at any time depends on the kernel configuration options and which modules are present).

`-t, --table table`

This option specifies the packet matching table which the command should operate on. If the kernel is configured with automatic module loading, an attempt will be made to load the appropriate module for that table if it is not already there.

The tables are as follows:

`filter:`

This is the default table (if no `-t` option is passed). It contains the built-in chains INPUT (for packets coming into the box itself), FORWARD (for packets being routed through the

box), and OUTPUT (for locally-generated packets).

nat:

This table is consulted when a packet that creates a new connection is encountered. It consists of three built-ins: PREROUTING (for altering packets as soon as they come in), OUTPUT (for altering locally-generated packets before routing), and POSTROUTING (for altering packets as they are about to go out).

mangle:

This table is used for specialized packet alteration. Until kernel 2.4.17 it had two built-in chains: PREROUTING (for altering incoming packets before routing) and OUTPUT (for altering locally-generated packets before routing). Since kernel 2.4.18, three other built-in chains are also supported: INPUT (for packets coming into the box itself), FORWARD (for altering packets being routed through the box), and POSTROUTING (for altering packets as they are about to go out).

OPTIONS

The options that are recognized by iptables can be divided into several different groups.

COMMANDS

These options specify the specific action to perform. Only one of them can be specified on the command line unless otherwise specified below. For all the long versions of the command and option names, you need to use only enough letters to ensure that iptables can differentiate it from all other options.

`-A, --append chain rule-specification`

Append one or more rules to the end of the selected chain. When the source and/or destination names resolve to more than one address, a rule will be added for each possible address combination.

`-D, --delete chain rule-specification`

`-D, --delete chain rulenum`

Delete one or more rules from the selected chain. There are two versions of this command: the rule can be specified as a number in the chain (starting at 1 for the first rule) or a rule to match.

`-I, --insert chain [rulenum] rule-specification`

Insert one or more rules in the selected chain as the given rule number. So, if the rule number is 1, the rule or rules are inserted at the head of the chain. This is also the default if no rule number is specified.

`-R, --replace chain rulenum rule-specification`

Replace a rule in the selected chain. If the source and/or destination names resolve to multiple addresses, the command will fail. Rules are numbered starting at 1.

`-L, --list [chain]`

List all rules in the selected chain. If no chain is selected, all chains are listed. As every other iptables command, it applies to the specified table (filter is the default), so NAT rules get listed by `iptables -t nat -n -L`

Please note that it is often used with the `-n` option, in order to avoid long reverse DNS lookups. It is legal to specify the `-Z` (zero) option as well, in which case the chain(s) will be atomically listed and zeroed. The exact output is affected by

the other arguments given. The exact rules are suppressed until you use

```
iptables -L -v
```

-F, --flush [chain]

Flush the selected chain (all the chains in the table if none is given). This is equivalent to deleting all the rules one by one.

-Z, --zero [chain]

Zero the packet and byte counters in all chains. It is legal to specify the **-L, --list (list)** option as well, to see the counters immediately before they are cleared. (See above.)

-N, --new-chain chain

Create a new user-defined chain by the given name. There must be no target of that name already.

-X, --delete-chain [chain]

Delete the optional user-defined chain specified. There must be no references to the chain. If there are, you must delete or replace the referring rules before the chain can be deleted. If no argument is given, it will attempt to delete every non-builtin chain in the table.

-P, --policy chain target

Set the policy for the chain to the given target. See the section **TARGETS** for the legal targets. Only built-in (non-user-defined) chains can have policies, and neither built-in nor user-defined chains can be policy targets.

-E, --rename-chain old-chain new-chain

Rename the user specified chain to the user supplied name. This is cosmetic, and has no effect on

the structure of the table.

-h Help. Give a (currently very brief) description of the command syntax.

PARAMETERS

The following parameters make up a rule specification (as used in the add, delete, insert, replace and append commands).

-p, --protocol [!] protocol

The protocol of the rule or of the packet to check. The specified protocol can be one of tcp, udp, icmp, or all, or it can be a numeric value, representing one of these protocols or a different one. A protocol name from /etc/protocols is also allowed. A "!" argument before the protocol inverts the test. The number zero is equivalent to all. Protocol all will match with all protocols and is taken as default when this option is omitted.

-s, --source [!] address[/mask]

Source specification. Address can be either a network name, a hostname (please note that specifying any name to be resolved with a remote query such as DNS is a really bad idea), a network IP address (with /mask), or a plain IP address. The mask can be either a network mask or a plain number, specifying the number of 1's at the left side of the network mask. Thus, a mask of 24 is equivalent to 255.255.255.0. A "!" argument before the address specification inverts the sense of the address. The flag --src is an alias for this option.

-d, --destination [!] address[/mask]

Destination specification. See the description of

the `-s` (source) flag for a detailed description of the syntax. The flag `--dst` is an alias for this option.

`-j, --jump target`

This specifies the target of the rule; i.e., what to do if the packet matches it. The target can be a user-defined chain (other than the one this rule is in), one of the special builtin targets which decide the fate of the packet immediately, or an extension (see EXTENSIONS below). If this option is omitted in a rule, then matching the rule will have no effect on the packet's fate, but the counters on the rule will be incremented.

`-i, --in-interface [!] name`

Name of an interface via which a packet is going to be received (only for packets entering the INPUT, FORWARD and PREROUTING chains). When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match. If this option is omitted, any interface name will match.

`-o, --out-interface [!] name`

Name of an interface via which a packet is going to be sent (for packets entering the FORWARD, OUTPUT and POSTROUTING chains). When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match. If this option is omitted, any interface name will match.

[!] `-f, --fragment`

This means that the rule only refers to second and

further fragments of fragmented packets. Since there is no way to tell the source or destination ports of such a packet (or ICMP type), such a packet will not match any rules which specify them. When the "!" argument precedes the "-f" flag, the rule will only match head fragments, or unfragmented packets.

