



Linuxdays 2005

Netfilters and QOS

http://www.lilux.lu/presentations/2005/LinuxDays/Netfilters_QOS/

Thierry Coutelier<Thierry.Coutelier@lilux.lu>

Table of Contents

1	Netfilters.....	4
1.1	What are Netfilters.....	4
1.2	What you can do with Netfilters:.....	4
1.3	What is Needed.....	4
1.4	Network hooks and packet traversal.....	5
1.5	How to use the iptables.....	6
1.6	What is connection tracking?	7
1.7	Examples:.....	7
2	QOS.....	7
2.1	What is QOS.....	7
2.2	What you can do with QOS:.....	7
2.3	What is Needed.....	8
2.4	Kernel modules needed:.....	8
2.5	The tc commands.....	9
2.5.1	Queue disciplines:.....	10
2.5.2	Classes:.....	10
2.5.3	Filters:.....	10
2.6	Queuing disciplines.....	11
2.7	Usage with iptables.....	11
2.7.1	Using the Netfilter CLASSIFY Target.....	11
2.7.2	Using the Netfilter MARK Target.....	11
2.8	Testing tools.....	12
2.8.1	Netem.....	12
2.8.2	IPTraff.....	12
2.8.3	Netcat.....	12
2.8.4	iperf.....	12
2.9	Example.....	13
2.10	Exercise.....	13
3	Links.....	14
4	Appendix A: A firewall sample.....	14

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.

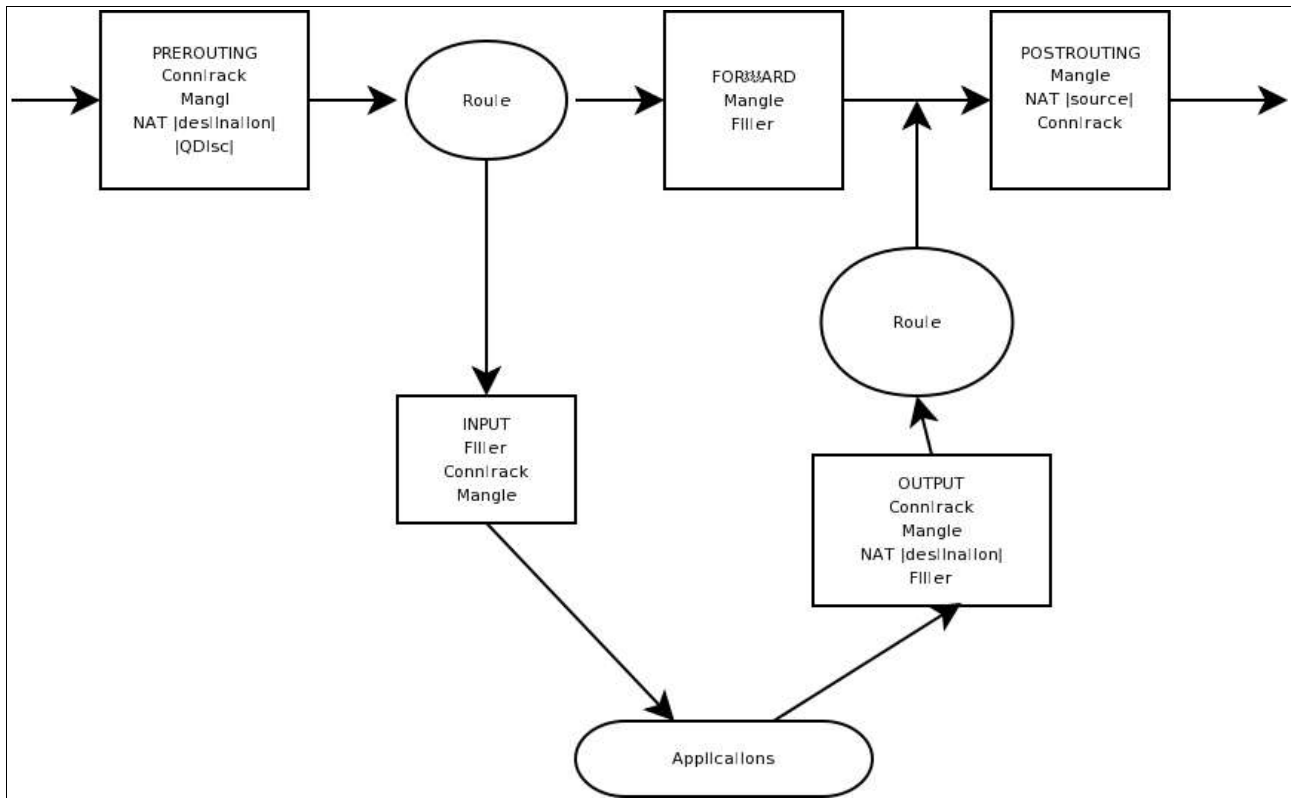
The netfilters need to be enabled in a Linux Kernel (version 2.2 or above)

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

1.4 Network hooks and packet traversal.



1.5 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.

1.6 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).

1.7 Examples:

See appendix A 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 PRIO 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)

```

2.5 The 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[atc] 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: 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
TCPSERVICES="ftp ftp-data ssh sftp telnet domain
             smtp smtps pop3 pop3s imap imap3 imaps
             http https auth submission rsync 3128
             4000:7200 9696"
UDPSERVICES="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

```