`-c, --set-counters PKTS BYTES`

This enables the administrator to initialize the packet and byte counters of a rule (during INSERT, APPEND, REPLACE operations).

OTHER OPTIONS

The following additional options can be specified:

`-v, --verbose`

Verbose output. This option makes the list command show the interface name, the rule options (if any), and the TOS masks. The packet and byte counters are also listed, with the suffix 'K', 'M' or 'G' for 1000, 1,000,000 and 1,000,000,000 multipliers respectively (but see the `-x` flag to change this). For appending, insertion, deletion and replacement, this causes detailed information on the rule or rules to be printed.

`-n, --numeric`

Numeric output. IP addresses and port numbers will be printed in numeric format. By default, the program will try to display them as host names, network names, or services (whenever applicable).

`-x, --exact`

Expand numbers. Display the exact value of the packet and byte counters, instead of only the rounded number in K's (multiples of 1000) M's (mul

tuples of 1000K) or G's (multiples of 1000M). This option is only relevant for the -L command.

--line-numbers

When listing rules, add line numbers to the beginning of each rule, corresponding to that rule's position in the chain.

--modprobe=command

When adding or inserting rules into a chain, use command to load any necessary modules (targets, match extensions, etc).

MATCH EXTENSIONS

iptables can use extended packet matching modules. These are loaded in two ways: implicitly, when -p or --protocol is specified, or with the -m or --match options, followed by the matching module name; after these, various extra command line options become available, depending on the specific module. You can specify multiple extended match modules in one line, and you can use the -h or --help options after the module has been specified to receive help specific to that module.

The following are included in the base package, and most of these can be preceded by a ! to invert the sense of the match.

ah

This module matches the SPIs in AH header of IPsec packets.

--ahspi [!] spi[:spi]

conntrack

This module, when combined with connection tracking, allows access to more connection tracking information than

the "state" match. (this module is present only if iptables was compiled under a kernel supporting this feature)

`--ctstate state`

Where state is a comma separated list of the connection states to match. Possible states are INVALID meaning that the packet is associated with no known connection, ESTABLISHED meaning that the packet is associated with a connection which has seen packets in both directions, NEW meaning that the packet has started a new connection, or otherwise associated with a connection which has not seen packets in both directions, and RELATED meaning that the packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer, or an ICMP error. SNAT A virtual state, matching if the original source address differs from the reply destination. DNAT A virtual state, matching if the original destination differs from the reply source.

`--ctproto proto`

Protocol to match (by number or name)

`--ctorigsrc [!] address[/mask]`

Match against original source address

`--ctorigdst [!] address[/mask]`

Match against original destination address

`--ctreplsrc [!] address[/mask]`

Match against reply source address

`--ctrepldst [!] address[/mask]`

Match against reply destination address

`--ctstatus [NONE|EXPECTED|SEEN_REPLY|ASSURED][,...]`

Match against internal conntrack states

`--ctexpire time[:time]`

Match remaining lifetime in seconds against given value or range of values (inclusive)

dscp

This module matches the 6 bit DSCP field within the TOS field in the IP header. DSCP has superseded TOS within the IETF.

`--dscp value`

Match against a numeric (decimal or hex) value [0-32].

`--dscp-class DiffServ Class`

Match the DiffServ class. This value may be any of the BE, EF, AFxx or CSx classes. It will then be converted into it's according numeric value.

esp

This module matches the SPIs in ESP header of IPsec packets.

`--espspi [!] spi[:spi]`

helper

This module matches packets related to a specific conntrack-helper.

`--helper string`

Matches packets related to the specified conntrack-helper.

string can be "ftp" for packets related to a ftp-session on default port. For other ports append -portnr to the value, ie. "ftp-2121".

Same rules apply for other conntrack-helpers.

icmp

This extension is loaded if '--protocol icmp' is specified. It provides the following option:

`--icmp-type [!] typename`

This allows specification of the ICMP type, which can be a numeric ICMP type, or one of the ICMP type names shown by the command

```
iptables -p icmp -h
```

length

This module matches the length of a packet against a specific value or range of values.

`--length length[:length]`

limit

This module matches at a limited rate using a token bucket filter. A rule using this extension will match until this limit is reached (unless the '!' flag is used). It can be used in combination with the LOG target to give limited logging, for example.

`--limit rate`

Maximum average matching rate: specified as a number, with an optional '/second', '/minute', '/hour', or '/day' suffix; the default is 3/hour.

`--limit-burst number`

Maximum initial number of packets to match: this number gets recharged by one every time the limit specified above is not reached, up to this number; the default is 5.

mac

`--mac-source [!] address`

Match source MAC address. It must be of the form `XX:XX:XX:XX:XX:XX`. Note that this only makes sense for packets coming from an Ethernet device and entering the `PREROUTING`, `FORWARD` or `INPUT` chains.

mark

This module matches the netfilter `mark` field associated with a packet (which can be set using the `MARK` target below).

`--mark value[/mask]`

Matches packets with the given unsigned `mark` value (if a `mask` is specified, this is logically ANDed with the `mask` before the comparison).

multiport

This module matches a set of source or destination ports. Up to 15 ports can be specified. It can only be used in conjunction with `-p tcp` or `-p udp`.

`--source-ports port[,port[,port...]]`

Match if the source port is one of the given ports. The flag `--sports` is a convenient alias for this option.

`--destination-ports port[,port[,port...]]`

Match if the destination port is one of the given ports. The flag `--dports` is a convenient alias for this option.

`--ports port[,port[,port...]]`

Match if the both the source and destination ports are equal to each other and to one of the given ports.

owner

This module attempts to match various characteristics of the packet creator, for locally-generated packets. It is only valid in the OUTPUT chain, and even this some packets (such as ICMP ping responses) may have no owner, and hence never match.

--uid-owner userid

Matches if the packet was created by a process with the given effective user id.

--gid-owner groupid

Matches if the packet was created by a process with the given effective group id.

--pid-owner processid

Matches if the packet was created by a process with the given process id.

--sid-owner sessionid

Matches if the packet was created by a process in the given session group.

--cmd-owner name

Matches if the packet was created by a process with the given command name. (this option is present only if iptables was compiled under a kernel supporting this feature)

physdev

This module matches on the bridge port input and output devices enslaved to a bridge device. This module is a part of the infrastructure that enables a transparent bridging IP firewall and is only useful for kernel versions above version 2.5.44.

--physdev-in name

Name of a bridge port via which a packet is received (only for packets entering the INPUT, FORWARD and PREROUTING chains). If the interface name ends in a "+", then any interface which begins with this name will match. If the packet didn't arrive through a bridge device, this packet won't match this option, unless '!' is used.

`--physdev-out name`

Name of a bridge port via which a packet is going to be sent (for packets entering the FORWARD, OUTPUT and POSTROUTING chains). If the interface name ends in a "+", then any interface which begins with this name will match. Note that in the nat and mangle OUTPUT chains one cannot match on the bridge output port, however one can in the filter OUTPUT chain. If the packet won't leave by a bridge device or it is yet unknown what the output device will be, then the packet won't match this option, unless

`--physdev-is-in`

Matches if the packet has entered through a bridge interface.

`--physdev-is-out`

Matches if the packet will leave through a bridge interface.

`--physdev-is-bridged`

Matches if the packet is being bridged and there fore is not being routed. This is only useful in the FORWARD and POSTROUTING chains.

`pkttype`

This module matches the link-layer packet type.

`--pkt-type [unicast|broadcast|multicast]`

state

This module, when combined with connection tracking, allows access to the connection tracking state for this packet.

--state state

Where state is a comma separated list of the connection states to match. Possible states are INVALID meaning that the packet could not be identified for some reason which includes running out of memory and ICMP errors which don't correspond to any known connection, ESTABLISHED meaning that the packet is associated with a connection which has seen packets in both directions, NEW meaning that the packet has started a new connection, or otherwise associated with a connection which has not seen packets in both directions, and RELATED meaning that the packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer, or an ICMP error.

tcp

These extensions are loaded if '--protocol tcp' is specified. It provides the following options:

--source-port [!] port[:port]

Source port or port range specification. This can either be a service name or a port number. An inclusive range can also be specified, using the format port:port. If the first port is omitted, "0" is assumed; if the last is omitted, "65535" is assumed. If the second port greater than the first they will be swapped. The flag --sport is a convenient alias for this option.

--destination-port [!] port[:port]

Destination port or port range specification. The flag `--dport` is a convenient alias for this option.

`--tcp-flags [!] mask comp`

Match when the TCP flags are as specified. The first argument is the flags which we should examine, written as a comma-separated list, and the second argument is a comma-separated list of flags which must be set. Flags are: SYN ACK FIN RST URG PSH ALL NONE. Hence the command

```
iptables -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST SYN
```

will only match packets with the SYN flag set, and the ACK, FIN and RST flags unset.

`[!] --syn`

Only match TCP packets with the SYN bit set and the ACK and RST bits cleared. Such packets are used to request TCP connection initiation; for example, blocking such packets coming in an interface will prevent incoming TCP connections, but outgoing TCP connections will be unaffected. It is equivalent to `--tcp-flags SYN,RST,ACK SYN`. If the `!"` flag precedes the `--syn`, the sense of the option is inverted.

`--tcp-option [!] number`

Match if TCP option set.

`--mss value[:value]`

Match TCP SYN or SYN/ACK packets with the specified MSS value (or range), which control the maximum packet size for that connection.

`tos`

This module matches the 8 bits of Type of Service field in the IP header (ie. including the precedence bits).

`--tos tos`

The argument is either a standard name, (use `iptables -m tos -h` to see the list), or a numeric value to match.

`ttl`

This module matches the time to live field in the IP header.

`--ttl ttl`

Matches the given TTL value.

`udp`

These extensions are loaded if '`--protocol udp`' is specified. It provides the following options:

`--source-port [!] port[:port]`

Source port or port range specification. See the description of the `--source-port` option of the TCP extension for details.

`--destination-port [!] port[:port]`

Destination port or port range specification. See the description of the `--destination-port` option of the TCP extension for details.

`unclean`

This module takes no options, but attempts to match packets which seem malformed or unusual. This is regarded as experimental.

TARGET EXTENSIONS

`iptables` can use extended target modules: the following are included in the standard distribution.

`DNAT`

This target is only valid in the `nat` table, in the `PRE`

ROUTING and OUTPUT chains, and user-defined chains which are only called from those chains. It specifies that the destination address of the packet should be modified (and all future packets in this connection will also be mangled), and rules should cease being examined. It takes one type of option:

`--to-destination ipaddr[-ipaddr][:port-port]`

which can specify a single new destination IP address, an inclusive range of IP addresses, and optionally, a port range (which is only valid if the rule also specifies `-p tcp` or `-p udp`). If no port range is specified, then the destination port will never be modified.

You can add several `--to-destination` options. If you specify more than one destination address, either via an address range or multiple `--to-destination` options, a simple round-robin (one after another in cycle) load balancing takes place between these addresses.

DSCP

This target allows to alter the value of the DSCP bits within the TOS header of the IPv4 packet. As this manipulates a packet, it can only be used in the mangle table.

`--set-dscp value`

Set the DSCP field to a numerical value (can be decimal or hex)

`--set-dscp-class class`

Set the DSCP field to a DiffServ class.

ECN

This target allows to selectively work around known ECN blackholes. It can only be used in the mangle table.

`--ecn-tcp-remove`

Remove all ECN bits from the TCP header. Of course, it can only be used in conjunction with `-p tcp`.

LOG

Turn on kernel logging of matching packets. When this option is set for a rule, the Linux kernel will print some information on all matching packets (like most IP header fields) via the kernel log (where it can be read with `dmesg` or `syslogd(8)`). This is a "non-terminating target", i.e. rule traversal continues at the next rule. So if you want to LOG the packets you refuse, use two separate rules with the same matching criteria, first using target LOG then DROP (or REJECT).

`--log-level level`

Level of logging (numeric or see `syslog.conf(5)`).

`--log-prefix prefix`

Prefix log messages with the specified prefix; up to 29 letters long, and useful for distinguishing messages in the logs.

`--log-tcp-sequence`

Log TCP sequence numbers. This is a security risk if the log is readable by users.

`--log-tcp-options`

Log options from the TCP packet header.

`--log-ip-options`

Log options from the IP packet header.

MARK

This is used to set the netfilter mark value associated

with the packet. It is only valid in the mangle table. It can for example be used in conjunction with iproute2.

`--set-mark mark`

MASQUERADE

This target is only valid in the nat table, in the POSTROUTING chain. It should only be used with dynamically assigned IP (dialup) connections: if you have a static IP address, you should use the SNAT target. Masquerading is equivalent to specifying a mapping to the IP address of the interface the packet is going out, but also has the effect that connections are forgotten when the interface goes down. This is the correct behavior when the next dialup is unlikely to have the same interface address (and hence any established connections are lost anyway). It takes one option:

`--to-ports port[-port]`

This specifies a range of source ports to use, overriding the default SNAT source port-selection heuristics (see above). This is only valid if the rule also specifies `-p tcp` or `-p udp`.

MIRROR

This is an experimental demonstration target which inverts the source and destination fields in the IP header and retransmits the packet. It is only valid in the INPUT, FORWARD and PREROUTING chains, and user-defined chains which are only called from those chains. Note that the outgoing packets are NOT seen by any packet filtering chains, connection tracking or NAT, to avoid loops and other problems.

REDIRECT

This target is only valid in the nat table, in the PRE ROUTING and OUTPUT chains, and user-defined chains which

are only called from those chains. It alters the destination IP address to send the packet to the machine itself (locally-generated packets are mapped to the 127.0.0.1 address). It takes one option:

`--to-ports port[-port]`

This specifies a destination port or range of ports to use: without this, the destination port is never altered. This is only valid if the rule also specifies `-p tcp` or `-p udp`.

REJECT

This is used to send back an error packet in response to the matched packet: otherwise it is equivalent to DROP so it is a terminating TARGET, ending rule traversal. This target is only valid in the INPUT, FORWARD and OUTPUT chains, and user-defined chains which are only called from those chains. The following option controls the nature of the error packet returned:

`--reject-with type`

The type given can be
 `icmp-net-unreachable`
 `icmp-host-unreachable`
 `icmp-port-unreachable`
 `icmp-proto-unreachable`
 `icmp-net-prohibited`
 `icmp-host-prohibited` or
 `icmp-admin-prohibited (*)`

which return the appropriate ICMP error message (port-unreachable is the default). The option `tcp-reset` can be used on rules which only match the TCP protocol: this causes a TCP RST packet to be sent back. This is mainly useful for blocking ident (113/tcp) probes which frequently occur when sending mail to broken mail hosts (which won't accept your mail otherwise).

(*) Using `icmp-admin-prohibited` with kernels that do not support it will result in a plain DROP instead of REJECT

SNAT

This target is only valid in the `nat` table, in the `POSTROUTING` chain. It specifies that the source address of the packet should be modified (and all future packets in this connection will also be mangled), and rules should cease being examined. It takes one type of option:

```
--to-source ipaddr[-ipaddr][:port-port]
```

which can specify a single new source IP address, an inclusive range of IP addresses, and optionally, a port range (which is only valid if the rule also specifies `-p tcp` or `-p udp`). If no port range is specified, then source ports below 512 will be mapped to other ports below 512: those between 512 and 1023 inclusive will be mapped to ports below 1024, and other ports will be mapped to 1024 or above. Where possible, no port alteration will occur.

You can add several `--to-source` options. If you specify more than one source address, either via an address range or multiple `--to-source` options, a simple round-robin (one after another in cycle) takes place between these addresses.

TCPMSS

This target allows to alter the MSS value of TCP SYN packets, to control the maximum size for that connection (usually limiting it to your outgoing interface's MTU minus 40). Of course, it can only be used in conjunction with `-p tcp`.

This target is used to overcome criminally braindead ISPs or servers which block ICMP Fragmentation Needed packets.

The symptoms of this problem are that everything works fine from your Linux firewall/router, but machines behind it can never exchange large packets:

- 1) Web browsers connect, then hang with no data received.
- 2) Small mail works fine, but large emails hang.
- 3) ssh works fine, but scp hangs after initial handshaking.

Workaround: activate this option and add a rule to your firewall configuration like:

```
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN \
        -j TCPMSS --clamp-mss-to-pmtu
```

`--set-mss value`

Explicitly set MSS option to specified value.

`--clamp-mss-to-pmtu`

Automatically clamp MSS value to (path_MTU - 40).

These options are mutually exclusive.

TOS

This is used to set the 8-bit Type of Service field in the IP header. It is only valid in the mangle table.

`--set-tos tos`

You can use a numeric TOS values, or use

```
iptables -j TOS -h
```

to see the list of valid TOS names.

ULOG

This target provides userspace logging of matching packets. When this target is set for a rule, the Linux kernel will multicast this packet through a netlink socket. One or more userspace processes may then subscribe to various multicast groups and receive the packets. Like LOG, this is a "non-terminating target", i.e. rule traversal continues at the next rule.

`--ulog-nlgroup nlgroup`

This specifies the netlink group (1-32) to which the packet is sent. Default value is 1.

`--ulog-prefix prefix`

Prefix log messages with the specified prefix; up to 32 characters long, and useful for distinguishing messages in the logs.

`--ulog-cprange size`

Number of bytes to be copied to userspace. A value of 0 always copies the entire packet, regardless of its size. Default is 0.

`--ulog-qthreshold size`

Number of packet to queue inside kernel. Setting this value to, e.g. 10 accumulates ten packets inside the kernel and transmits them as one netlink multipart message to userspace. Default is 1 (for backwards compatibility).

DIAGNOSTICS

Various error messages are printed to standard error. The exit code is 0 for correct functioning. Errors which appear to be caused by invalid or abused command line parameters cause an exit code of 2, and other errors cause an exit code of 1.

BUGS

Bugs? What's this? ;-) Well... the counters are not reliable on sparc64.

COMPATIBILITY WITH IPCHAINS

This iptables is very similar to ipchains by Rusty Russell. The main difference is that the chains INPUT and OUTPUT are only traversed for packets coming into the

local host and originating from the local host respectively. Hence every packet only passes through one of the three chains (except loopback traffic, which involves both INPUT and OUTPUT chains); previously a forwarded packet would pass through all three.

The other main difference is that `-i` refers to the input interface; `-o` refers to the output interface, and both are available for packets entering the FORWARD chain.

iptables is a pure packet filter when using the default 'filter' table, with optional extension modules. This should simplify much of the previous confusion over the combination of IP masquerading and packet filtering seen previously. So the following options are handled differently:

- `-j MASQ`
- `-M -S`
- `-M -L`

There are several other changes in iptables.

SEE ALSO

iptables-save(8), iptables-restore(8), ip6tables(8), ip6tables-save(8), ip6tables-restore(8).

The packet-filtering-HOWTO details iptables usage for packet filtering, the NAT-HOWTO details NAT, the netfilter-extensions-HOWTO details the extensions that are not in the standard distribution, and the netfilter-hacking-HOWTO details the netfilter internals.

See <http://www.netfilter.org/>.

AUTHORS

Rusty Russell wrote iptables, in early consultation with Michael Neuling.

Marc Boucher made Rusty abandon ipnatctl by lobbying for a

generic packet selection framework in iptables, then wrote the mangle table, the owner match, the mark stuff, and ran around doing cool stuff everywhere.

James Morris wrote the TOS target, and tos match.

Jozsef Kadlecik wrote the REJECT target.

Harald Welte wrote the ULOG target, TTL, DSCP, ECN matches and targets.

The Netfilter Core Team is: Marc Boucher, Martin Josefsson, Jozsef Kadlecik, James Morris, Harald Welte and Rusty Russell.

Man page written by Herve Eychenne <rv@wallfire.org>.

Mar 09, 2002

IPTABLES(8)

5 Appendix B: tc man pages:

TC(8)

Linux

TC(8)

NAME

tc - show / manipulate traffic control settings

SYNOPSIS

```
tc qdisc [ add | change | replace | link ] dev DEV [ parent qdisc-id | root ] [ handle qdisc-id ] qdisc [ qdisc specific parameters ]
```

```
tc class [ add | change | replace ] dev DEV parent qdisc-  
id [ classid class-id ] qdisc [ qdisc specific parameters  
]
```

```
tc filter [ add | change | replace ] dev DEV [ parent  
qdisc-id | root ] protocol protocol prio priority filter  
type [ filtertype specific parameters ] flowid flow-id
```

```
tc [-s | -d ] qdisc show [ dev DEV ]
```

```
tc [-s | -d ] class show dev DEV
```

```
tc filter show dev DEV
```

DESCRIPTION

Tc is used to configure Traffic Control in the Linux kernel. Traffic Control consists of the following:

SHAPING

When traffic is shaped, its rate of transmission is under control. Shaping may be more than lowering the available bandwidth - it is also used to smooth out bursts in traffic for better network behaviour. Shaping occurs on egress.

SCHEDULING

By scheduling the transmission of packets it is possible to improve interactivity for traffic that needs it while still guaranteeing bandwidth to bulk transfers. Reordering is also called prioritizing, and happens only on egress.

POLICING

Where shaping deals with transmission of traffic, policing pertains to traffic arriving. Policing thus occurs on ingress.

DROPPING

Traffic exceeding a set bandwidth may also be dropped forthwith, both on ingress and on egress.

Processing of traffic is controlled by three kinds of objects: qdiscs, classes and filters.

QDISCS

qdisc is short for 'queueing discipline' and it is elementary to understanding traffic control. Whenever the kernel needs to send a packet to an interface, it is enqueued to the qdisc configured for that interface. Immediately afterwards, the kernel tries to get as many packets as possible from the qdisc, for giving them to the network adaptor driver.

A simple QDISC is the 'pfifo' one, which does no processing at all and is a pure First In, First Out queue. It does however store traffic when the network interface can't handle it momentarily.

CLASSES

Some qdiscs can contain classes, which contain further qdiscs - traffic may then be enqueued in any of the inner qdiscs, which are within the classes. When the kernel tries to dequeue a packet from such a classful qdisc it

can come from any of the classes. A qdisc may for example prioritize certain kinds of traffic by trying to dequeue from certain classes before others.

FILTERS

A filter is used by a classful qdisc to determine in which class a packet will be enqueued. Whenever traffic arrives at a class with subclasses, it needs to be classified. Various methods may be employed to do so, one of these are the filters. All filters attached to the class are called, until one of them returns with a verdict. If no verdict was made, other criteria may be available. This differs per qdisc.

It is important to notice that filters reside within qdiscs - they are not masters of what happens.

CLASSLESS QDISCS

The classless qdiscs are:

[p|b]fifo

Simplest usable qdisc, pure First In, First Out behaviour. Limited in packets or in bytes.

pfifo_fast

Standard qdisc for 'Advanced Router' enabled kernels. Consists of a three-band queue which honors Type of Service flags, as well as the priority that may be assigned to a packet.

red Random Early Detection simulates physical congestion by randomly dropping packets when nearing configured bandwidth allocation. Well suited to very large bandwidth applications.

sfq Stochastic Fairness Queueing reorders queued traffic so each 'session' gets to send a packet in turn.

tbf The Token Bucket Filter is suited for slowing traffic down to a precisely configured rate. Scales well to large bandwidths.

CONFIGURING CLASSLESS QDISCS

In the absence of classful qdiscs, classless qdiscs can only be attached at the root of a device. Full syntax:

```
tc qdisc add dev DEV root QDISC QDISC-PARAMETERS
```

To remove, issue

```
tc qdisc del dev DEV root
```

The `pfifo_fast` qdisc is the automatic default in the absence of a configured qdisc.

CLASSFUL QDISCS

The classful qdiscs are:

CBQ Class Based Queueing implements a rich linksharing hierarchy of classes. It contains shaping elements as well as prioritizing capabilities. Shaping is performed using link idle time calculations based on average packet size and underlying link bandwidth. The latter may be ill-defined for some interfaces.

HTB The Hierarchy Token Bucket implements a rich linksharing hierarchy of classes with an emphasis on conforming to existing practices. HTB facilitates guaranteeing bandwidth to classes, while also

allowing specification of upper limits to inter-class sharing. It contains shaping elements, based on TBF and can prioritize classes.

PRIO The PRIO qdisc is a non-shaping container for a configurable number of classes which are dequeued in order. This allows for easy prioritization of traffic, where lower classes are only able to send if higher ones have no packets available. To facilitate configuration, Type Of Service bits are honored by default.

THEORY OF OPERATION

Classes form a tree, where each class has a single parent. A class may have multiple children. Some qdiscs allow for runtime addition of classes (CBQ, HTB) while others (PRIO) are created with a static number of children.

Qdiscs which allow dynamic addition of classes can have zero or more subclasses to which traffic may be enqueued.

Furthermore, each class contains a leaf qdisc which by default has pfifo behaviour though another qdisc can be attached in place. This qdisc may again contain classes, but each class can have only one leaf qdisc.

When a packet enters a classful qdisc it can be classified to one of the classes within. Three criteria are available, although not all qdiscs will use all three:

tc filters

If tc filters are attached to a class, they are consulted first for relevant instructions. Filters can match on all fields of a packet header, as well as on the firewall mark applied by ipchains or iptables. See tc-filters(8).

Type of Service

Some qdiscs have built in rules for classifying packets based on the TOS field.

skb->priority

Userspace programs can encode a class-id in the 'skb->priority' field using the SO_PRIORITY option.

Each node within the tree can have its own filters but higher level filters may also point directly to lower classes.

If classification did not succeed, packets are enqueued to the leaf qdisc attached to that class. Check qdisc specific manpages for details, however.

NAMING

All qdiscs, classes and filters have IDs, which can either be specified or be automatically assigned.

IDs consist of a major number and a minor number, separated by a colon.

QDISCS A qdisc, which potentially can have children, gets assigned a major number, called a 'handle', leaving the minor number namespace available for classes. The handle is expressed as '10:'. It is customary to explicitly assign a handle to qdiscs expected to have children.

CLASSES

Classes residing under a qdisc share their qdisc major number, but each have a separate minor number called a 'classid' that has no relation to their

parent classes, only to their parent qdisc. The same naming custom as for qdiscs applies.

FILTERS

Filters have a three part ID, which is only needed when using a hashed filter hierarchy, for which see `tc-filters(8)`.

UNITS

All parameters accept a floating point number, possibly followed by a unit.

Bandwidths or rates can be specified in:

kbps Kilobytes per second

mbps Megabytes per second

kbit Kilobits per second

mbit Megabits per second

bps or a bare number
 Bytes per second

Amounts of data can be specified in:

kb or k
 Kilobytes

mb or m
 Megabytes

mbit Megabits

kbit Kilobits

b or a bare number

Bytes.

Lengths of time can be specified in:

s, sec or secs

Whole seconds

ms, msec or msecs

Milliseconds

us, usec, usecs or a bare number

Microseconds.

TC COMMANDS

The following commands are available for qdiscs, classes and filter:

add Add a qdisc, class or filter to a node. For all entities, a parent must be passed, either by passing its ID or by attaching directly to the root of a device. When creating a qdisc or a filter, it can be named with the handle parameter. A class is named with the classid parameter.

remove A qdisc can be removed by specifying its handle, which may also be 'root'. All subclasses and their leaf qdiscs are automatically deleted, as well as any filters attached to them.

change Some entities can be modified 'in place'. Shares the syntax of 'add', with the exception that the handle cannot be changed and neither can the par

ent. In other words, change cannot move a node.

replace

Performs a nearly atomic remove/add on an existing node id. If the node does not exist yet it is created.

link Only available for qdiscs and performs a replace where the node must exist already.

HISTORY

tc was written by Alexey N. Kuznetsov and added in Linux 2.2.

SEE ALSO

tc-cbq(8), tc-htb(8), tc-sfq(8), tc-red(8), tc-tbf(8), tc-pfifo(8), tc-bfifo(8), tc-pfifo_fast(8), tc-filters(8)

AUTHOR

Manpage maintained by bert hubert (ahu@ds9a.nl)

iproute2

16 December 2001

TC(8)

6 Appendix C: A firewall sample.

Config file (/etc/firewall.cfg):

```
#!/bin/bash
# Some FW rules for this machine
```

```

# The real commands
IPT=/sbin/iptables
IPR=/sbin/ip
# To test the script
#IPT="echo /sbin/iptables"
#IPR="echo /sbin/ip"

sf=1 # Start filtering
sn=1 # Start nat
WRITE_LOG=1 # Write to syslog
# Information about Internet connection:
INTER="200.1.1.0/29"
IP_INTER="200.1.1.2"
IF_INTER=eth0
# Information about Intranet:
INTRA="192.1.68.0/24"
IP_INTRA="192.1.68.1"
IF_INTRA=eth2
# Information about WLAN:
WLAN="192.1.69.0/24"
IP_WLAN="192.1.69.1"
IF_WLAN=eth1
# PPP connections
PPPC="192.1.70.0/24"
IP_PPPC="192.1.70.1"
IF_PPPC=ppp0
# VM-WARE Lans
VMLAN="192.1.71.0/24"
# Proxy server we may use
PROXY_PORT="3128"
PROXY_IP="158.2.2.3"
PROXY="$PROXY_IP:$PROXY_PORT"
# Shurcuts:
LOCALNET="$INTRA $WLAN $PPPC $VMLAN"
# Inside servers
ISRV="MUROS CHERECK"
# Rules for those servers
MUROS_IP="200.1.1.3"
MUROS_TCPSERVICES="ssh http sftp"
MUROS_FROMINTER="rsync"
CHERECK_IP="200.1.1.4"
CHERECK_TCPSERVICES="ssh"
CHERECK_FROMINTER="rsync"
# TCP and UDP services we run and want to be accessible from outside
TCP_SERVICES="ftp ftp-data ssh sftp telnet domain
              smtp smtps pop3 pop3s imap imap3 imaps
              http https auth submission rsync 3128
              4000:7200 9696"
UDP_SERVICES="domain ntp 4662 8221"
# Ports we allow to be forwarded from the internet to other servers

```

```

TCPFORWARD="ftp ftp-data ssh pop3 imap imap3 https pgpkeyserver
$PROXY_PORT"
# Services local users may do
TCPOUT="ftp ftp-data ssh sftp telnet domain
smtp smtps pop3 pop3s imap imap3 imaps
http https auth submission rsync
x11 x11-ssh-offset
submission sunrpc cvspserver
6666 $PROXY_PORT"
UDPOUT="domain ntp snmp"
# Should access to the Intranet be allowed from the server
ALLOW_ACCESS_TO_LOCAL=0
# Should icmp be allowed to go out
ALLOW_ICMP_OUT=1
# My IP address at work
TCATWORK=212.1.1.2

```

The actual script (/usr/local/sbin/firewall):

```

#!/bin/bash

# Some FW rules for this machine

# Load configuration
. /etc/sendar.fw.cfg

start() {
    if [ $sf -eq 1 ] ; then
        start_filters
    fi
    if [ $sn -eq 1 ] ; then
        start_nat
    fi
}

#####
#   NAT   #
#####
start_nat() {
    modprobe ip_conntrack
    modprobe ip_conntrack_ftp

    # Flush all the rules
    $IPT -t nat -F PREROUTING
    $IPT -t nat -F POSTROUTING

    # Reroute telnet to MUD
    $IPT -t nat -I PREROUTING -p tcp --destination-port 23 -j REDIRECT --
to-ports 4000

    # Redirect all inside traffic through a proxy
    # $IPT -t nat -I PREROUTING -s $INTRA -d ! $MYNET -p tcp --dport 80 -j
DNAT --to-destination $PROXY

```

```

# Masquerade when going out
for SRC in $LOCALNET; do
    $IPT -t nat -A POSTROUTING -s $SRC -d ! $IP_INTER -j MASQUERADE
done
}

#####
# Filers #
#####
start_filters() {
    # Flush all the rules
    $IPT -F INPUT
    $IPT -F FORWARD
    $IPT -F OUTPUT
    # Policy deny all
    $IPT -P FORWARD DROP
    $IPT -P INPUT DROP
    $IPT -P OUTPUT DROP

    #----- INPUT -----

    # Accept me to do all I want from work
    $IPT -A INPUT -s $TCATWORK -j ACCEPT

    # Syn-flood protection:
    $IPT -A INPUT -p tcp --syn -m limit --limit 140/s -j ACCEPT

    # Furtive port scanner:
    $IPT -A INPUT -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit --limit
150/s -j ACCEPT

    # Ping of death:
    $IPT -A INPUT -p icmp -m limit --limit 5/s -j ACCEPT

    # Allow to talk to myself need more rules ?
    $IPT -A INPUT -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
    $IPT -A INPUT -s $IP_INTER -d $IP_INTER -j ACCEPT
    $IPT -A INPUT -s $IP_INTRA -d $IP_INTRA -j ACCEPT

    # Local services
    for PORT in $TCPSERVICES; do
        $IPT -A INPUT -p tcp --destination-port $PORT -j ACCEPT #
tcp
    done
    for PORT in $UDPSERVICES; do
        $IPT -A INPUT -p udp --destination-port $PORT -j ACCEPT #
udp
    done

    # Allow me to use my own server from home
    $IPT -A INPUT -i $IF_INTRA -s $INTRA -j ACCEPT

    # Allow dhcp from intranet and WLAN
    $IPT -A INPUT -i $IF_INTRA -p udp --destination-port 67:68 -j ACCEPT
    $IPT -A INPUT -i $IF_WLAN -p udp --destination-port 67:68 -j ACCEPT

```

```

# TODO add a list of MAC addresses which are on the local net

# Allow related and established traffic
$IPT -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Add some logging:
if [ $WRITE_LOG -eq 1 ] ; then
    $IPT -A INPUT -p udp --destination-port 137 -j DROP # Smaller
logs
    $IPT -A INPUT -p tcp --destination-port 137 -j DROP # Smaller
logs
    $IPT -A INPUT -p tcp --destination-port 138 -j DROP # Smaller
logs
    $IPT -A INPUT -p tcp --destination-port 139 -j DROP # Smaller
logs
    $IPT -A INPUT -j LOG --log-level notice --log-prefix "IPT-INPUT
"
fi
# --log-tcp-sequence --log-tcp-options --log-ip-options

#----- OUTPUT -----
# Here is what needs to go out

$IPT -A OUTPUT -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT # all local
$IPT -A OUTPUT -s $IP_INTER -d $IP_INTER -j ACCEPT

if [ $ALLOW_ACCESS_TO_LOCAL -eq 1 ]; then
    for SRC in $LOCALNET; do
        $IPT -A OUTPUT -d $INTRA -j ACCEPT # Only allow if
necessary
    done
fi

# allow me to do what I want
$IPT -A OUTPUT -m owner --uid-owner 500 -j ACCEPT
# Add rate limited icmp out
if [ $ALLOW_ICMP_OUT -eq 1 ]; then
    $IPT -A OUTPUT -p icmp -m limit --limit 3/s -j ACCEPT
fi

# Allow users to do some stuff
for PORT in $TCPOUT; do
    $IPT -A OUTPUT -p tcp --destination-port $PORT -j ACCEPT #
tcp
done
for PORT in $UDPOUT; do
    $IPT -A OUTPUT -p udp --destination-port $PORT -j ACCEPT #
udp
    $IPT -A OUTPUT -p udp --source-port $PORT -j ACCEPT # udp
done

# Protect other servers from ourself
$IPT -A OUTPUT -o $IF_INTER -s ! $INTER -j DROP

$IPT -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

```

```

# Log what should not get out
if [ $WRITE_LOG -eq 1 ] ; then
    $IPT -A OUTPUT -p udp --destination-port 137 -j DROP # Smaller
logs
    $IPT -A OUTPUT -p tcp --destination-port 137 -j DROP # Smaller
logs
    $IPT -A OUTPUT -p tcp --destination-port 138 -j DROP # Smaller
logs
    $IPT -A OUTPUT -p tcp --destination-port 139 -j DROP # Smaller
logs
    $IPT -A OUTPUT -j LOG --log-level notice --log-prefix "IPT-
OUTPUT " # --log-tcp-sequence
fi

#----- FORWARD -----

# Allow all traffic from the inside
$IPT -A FORWARD -i $IF_INTRA -s $INTRA -j ACCEPT

# Syn-flood protection:
$IPT -A FORWARD -p tcp --syn -m limit --limit 140/s -j ACCEPT

# Furtive port scanner:
$IPT -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit --
limit 150/s -j ACCEPT

# Ping of death:
$IPT -A FORWARD -p icmp -m limit --limit 3/s -j ACCEPT

# WLAN and other local servers with public IP have no more rights than
local users
for PORT in $TCPOUT; do
    $IPT -A FORWARD -i $IF_WLAN -s $WLAN -d ! $INTRA -p tcp --
destination-port $PORT -j ACCEPT
    $IPT -A FORWARD -s $INTER -d ! $INTRA -p tcp --destination-port
$PORT -j ACCEPT
    $IPT -A FORWARD -s $PPPC -d ! $INTRA -p tcp --destination-port
$PORT -j ACCEPT
done
for PORT in $UDPOUT; do
    $IPT -A FORWARD -i $IF_WLAN -s $WLAN -d ! $INTRA -p udp --
destination-port $PORT -j ACCEPT
    $IPT -A FORWARD -s $INTER -d ! $INTRA -p udp --destination-port
$PORT -j ACCEPT
    $IPT -A FORWARD -s $PPPC -d ! $INTRA -p udp --destination-port
$PORT -j ACCEPT
done
# Do not allow IP-spoofing from WLAN
$IPT -A FORWARD -i $IF_WLAN -s ! $WLAN -j LOG --log-level notice --
log-prefix "WLAN IP Spoofing "
$IPT -A FORWARD -i $IF_WLAN -s ! $WLAN -j DROP

# now allow other services to work for inside servers

```

```

for SRV in $ISRV; do
    IP=`eval echo '${SRV}_IP`
    SERVICES=`eval echo '${SRV}_TCPSERVICES`
    FROMINTER=`eval echo '${SRV}_FROMINTER`
    for PORT in $SERVICES; do
        $IPT -A FORWARD -d $IP -p tcp --destination-port $PORT -j
ACCEPT
        done
        for PORT in $FROMINTER; do
            $IPT -A FORWARD -s $INTER -d $IP -p tcp --destination-port
$PORT -j ACCEPT
        done
    done

    $IPT -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

    # Log what should not get through
    if [ $WRITE_LOG -eq 1 ] ; then
        $IPT -A FORWARD -p udp --destination-port 137 -j DROP # Smaller
logs
        $IPT -A FORWARD -p tcp --destination-port 137 -j DROP # Smaller
logs
        $IPT -A FORWARD -p tcp --destination-port 138 -j DROP # Smaller
logs
        $IPT -A FORWARD -p tcp --destination-port 139 -j DROP # Smaller
logs
        $IPT -A FORWARD -j LOG --log-level notice --log-prefix "IPT-
FORWARD "
    fi

    # Allow forwarding now
    echo 1 > /proc/sys/net/ipv4/ip_forward

    # Forward other public IP's to correct server
    echo 1 > /proc/sys/net/ipv4/conf/all/proxy_arp

    # We need to route to the internal servers

    for IP in $OTHER_SERVERS; do
        $IPR route add $IP dev $IF_INTRA
    done
}

stop() {
    $IPT -P INPUT ACCEPT
    $IPT -P OUTPUT ACCEPT
    $IPT -P FORWARD ACCEPT

    $IPT -F INPUT
    $IPT -F FORWARD
    $IPT -F OUTPUT

    # Remove nat too
    $IPT -t nat -F PREROUTING
    $IPT -t nat -F POSTROUTING
    $IPT -t nat -F OUTPUT

```

```
# allow ip forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward
}

case "$1" in
start)
    start
    ;;

stop)
    stop
    ;;

status)
    $IPT -L -n
    $IPT -L -n -t nat
    ;;

restart)
    stop
    start
    ;;

*)
    echo $"Usage: $0 {start|stop|restart|status}"
    exit 1

esac

exit $RETVAL
```
